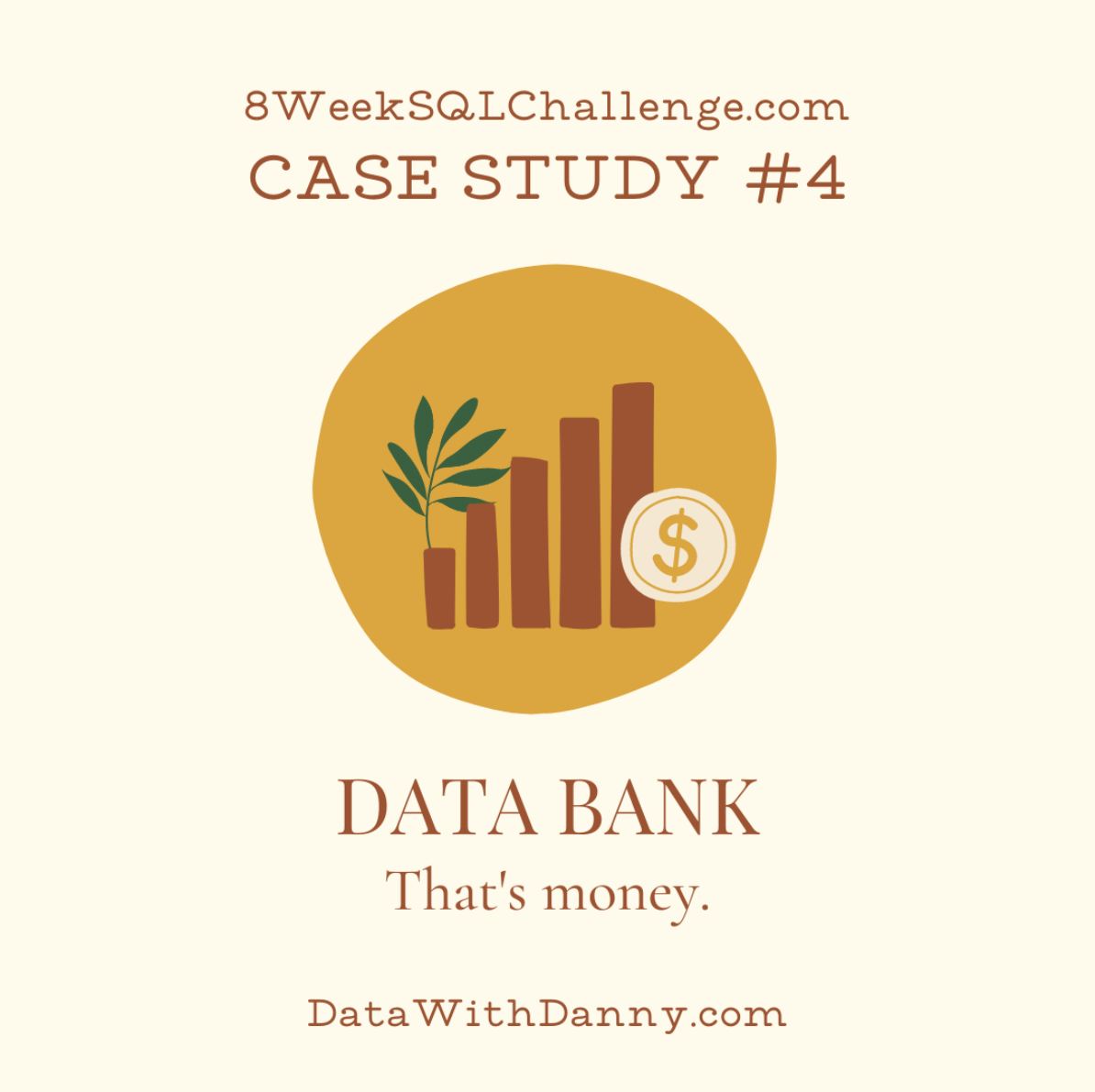
PIONEERING NEO-BANKING: AN ANALYTICAL REPORT ON DATA BANK'S PERFORMANCE METRICS

Fatih Sahin



https://8weeksqlchallenge.com/case-study-4/

CONTENTS

Introduction

This case study dives into the innovative world of Data Bank, a new-age neo-banking platform that seamlessly blends the realms of digital banking, cryptocurrency, and secure data storage. Unlike traditional banks, Data Bank offers cloud data storage limits that are directly proportional to the funds a customer holds in their account, presenting a unique model in the financial sector.

I embarked on this analytical journey after taking up the "Data Bank SQL Challenge", which provides a simulated environment to understand the intricacies of a neo-banking system. You can explore more about the challenge at https://8weeksqlchallenge.com/case-study-4/

Such challenges offer an engaging way to hone SQL skills while working on real-world-like business scenarios.

For this analysis, I employed the Fiddle DB platform to craft and validate my SQL queries. Everything presented in this report stems from the insights and expertise I've garnered through diligent SQL practice.

The primary aim of this report is not just to elucidate the findings and inferences drawn from the case study, but also to stand as a testament to the capabilities of SQL in driving business strategies. Through this, I hope to inspire and guide fellow SQL enthusiasts and professionals on the path to data-driven innovation.

Problem Statement

Danny, the visionary behind "Data Bank," aims to revitalize and expand his neo-banking platform by delving into critical business dimensions:

- Assessing customer allocation across various regions and nodes.
- Monitoring transactional behaviours, specifically deposits, withdrawals, and purchases.
- Gauging the efficiency and implications of data allocation strategies.
- Evaluating security measures and customer reallocation frequency to ensure optimal protection against potential breaches.

The case study is meticulously segmented, each focusing on distinct aspects of the business:

- **Regional Allocation Metrics**: Understand the distribution of customers across regions, ensuring balanced and efficient service delivery.
- **Transactional Insights**: Scrutinize the patterns and frequencies of various transaction types, understanding customer financial behaviours and the potential for upselling or cross-selling services.
- **Data Allocation Analysis**: Evaluate the efficacy of different data allocation methods, from end-of-month balances to real-time updates, and determine the most cost-effective and customer-centric approach.
- **Security and Innovation**: Address the frequency of customer reallocation across nodes as a security measure and brainstorm potential enhancements to this system.

Creating Schema and Tables

All datasets exist within the `data_bank` database schema.

Schema SQL

```sql
CREATE SCHEMA data_bank;
SET search_path = data_bank;

CREATE TABLE regions (
  region_id INTEGER,
  region_name VARCHAR(9)
);

INSERT INTO regions
  (region_id, region_name)
VALUES
  ('1', 'Australia'),
  ('2', 'America'),
  ('3', 'Africa'),
  ('4', 'Asia'),
  ('5', 'Europe');


CREATE TABLE customer_nodes (
  customer_id INTEGER,
  region_id INTEGER,
  node_id INTEGER,
  start_date DATE,
  end_date DATE
);

INSERT INTO customer_nodes
  (customer_id, region_id, node_id, start_date, end_date)
VALUES
  ('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'),
  ('11', '2', '5', '2020-01-19', '2020-01-25'),
  ('12', '1', '2', '2020-01-13', '2020-01-14'),
  ('13', '2', '3', '2020-01-02', '2020-01-14'),
  ('14', '1', '2', '2020-01-25', '2020-01-25'),
  ('15', '1', '3', '2020-01-25', '2020-02-08'),
  ('16', '4', '4', '2020-01-13', '2020-01-18'),
  ('17', '2', '3', '2020-01-19', '2020-01-27'),
  ('18', '1', '3', '2020-01-17', '2020-02-15'),
  ('19', '2', '2', '2020-01-17', '2020-02-06'),
  ('20', '2', '4', '2020-01-18', '2020-02-09'),
  ('21', '3', '2', '2020-01-12', '2020-01-25'),
  ('22', '4', '3', '2020-01-19', '2020-02-18'),
  ('23', '1', '5', '2020-01-21', '2020-02-15'),
  ('24', '2', '5', '2020-01-26', '2020-02-14'),
  ('25', '5', '1', '2020-01-28', '2020-02-10'),
  ('26', '3', '3', '2020-01-17', '2020-01-25'),
  ('27', '4', '3', '2020-01-01', '2020-01-22'),
  ('28', '4', '2', '2020-01-20', '2020-02-13'),
  ('29', '2', '3', '2020-01-19', '2020-01-31'),
  ('30', '2', '1', '2020-01-26', '2020-02-06'),
  ('31', '3', '2', '2020-01-06', '2020-01-13'),
  ('32', '4', '4', '2020-01-12', '2020-01-14'),
  ('33', '3', '4', '2020-01-24', '2020-02-17'),
  ('34', '2', '3', '2020-01-30', '2020-02-18'),
  ('35', '4', '5', '2020-01-17', '2020-02-15'),
  ('36', '2', '3', '2020-01-30', '2020-02-19'),
  ('37', '1', '3', '2020-01-29', '2020-02-04'),
  ('38', '5', '1', '2020-01-21', '2020-02-14'),
  ('39', '5', '2', '2020-01-22', '2020-01-23'),
  ('40', '2', '1', '2020-01-21', '2020-02-17'),
  ('41', '5', '1', '2020-01-30', '2020-02-26'),
```

Entity Relationship Diagram

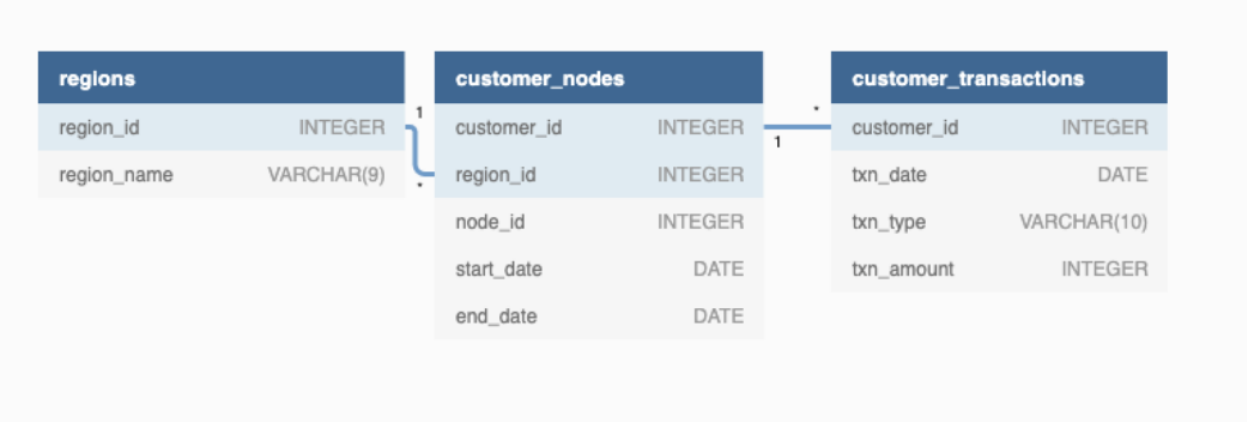
Data Structure

Data Bank manages its complex ecosystem of customer data, transactions, and regional nodes using several interconnected tables:

- **Regions**: This table holds information about the various geographical regions where Data Bank operates. Each region is uniquely identified by a region_id and is associated with a region_name that denotes the geographical area, such as Africa, America, Asia, etc.
- **Customer Nodes**: An integral part of Data Bank's security protocol, this table maps customers to specific nodes in various regions. Each record contains a customer_id indicating the unique customer, the region_id denoting the region where the node is located, and a node_id that identifies the specific node. Additionally, the start_date and end_date fields capture the duration during which a customer is associated with a particular node.
- **Customer Transactions**: This table chronicles all transactions undertaken by the customers. Each transaction is linked to a customer_id, the date of the transaction (txn_date), the type of transaction (txn_type such as deposit, withdrawal, or purchase), and the transaction amount (txn_amount).

The structure and relationships among these tables enable Data Bank to efficiently manage its operations, ensure security, and facilitate smooth transactions for its customers.

Entity Relationship Diagram



A. Customer Nodes Exploration

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

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

count
5

2. What is the number of nodes per region?

```
SELECT region_id, COUNT(DISTINCT node_id) AS
num_nodes
FROM data_bank.customer_nodes
GROUP BY region_id
ORDER BY num_nodes DESC;
```

region_id	num_nodes
1	5
2	5
3	5
4	5
5	5

3. How many customers are allocated to each region?

```
SELECT region_id, COUNT(DISTINCT customer_id)
FROM data_bank.customer_nodes
GROUP BY region_id
ORDER BY region_id
```

region_id	count
1	110
2	105
3	102
4	95
5	88

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

```sql
SELECT ROUND(AVG(end_date - start_date),2) AS
avg_reallocation_days
FROM data_bank.customer_nodes
WHERE (end_date - start_date) >= 0 AND
(end_date - start_date) <= 29;
```

avg_reallocation_days
14.55

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

```sql
-- The query might differ based on the SQL dialect you are using
SELECT
    region_id,
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY duration) AS median_duration,
    PERCENTILE_CONT(0.8) WITHIN GROUP (ORDER BY duration) AS p80_duration,
    PERCENTILE_CONT(0.95) WITHIN GROUP (ORDER BY duration) AS p95_duration
FROM (
    SELECT
        region_id,
        end_date - start_date AS duration
    FROM
        data_bank.customer_nodes
    WHERE
        (end_date - start_date) >= 0 AND (end_date - start_date) <= 28
) AS subquery
GROUP BY region_id;
```

region_id	median_duration	p80_duration	p95_duration
1	14	22	27
2	15	23	27
3	14	23	26.850000000000023
4	14	22	27
5	14	23	27

B. Customer Transactions

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

```sql
SELECT
    txn_type,
    COUNT(*) AS unique_count,
    SUM(txn_amount) AS total_amount
FROM
    data_bank.customer_transactions
GROUP BY
    txn_type;
```

txn_type	unique_count	total_amount
purchase	1617	806537
deposit	2671	1359168
withdrawal	1580	793003

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

```sql
SELECT
    ROUND(AVG(deposit_count),2) AS avg_deposit_count,
    ROUND(AVG(total_deposit_amount),2) AS avg_deposit_amount
FROM (
    SELECT
        customer_id,
        COUNT(*) AS deposit_count,
        SUM(txn_amount) AS total_deposit_amount
    FROM
        data_bank.customer_transactions
    WHERE
        txn_type = 'deposit'
    GROUP BY
        customer_id
) AS subquery;
```

avg_deposit_count	avg_deposit_amount
5.34	2718.34

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
SELECT
    EXTRACT(YEAR FROM txn_date) AS year,
    EXTRACT(MONTH FROM txn_date) AS month,
    COUNT(DISTINCT CASE WHEN txn_type = 'deposit' THEN customer_id ELSE NULL END) AS customers_with_multiple_deposits,
    COUNT(DISTINCT CASE WHEN txn_type IN ('purchase', 'withdrawal') THEN customer_id ELSE NULL END) AS customers_with_purchase_or_withdrawal
FROM
    data_bank.customer_transactions
GROUP BY
    year, month
HAVING
    COUNT(DISTINCT CASE WHEN txn_type = 'deposit' THEN customer_id ELSE NULL END) > 1
    AND (COUNT(DISTINCT CASE WHEN txn_type = 'purchase' THEN customer_id ELSE NULL END) >= 1
         OR COUNT(DISTINCT CASE WHEN txn_type = 'withdrawal' THEN customer_id ELSE NULL END) >= 1);

```

year	month	customers_with_multiple_deposits	customers_with_purchase_or_withdrawal
2020	1	500	295
2020	2	350	403
2020	3	372	413
2020	4	204	243

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

```sql
WITH MonthlyTransactions AS (
    SELECT
        customer_id,
        EXTRACT(YEAR FROM txn_date) AS year,
        EXTRACT(MONTH FROM txn_date) AS month,
        SUM(CASE WHEN txn_type = 'deposit' THEN txn_amount ELSE 0 END) -
        SUM(CASE WHEN txn_type IN ('purchase', 'withdrawal') THEN txn_amount ELSE 0 END) AS monthly_balance_change
    FROM
        data_bank.customer_transactions
    GROUP BY
        customer_id, year, month
),
MonthlyClosingBalances AS (
    SELECT
        customer_id,
        year,
        month,
        SUM(monthly_balance_change) OVER (PARTITION BY customer_id ORDER BY year, month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS closing_balance
    FROM
        MonthlyTransactions
)
SELECT
    customer_id,
    year,
```

customer_id	year	month	closing_balance
1	2020	1	312
1	2020	3	-640
2	2020	1	549
2	2020	3	610
3	2020	1	144
3	2020	2	-821
3	2020	3	-1222
3	2020	4	-729
4	2020	1	848
4	2020	3	655
5	2020	1	954
5	2020	3	-1923

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

```sql
CREATE TEMP TABLE MonthlyClosingBalances AS
WITH MonthlyTransactions AS (
    SELECT
        customer_id,
        EXTRACT(YEAR FROM txn_date) AS year,
        EXTRACT(MONTH FROM txn_date) AS month,
        SUM(CASE WHEN txn_type = 'deposit' THEN txn_amount ELSE 0 END) -
        SUM(CASE WHEN txn_type IN ('purchase', 'withdrawal') THEN txn_amount ELSE 0 END) AS monthly_balance_change
    FROM
        data_bank.customer_transactions
    GROUP BY
        customer_id, year, month
)
SELECT
    customer_id,
    year,
    month,
    SUM(monthly_balance_change) OVER (PARTITION BY customer_id ORDER BY year, month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS closing_balance
FROM
    MonthlyTransactions;

WITH PercentageChange AS (
    SELECT
        customer_id,
        year,
        month,
        closing_balance,
        LAG(closing_balance) OVER (PARTITION BY customer_id ORDER BY year, month) AS previous_month_closing_balance,
        (closing_balance - LAG(closing_balance) OVER (PARTITION BY customer_id ORDER BY year, month)) / NULLIF(LAG(closing_balance) OVER (PARTITION BY customer_id ORDER BY year, month), 0) AS balance_percentage_increase
    FROM
        MonthlyClosingBalances
)
SELECT
    COUNT(DISTINCT customer_id) * 100.0 / (SELECT COUNT(DISTINCT customer_id) FROM data_bank.customer_transactions) AS percentage_customers_with_increased_balance
FROM
    PercentageChange
WHERE
    balance_percentage_increase > 0.05;
```

percentage_customers_with_increased_balance
75.80

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?

Option 1: Based on Previous Month's End Balance

- The data allocation for a customer in a given month is determined by their account balance at the end of the previous month.
- Example: If a customer had $10,000 in their account at the end of January, then their data allocation for February would be based on this amount.

```sql
WITH MonthlyTransactions AS (
    -- Calculate monthly balance change for each customer.
    SELECT
        customer_id,
        EXTRACT(YEAR FROM txn_date) AS year,
        EXTRACT(MONTH FROM txn_date) AS month,
        SUM(CASE WHEN txn_type = 'deposit' THEN txn_amount ELSE 0 END) -
        SUM(CASE WHEN txn_type IN ('purchase', 'withdrawal') THEN txn_amount ELSE 0 END) AS monthly_balance_change
    FROM
        data_bank.customer_transactions
    GROUP BY
        customer_id, year, month
),
MonthlyClosingBalances AS (
    -- Calculate closing balance for each customer at the end of each month.
    SELECT
        customer_id,
        year,
        month,
        SUM(monthly_balance_change) OVER (PARTITION BY customer_id ORDER BY year, month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS closing_balance
    FROM
        MonthlyTransactions
),
DataAllocation AS (
    -- Allocate data storage based on the closing balance.
    SELECT
        customer_id,
        year,
        CASE
            WHEN month = 12 THEN 1
            ELSE month + 1
        END AS allocation_month,
        CASE
            WHEN month = 12 THEN year + 1
            ELSE year
        END AS allocation_year,
        GREATEST(CEIL(closing_balance / 1000), 0) AS allocated_data_in_GB -- Ensuring non-negative data allocation.
    FROM
        MonthlyClosingBalances
```

customer_id	allocation_year	allocation_month	allocated_data_in_gb
1	2020	2	1
1	2020	4	0
2	2020	2	1
2	2020	4	1
3	2020	2	1
3	2020	3	0
3	2020	4	0
3	2020	5	0
4	2020	2	1
4	2020	4	1
5	2020	2	1
5	2020	4	0
5	2020	5	0
6	2020	2	1
6	2020	3	0
6	2020	4	1
7	2020	2	1
7	2020	3	4

- The data allocation for a customer in a given month is based on the average balance they maintained over the previous 30 days.
- Example: If a customer's average balance over the last 30 days (from, say, January 2nd to January 31st) is $8,000, then their data allocation for February would be based on this average.

```sql
WITH RunningBalances AS (
    -- Calculate a running balance for each customer.
    SELECT
        customer_id,
        txn_date,
        SUM(
            CASE
                WHEN txn_type = 'deposit' THEN txn_amount
                WHEN txn_type IN ('purchase', 'withdrawal') THEN -txn_amount
                ELSE 0
            END
        ) OVER (PARTITION BY customer_id ORDER BY txn_date) AS running_balance
    FROM
        data_bank.customer_transactions
),
AverageBalances AS (
    -- Calculate the 30-day average balance for each day.
    SELECT
        a.customer_id,
        a.txn_date,
        AVG(b.running_balance) AS avg_balance_last_30_days
    FROM
        RunningBalances a
    JOIN
        RunningBalances b ON a.customer_id = b.customer_id AND b.txn_date BETWEEN a.txn_date - INTERVAL '30 days' AND a.txn_date
    GROUP BY
        a.customer_id, a.txn_date
),
MonthlyDataAllocation AS (
    -- Determine data allocation for each month based on the 30-day average of the last day of the month.
    SELECT
        customer_id,
        EXTRACT(YEAR FROM txn_date) AS year,
        EXTRACT(MONTH FROM txn_date) AS month,
        GREATEST(CEIL(avg_balance_last_30_days / 1000), 0)
 AS allocated_data_in_GB
    FROM
        AverageBalances
    WHERE
        txn_date = DATE_TRUNC('MONTH', txn_date + INTERVAL '1 month') - INTERVAL '1 day' -- This logic gets the last day of each month
)
SELECT
    customer_id,
    year,
    month,
    allocated_data_in_GB
FROM
    MonthlyDataAllocation
ORDER BY
    customer_id, year, month;

```

customer_id	year	month	allocated_data_in_gb
5	2020	1	2
5	2020	3	0
6	2020	3	1
7	2020	2	2
11	2020	3	0
22	2020	2	0
23	2020	3	1
24	2020	2	1
29	2020	1	0
41	2020	1	1
42	2020	2	1
43	2020	1	1
45	2020	3	0
46	2020	1	1
47	2020	1	0
47	2020	3	0
48	2020	2	0
50	2020	1	1
51	2020	2	0
60	2020	3	1
61	2020	2	1
61	2020	3	0
63	2020	3	0
70	2020	1	1

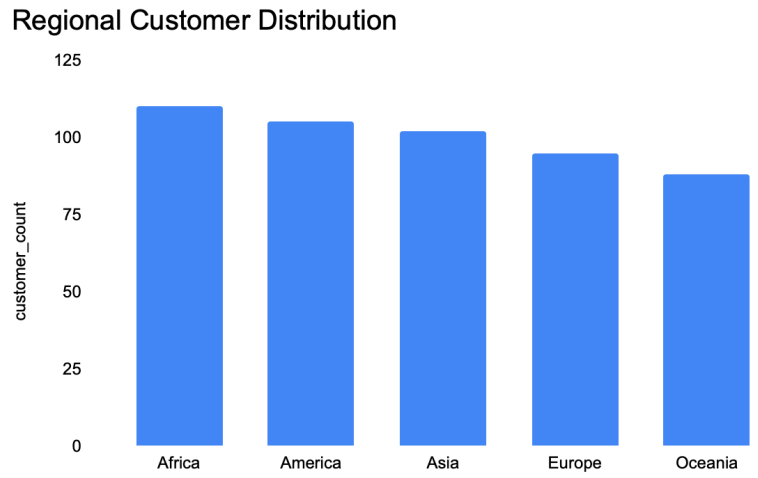
Option 3: Real-time Update

- The data allocation is updated in real-time, i.e., it changes dynamically as a customer's balance changes due to deposits, withdrawals, or purchases.
- Example: If a customer makes a deposit today, their data allocation would immediately increase proportionately.

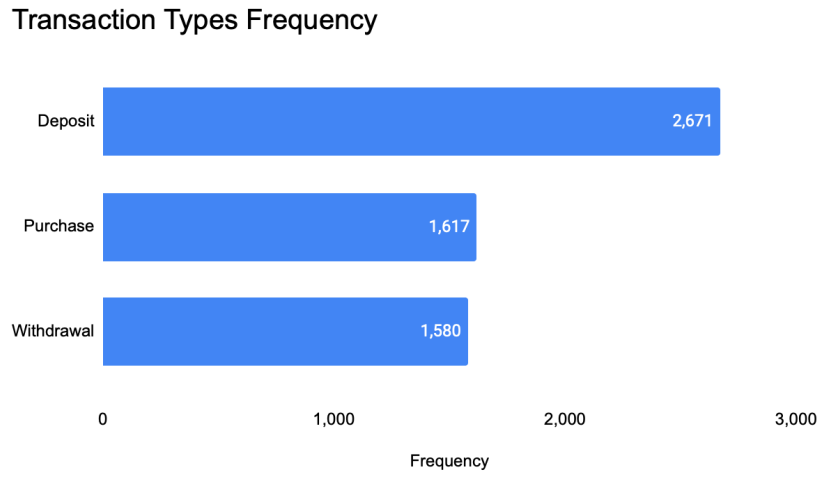
```sql
WITH RunningBalances AS (
    -- Calculate a running balance for each customer after each transaction.
    SELECT
        customer_id,
        txn_date,
        SUM(
            CASE
                WHEN txn_type = 'deposit' THEN txn_amount
                WHEN txn_type IN ('purchase', 'withdrawal') THEN -txn_amount
                ELSE 0
            END
        ) OVER (PARTITION BY customer_id ORDER BY txn_date) AS running_balance
    FROM
        data_bank.customer_transactions
),
RealTimeDataAllocation AS (
    -- Determine data allocation in real-time based on the running balance after each transaction.
    SELECT
        customer_id,
        txn_date,
        GREATEST(CEIL(running_balance / 1000), 0) AS allocated_data_in_GB
    FROM
        RunningBalances
)
SELECT
    customer_id,
    txn_date,
    allocated_data_in_GB
FROM
    RealTimeDataAllocation
ORDER BY
    customer_id, txn_date;
```

customer_id	txn_date	allocated_data_in_gb
1	2020-01-02T00:00:00.000Z	0
1	2020-03-05T00:00:00.000Z	0
1	2020-03-17T00:00:00.000Z	0
1	2020-03-19T00:00:00.000Z	0
2	2020-01-03T00:00:00.000Z	0
2	2020-03-24T00:00:00.000Z	0
3	2020-01-27T00:00:00.000Z	0
3	2020-02-22T00:00:00.000Z	0
3	2020-03-05T00:00:00.000Z	0
3	2020-03-19T00:00:00.000Z	0
3	2020-04-12T00:00:00.000Z	0
4	2020-01-07T00:00:00.000Z	0
4	2020-01-21T00:00:00.000Z	0
4	2020-03-25T00:00:00.000Z	0
5	2020-01-15T00:00:00.000Z	0
5	2020-01-25T00:00:00.000Z	1
5	2020-01-31T00:00:00.000Z	0
5	2020-03-02T00:00:00.000Z	0
5	2020-03-19T00:00:00.000Z	0
5	2020-03-26T00:00:00.000Z	0
5	2020-03-27T00:00:00.000Z	0
5	2020-03-27T00:00:00.000Z	0

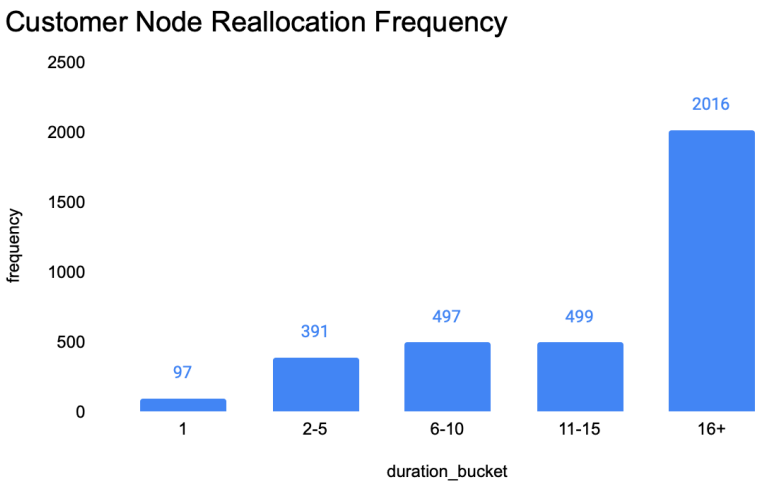
Key Findings and Observations



The customer distribution is **relatively uniform across regions**, with **Africa** having the **highest** number of customers. This balanced distribution indicates a global trust in Data Bank's services and suggests a potential for targeted marketing in regions with slightly lower numbers, such as Oceania.



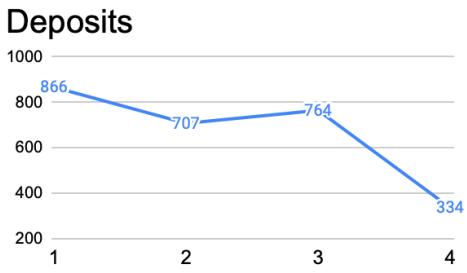
Deposits outnumber withdrawals and purchases, reflecting a trust in the bank's services and highlighting a potential to introduce more purchase incentives.



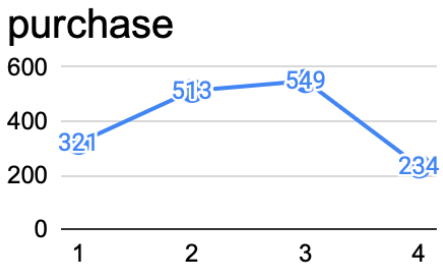
Most customers are **reallocated** between **10 to 20 days,** showcasing the bank's robust security measures.

This distribution underscores Data Bank's dynamic reallocation strategy, balancing both short-term and long-term node assignments to enhance security and operational efficiency. The wide variety of reallocation durations might be a part of Data Bank's unpredictable strategy to thwart potential security breaches.

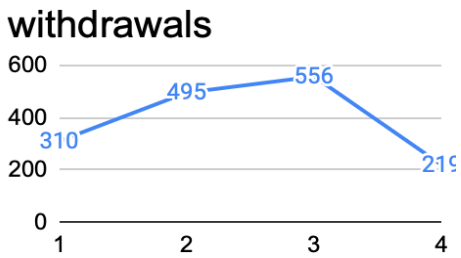
Customer Transaction Trends Over Time



There was a **peak** in deposits in **March** 2020 with 764 transactions totaling 390,103 units. However, this **dropped significantly in April** to just 334 transactions.



Purchases saw an **increasing** trend from **January to March**, with the highest being in March with 549 transactions. This again saw a **drop** in **April**.



The trend for withdrawals was quite **similar** to purchases with a **peak** in **March**.

<u>Overall</u>: **March** seemed to be the **busiest month** across all transaction types, but there was a **noticeable dip in all activities in April.**

CONCLUSION

The Data Bank case study presented an intriguing exploration into the dynamics of **neo-banking transactions** and **customer behavior**. Through this endeavour, I harnessed the capabilities of SQL to extract, analyze, and interpret vast datasets, revealing key insights that could shape the future strategies of neo-banks.

Utilizing **DB Fiddle**, I managed to seamlessly run complex SQL queries, transforming raw data into coherent structures ready for analysis. The subsequent phase of visualizing this data was accomplished using Google Sheets, which enabled the conversion of numeric data into intuitive visual representations. These visuals painted a clearer picture of **transactional trends, customer preferences**, and **banking anomalies,** offering a comprehensive overview of the neo-banking landscape.

In conclusion, this project showcased the importance of SQL in the world of data analytics and emphasized the role of effective visualization in conveying complex data narratives. The insights derived from this case study underscore the potential of data-driven decision-making in shaping the future of the **banking industry.**