TECHNICAL TEST

Enclosed you will find a dataset of bling accounts with a balance for some dates:

- bba_id : the bling bank account id
- bba_updated_at : date at which the bling bank account was updated
- bba_balance : balance registered at the above date

Task : Calculate the balance of each account every day so that it could be easily displayed in a visualization tool (eg graph in Excel). The balance is not registered every day so you will need to reconstitute it.

Candidate Name: Chidiebere Ogbuchi

Solution using PostgreSQL:

This query processes daily balances in the following steps:

1. **Filter Duplicates**: Keeps only the latest balance for each `bba_id` on each day.
2. **Date Range Creation**: Generates a full range of dates from the minimum to maximum date in the data.
3. **Cross Join**: Combines each `bba_id` with every date in the range, ensuring every `bba_id` has an entry for each date.
4. **Join Balances**: Left joins existing balances with these combinations, leaving gaps as NULL where balances are missing.
5. **Grouping for Forward Fill**: Groups non-null balances to prepare for forward filling.
6. **Forward Fill**: Propagates the last known balance forward to fill in gaps.
7. Select Final data

```
-- Step 1: Filter daily duplicates to keep the latest balance for each bba_id and day, and create datekey
WITH latest_daily_balance AS (
  SELECT
    bba_id,
    bba_updated_at::date AS date,  -- Convert timestamp to date
    bba_balance,
        ROW_NUMBER() OVER (PARTITION BY bba_id, bba_updated_at::date ORDER BY bba_updated_at DESC) AS row_num  -- Assign row number based on date
```

```
    FROM bling_accounts
),

-- Step 2: Keep only the latest entry per day for each bba_id
filtered_balance AS (
    SELECT
        bba_id,
        date,
        bba_balance  -- Include datekey here
    FROM latest_daily_balance
    WHERE row_num = 1  -- Filter to keep only the latest entry
),

-- Step 3: Create a date range for all dates from the minimum to maximum date in filtered_balance
date_range AS (
    SELECT
        generate_series(
            (SELECT MIN(date) FROM filtered_balance),  -- Minimum date
            (SELECT MAX(date) FROM filtered_balance),  -- Maximum date
            '1 day'::interval  -- Increment by one day
        )::date AS date  -- Cast to date type
),

-- Step 4: Get all combinations of bba_id and each date in the range
all_dates_per_account AS (
    SELECT DISTINCT
        f.bba_id,
        d.date
    FROM
        (SELECT DISTINCT bba_id FROM filtered_balance) AS f  -- Distinct bba_ids
    CROSS JOIN
        date_range AS d  -- Cross join to create all combinations of bba_id and dates
),

-- Step 5: Left join to bring in the existing balances for each bba_id and date
joined_data AS (
    SELECT
        a.bba_id,
        a.date,
```

```
    f.bba_balance  -- Include balance for matching bba_id and date
  FROM
    all_dates_per_account AS a
  LEFT JOIN
    filtered_balance AS f
  ON
    a.bba_id = f.bba_id AND a.date = f.date  -- Match on bba_id and date
),

-- Step 6: Create a grouping for forward fill
reconstitution_data AS (
  SELECT
    *,
     COUNT(bba_balance) OVER (PARTITION BY bba_id ORDER BY date) AS Grp
-- Create group based on bba_id and order by date
  FROM
    joined_data
)

-- Step 7: Final selection with forward filling of balances
SELECT
  bba_id,
  date,
  bba_balance,
    FIRST_VALUE(bba_balance) OVER (PARTITION BY bba_id, Grp ORDER BY
date) AS bba_balance_fill  -- Forward fill the balances
FROM
  reconstitution_data;
```

Solution using Python:

Summary of the steps we took to calculate daily balances for each account and prepare the data for analysis:

1. **Data Preparation and Initial Cleaning**

   - Loaded the dataset and examined the columns: `bba_id`, `bba_balance`, `bba_updated_at`

```
[5] df.describe(include='all')
```

	bba_id	bba_balance	bba_updated_at
count	3139	3139.000000	3139
unique	100	NaN	3139
top	00f1255b-83c7-44f8-9254-21dc9c369e65	NaN	2024-01-09 23:10:39.399853+00
freq	173	NaN	1
mean	NaN	178.535081	NaN
std	NaN	337.704236	NaN
min	NaN	0.000000	NaN
25%	NaN	0.895000	NaN
50%	NaN	37.400000	NaN
75%	NaN	212.930000	NaN
max	NaN	4899.490000	NaN

- Extracted date hierarchies (year, month, day) from `bba_updated_at` and created a `date` column for ease of grouping and consistency.

- Sorted the dataset by `bba_updated_at` to ensure that the latest balance entry for each account per day could be retained.

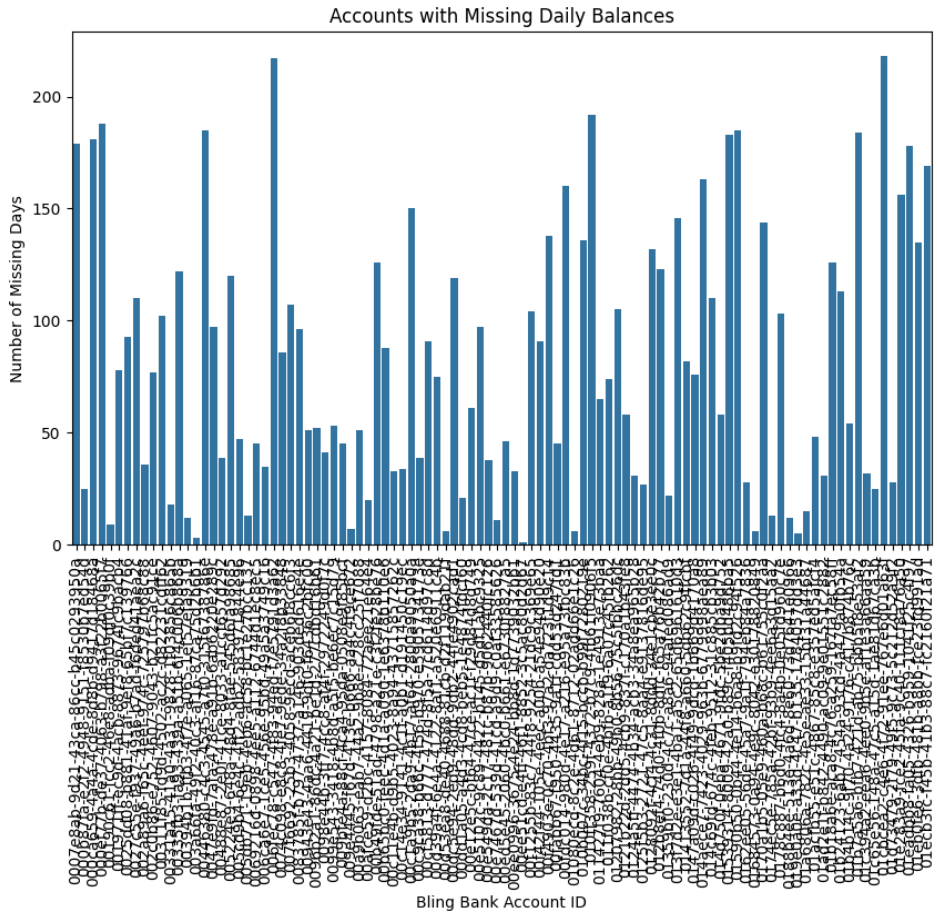
- Removed duplicate entries within each `bba_id` and `date` combination, keeping the most recent balance for each day.

2. Reconstituting Dataset

A. Generate Daily Entries for Each Account

- Investigated the date range from the earliest date of an id to the latest date in the dataset, assuming that the first occurrence of a user balance is when they are registered.

	bba_id	recorded_dates	start_date	expected_dates	missing_dates
0	0007e8ab-9d21-43ca-86cc-b45c0293950a	109	2024-01-09	288	179
1	000b87fa-e2f2-4948-9151-f8530b7ed54d	30	2024-08-29	55	25
2	000ca659-4a4a-4cde-8d8b-d9417d18468a	3	2024-04-22	184	181
3	000ef77b-de37-4063-b798-a3f22b0065f7	3	2024-04-15	191	188
4	00190c61-fcc2-46e8-adba-f09df0c39b0f	4	2024-10-10	13	9



- Generated a new dataframe, `df_fill`, with an entry for each `bba_id` based on min to max date range

B. Merge Dataframes and Fill Missing Values

- Merged the original dataset (`df`) with `df_fill` on `bba_id` and `date` to combine recorded balances with the daily entries for each account.

- Forward-filled missing balance values within each `bba_id` to propagate the last known balance forward, ensuring a continuous daily balance history.

- Leaving missing values that may have occurred before the first recorded balance for each account assuming they were not registered then.

.

Validating Reconstitution

3. Extract Date Hierarchies and Perform Validation

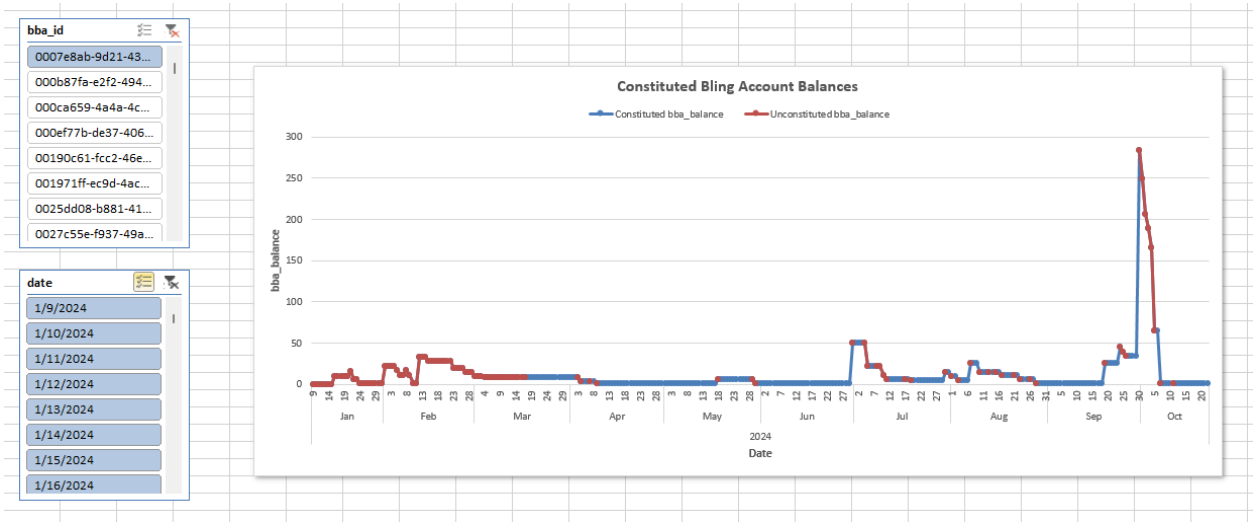
- Extracted additional date hierarchies (day of week, week of year, quarter, etc.) to support time-based analysis and aggregation.

- Conducted checks to ensure the fill was applied in the correct chronological order and that all dates were correctly populated for each account.

Visualization

4. Exploratory Data Analysis (EDA) and Visualization Preparation

- Calculating summary statistics for each account, such as average and total balance over the recorded period can be done
- Preparing a sample visualization of a single account's balance trend over time, with code that allows selection of different accounts for visual inspection.
- Organized the reconstituted data for export to a CSV and xlsx file, facilitating further analysis or use in visualization tools like Excel or Power BI.



In this workflow, I built a comprehensive constituted daily balance history for each account, utilizing python notebook for analysis and and Excel for visualization.

Assumptions

- Since I am interested in final balance at the end of a day, we keep latest value for each day and remove other duplicates.
- I consider the only available range to be the min for the account and max bba_updated_at.
- To reconstitute the balance for each account for every day, I filled in the missing dates with the most recent balance before that date. Considering if no data is recorded on a specific date, I assume the balance from the most recent previous date persists until the next update.