Bank Statement Processing Application: Implementation Report

1. Project Overview

This report details the implementation of a Python-based application designed to process bank account statements. The primary objective was to create a simple yet effective pipeline to upload statement files, store the transaction data in a structured database, and provide a user-friendly interface to view and filter this data.

The project was divided into two main components:

1. A **data ingestion tool** for uploading and parsing statement files.
2. A **web-based dashboard** for data visualization and exploration.

2. Core Components & Technologies

2.1. Data Ingestion: uploader_notebook.ipynb

The data ingestion component was developed as a Jupyter Notebook to provide an interactive and accessible environment for file processing.

- **Technology Used:** pandas for data manipulation, ipywidgets for the interactive file upload functionality, and sqlite3 for database communication.
- **Functionality:**
    - An interactive "Upload Statement" button allows the user to select a local .csv or .xlsx file.
    - The notebook includes a robust parsing function, parse_statement, which can intelligently detect the format of different bank statements (tested with two distinct formats). It achieves this by mapping the unique column names of each bank's statement to a standardized database schema (transaction_date, description, etc.).
    - Once parsed, the cleaned transaction data is committed to the database.

2.2. Data Storage: transactions.db

A simple and serverless database solution was chosen to store the transaction data.

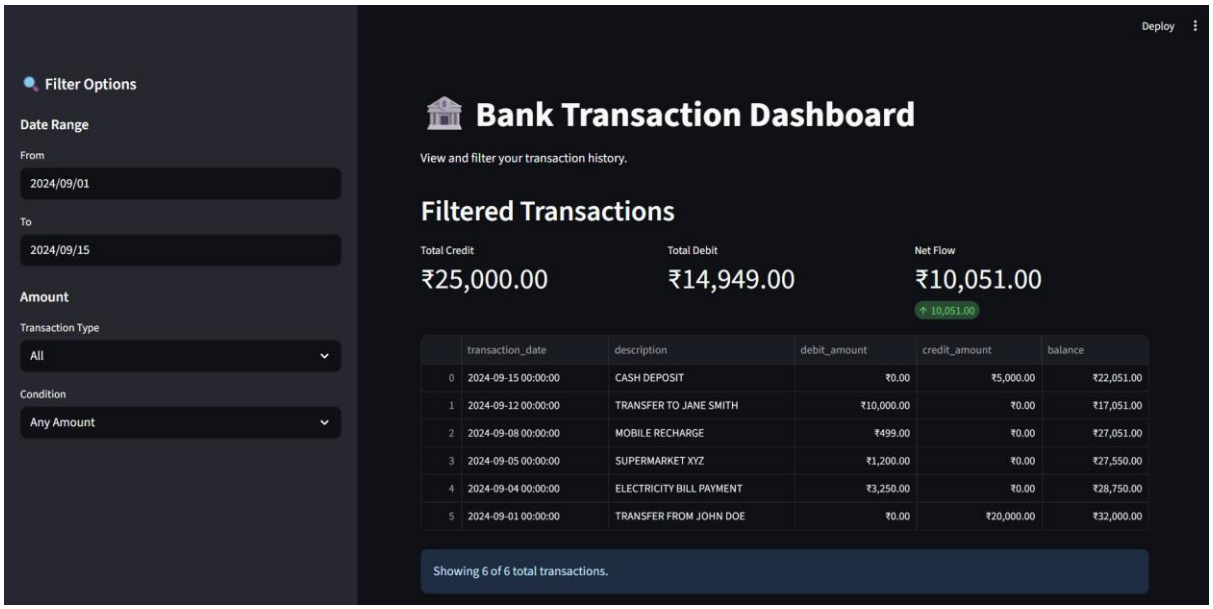
- **Technology Used:** SQLite 3.
- **Schema:** A single table named transactions was created with the following columns to store the essential details of each transaction:
    - id (Primary Key)
    - transaction_date
    - description

- debit_amount
- credit_amount
- balance

2.3. Data Visualization: view_transactions.py

A web application was built to provide an intuitive interface for viewing and analyzing the stored transaction data.

- **Technology Used:** Streamlit.
- **Functionality:**
  - Connects directly to the transactions.db database to fetch and display all records in a clean, tabular format.
  - Features a sidebar with a comprehensive set of filters, allowing users to query the data based on:
    - A specific date range (start date and end date).
    - Credit and Debit amounts (e.g., greater than, less than, or between specified values).
  - The displayed data updates in real-time as the user adjusts the filters.



3. Key Challenge & Resolution: Data Duplication

A significant challenge identified during testing was the creation of duplicate records in the database.

3.1. The Problem

The initial version of the uploader notebook appended all rows from an uploaded file directly into the database. If a user uploaded the same statement file more than once, it would result in the same

set of transactions being stored multiple times. This compromised data integrity and was clearly visible in the Streamlit dashboard, which showed duplicate entries.

3.2. The Solution

A two-pronged approach was implemented in the uploader_notebook.ipynb to resolve this issue:

1. **Prevention of New Duplicates:** The save_to_db function was enhanced significantly. Before attempting to insert a new transaction, the function now executes a SELECT query to check if a record with the exact same transaction_date, description, debit_amount, and credit_amount already exists. Only transactions that are not found in the database are inserted, effectively preventing any new duplicates from being created.
2. **Cleanup of Existing Duplicates:** To fix the data that had already been duplicated, a one-time cleanup utility, clean_duplicates(), was added to the notebook (in Step 5). This function runs a SQL query that identifies all unique transactions, keeps the first instance (MIN(id)), and deletes all other identical entries. Running this utility purged the database of all historical duplicates, ensuring the dataset was clean.

4. Conclusion

The implemented solution successfully meets the project requirements, providing a full pipeline from file upload to data visualization. The use of Jupyter Notebooks for ingestion and Streamlit for reporting creates a flexible and user-friendly system. The resolution of the data duplication issue has made the system more robust and reliable for maintaining an accurate transaction history.