Project Report *on*

"Bank Transaction Management System Using Mainframe and DB2"

Submitted by

Aniket Bansod	22210386
Soham Deshmukh	22211060
Tejaswini Durge	22210270
Arya Kadam	22210766

Under the Guidance of

Dr. Swati B Patil

At

DEPARTMENT OF INFORMATION TECHNOLOGY
BRACT's, Vishwakarma Institute of Information Technology

[April-2025]

Affiliated to

SAVITRIBAI PHULE PUNE UNIVERSITY

CONTENTS

Abstract……4

1. Introduction……..……5

1.1 Overview……..5

1.2 Objective……..8

2. Literature Review ……9

3. Output (Working of the project) ……...…….15

3.1. COBOL……
3.2. JCL……
3.3. GDG……..
3.4. REXX……

4. Conclusion and feature scope…….26

4.1. Conclusion…….26

4.2. Future Scope…….26

5. References……29

DEPARTMENT OF INFORMATION TECHNOLOGY

Certificate

This is to certify that,

Aniket Bansod	22210386
Soham Deshmukh	22211060
Tejaswini Durge	22210270
Arya Kadam	22210766

have successfully completed this project report entitled "**Bank Transaction Management System Using Mainframe and DB2**", under my guidance in partial fulfillment of the requirements for the degree of Bachelor of Engineering in the Department of **INFORMATION TECHNOLOGY** of Vishwakarma Institute of Information Technology, Savitibai Phule Pune University, Pune during the academic year 2024-25.

Date: - 13 April 2025
Place: - Pune

Mrs. Swati Patil
Guide

Dr. P.R. Futane
Head of Department

Abstract

This project, titled Bank Transaction Management System Using Mainframe and DB2, is a mini project designed to simulate core banking operations within an IBM Z/OS mainframe environment using Zowe. The primary objective is to demonstrate practical knowledge of mainframe technologies and showcase how they are applied in real-world sectors such as banking, where reliability, performance, and secure transaction processing are critical.

The system is developed using COBOL for implementing business logic, JCL (Job Control Language) for managing batch processes, GDG (Generation Data Groups) for maintaining historical transaction data, DB2 for managing relational database operations, and REXX for scripting and automation. The database consists of two main tables: an Accounts table that stores customer and balance information, and a Transactions table that records each financial operation such as deposits and withdrawals.

Users can perform transactions that are recorded in sequence, processed one by one, and reflected in the account balances accordingly. The project emphasizes proper transaction handling, sequencing, and data integrity, simulating the critical transaction management processes typical of banking systems.

Overall, this system serves as a proof-of-concept to demonstrate deployment, integration, and execution of key mainframe components, reflecting a real-world scenario of how enterprise-grade systems manage sensitive financial data securely and efficiently.

1. Introduction

1.1 Overview

In today's digital world, mainframe systems continue to play a vital role in managing large-scale, mission-critical applications—particularly in sectors like banking, finance, insurance, and government. These systems are known for their reliability, security, and ability to handle massive volumes of transactions with consistency and speed. The **Bank Transaction Management System Using Mainframes and DB2** is a mini-project designed to showcase the implementation of such systems in a simulated banking environment.

This project aims to replicate a simplified banking transaction process using the IBM Z/OS mainframe platform, employing a range of essential technologies including **COBOL**, **JCL**, **GDG**, **DB2**, and **REXX**. The project is executed in a Zowe-based development environment, which enables interaction with the mainframe using modern interfaces.

At the core of the system are two relational tables managed by DB2:

- **Accounts Table** – Contains account-specific data such as account number, customer name, and balance.
- **Transactions Table** – Stores each transaction with details such as account number, transaction type (credit/debit), amount, and timestamp.

Users can simulate transactions such as deposits and withdrawals. These transactions are processed sequentially and in compliance with banking logic to ensure accuracy and consistency. The transaction management logic ensures each operation is executed in order, and any change in account balance is correctly reflected in the system.

The overall goal of this project is not only to build a working model of a banking transaction system but also to demonstrate a comprehensive understanding of mainframe concepts, data management practices, and real-world application deployment. It bridges academic learning with enterprise-level application scenarios, making it a strong foundation for working on real mainframe systems in the future.

1.2 Objective

The primary objective of this project and report is to design and implement a simplified **Bank Transaction Management System** using IBM Mainframe technologies in a Z/OS environment. The aim is to replicate real-world banking operations and gain

hands-on experience with core components of enterprise mainframe computing.

The specific goals of this project are:

- **To simulate a basic banking transaction system** that can handle deposits and withdrawals using two DB2 tables: *Accounts* and *Transactions*.
- **To develop and execute COBOL programs** that perform transaction logic, update balances, and ensure data consistency.
- **To use JCL (Job Control Language)** for batch job execution, job scheduling, and resource handling.
- **To manage historical data** using GDG (Generation Data Groups) to maintain sequential versions of transaction logs or outputs.
- **To automate and enhance workflows** through the use of REXX scripts.
- **To demonstrate practical deployment and integration** of various mainframe tools within the Zowe interface, aligning the project with modern mainframe development practices.
- **To showcase a real-world application of mainframe systems** in the banking domain, where reliability, transaction integrity, and data security are essential.

This project not only fulfills academic requirements but also provides foundational exposure to the tools, languages, and structures commonly used in enterprise IT systems, especially in banking and finance sectors where mainframes are still heavily relied upon.

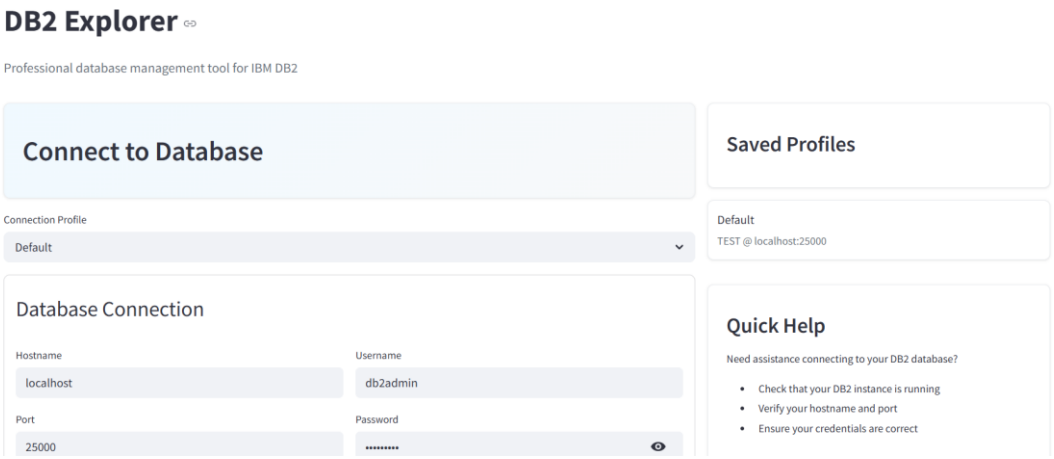
2. Literature Review

1. **IBM Z Xplore:** The **OMP COBOL Programming Course** provided comprehensive insights into the COBOL programming language, focusing on its syntax, data handling capabilities, and structured programming approach. More importantly, the course offered practical guidance on how to **integrate COBOL with DB2**, enabling database operations such as queries, inserts, and updates within

COBOL applications—a critical skill for mainframe-based transaction systems. (Xplore, 2025)

2. **IBM Advanced Track**: Introduction to **REXX scripting** for automation and **Job Control Language (JCL)** for compiling and executing mainframe programs. These tutorials emphasized real-world development workflows, including job submission, output handling, and debugging techniques, all within the Z/OS environment. The hands-on tutorials and simulated exercises provided in the IBM Z Xplore environment served as an essential foundation for the successful implementation of this banking transaction management system. (Xplore, 2025; Amin, Alauddin, & Azad, 2012; Caruso, 2022; Gill, 2025)

3. **Installing DB2 Client:** To deepen practical understanding of relational database management systems on mainframes, the **installation and usage of the IBM DB2 Client** formed an important part of this project's technical preparation. The DB2 client was employed to explore and execute core **DB2 functionalities**, such as creating tables, performing SQL operations, and managing database schemas. (IBM, 2025) Through this hands-on exploration, a strong grasp of how DB2 integrates with COBOL programs was developed.



In addition to using the DB2 client for command-line operations, a **custom application interface** was created to manage and interact with DB2 databases, similar in usability to tools like **PgAdmin for PostgreSQL** and **MongoDB Atlas for MongoDB**. This user-friendly approach enabled easier visualization of data and streamlined testing of SQL queries, offering a practical, GUI-based enhancement to traditional mainframe database management practices. This exercise not only reinforced core concepts of DB2 but also bridged the gap between traditional and modern database interaction methods. (IBM, 2025)

4. **Transaction Processing Systems:** (Rahmatian, 2003) discusses the critical components of transaction processing systems (TPS), emphasizing their role in ensuring data integrity, availability, and security. The study highlights the

importance of ACID properties (Atomicity, Consistency, Isolation, Durability) in maintaining reliable transaction processing in banking systems.

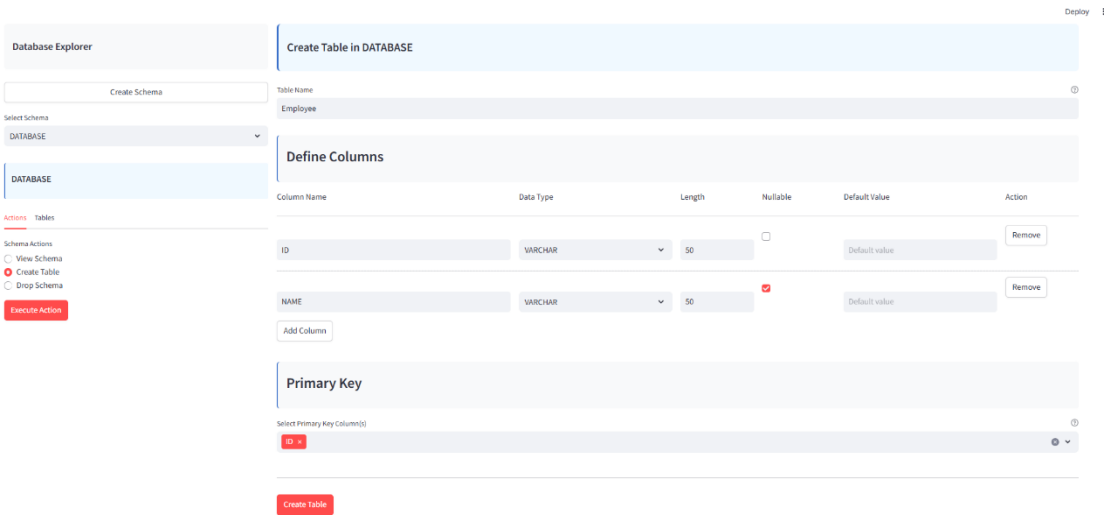
5. **Modernizing Banking Systems**: (Amin, Alauddin, & Azad, 2012) introduces a strategic approach to modernizing aging mainframe systems in the banking sector. Instead of complete overhauls, the paper suggests incremental modernization by integrating secondary databases and offloading data processing tasks, thereby reducing operational risks.

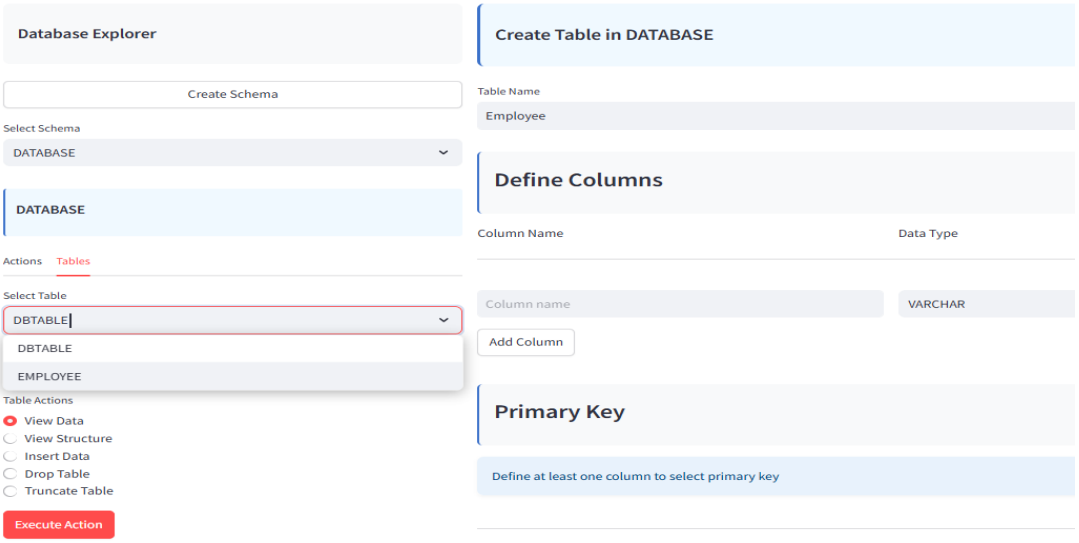
6. **Mainframe Relevance in Modern IT:** (Caruso, 2022), in an analysis by the University of New Hampshire underscores the continued relevance of mainframes in modern IT infrastructures, particularly in sectors requiring high security and reliability, such as banking. The study emphasizes mainframes' strengths in handling large volumes of data and supporting multiple operating systems.

7. **Building Scalable Batch Processing Systems**: (Gill, 2025) explores the development of scalable batch processing systems for financial transactions using mainframes. The research highlights how mainframes can enhance the scalability and robustness of financial systems, ensuring efficient handling of high transaction volumes.

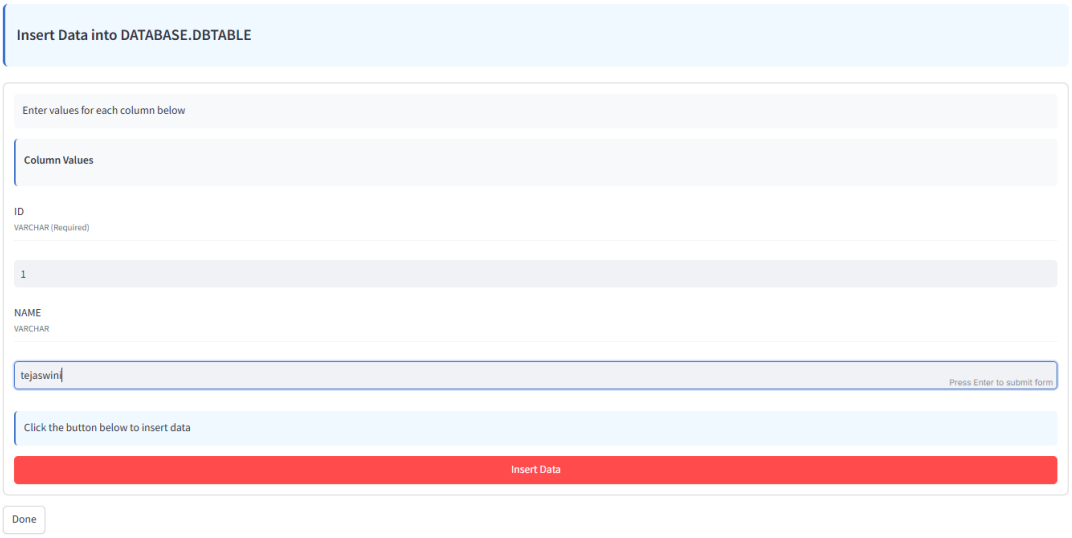
GUI – PROJECT

Create Table:

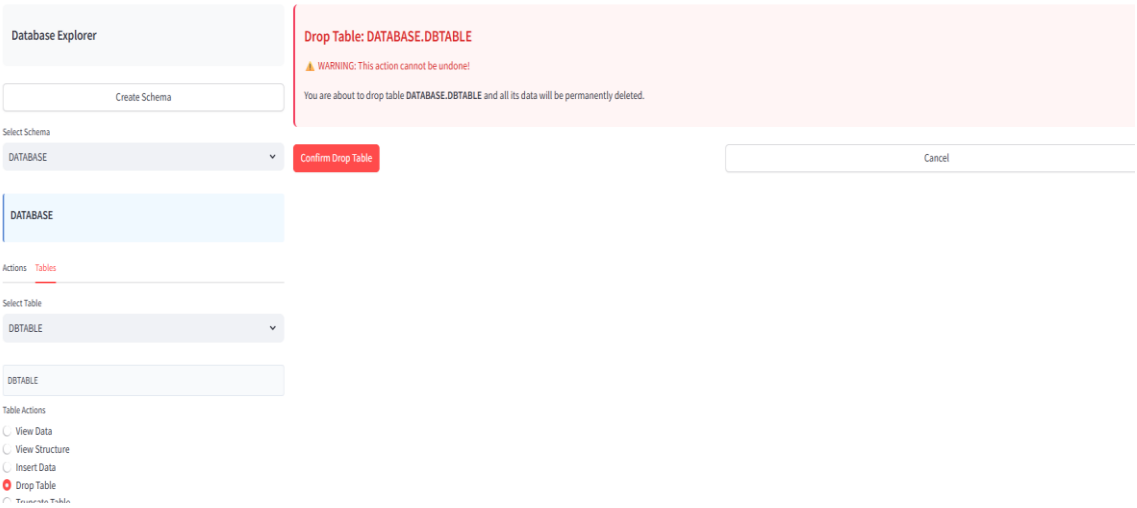



Database Explorer
Create Schema
Select Schema
DATABASE
DATABASE
Actions
Tables
Select Table
DBTABLE
DBTABLE
EMPLOYEE
Table Actions
View Data
View Structure
Insert Data
Drop Table
Truncate Table
Execute Action
Create Table in DATABASE
Table Name
Employee
Define Columns
Column Name
Data Type
Column name
VARCHAR
Add Column
Primary Key
Define at least one column to select primary key

Insert Data:

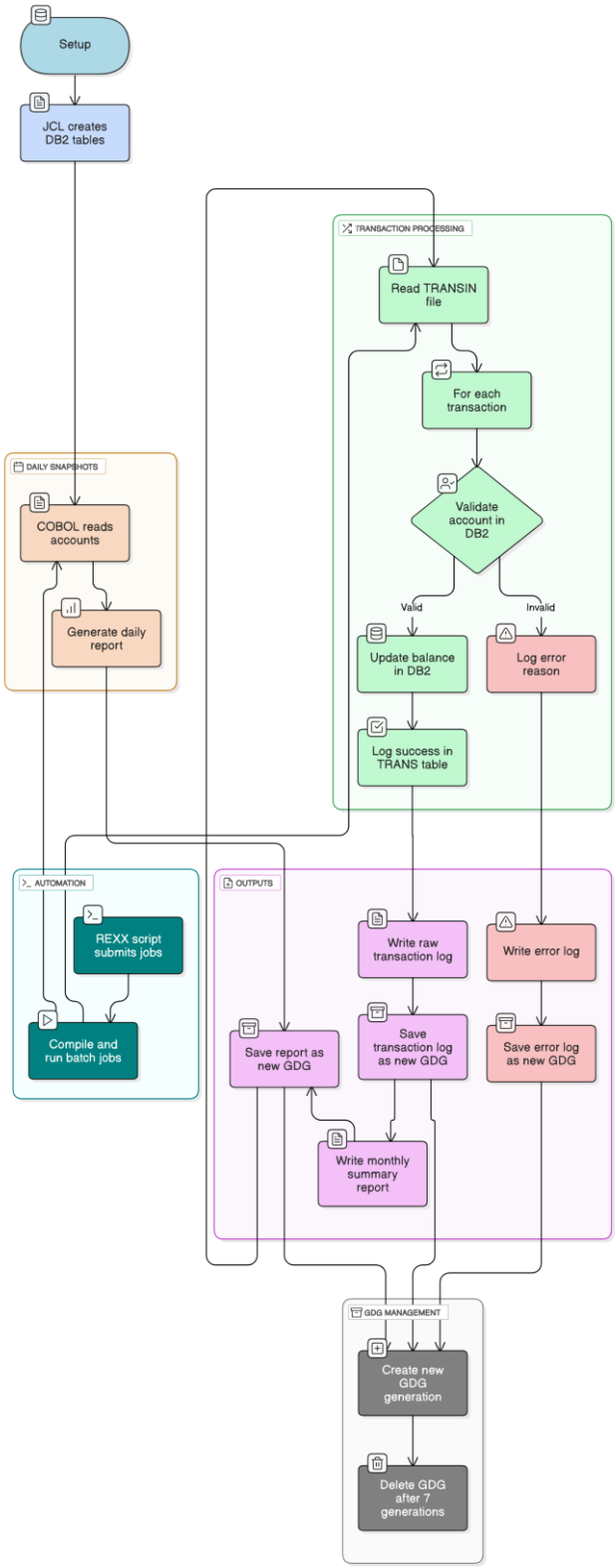

Insert Data into DATABASE.DBTABLE
Enter values for each column below
Column Values
ID
VARCHAR (Required)
1
NAME
VARCHAR
tejaswini
Press Enter to submit form
Click the button below to insert data
Insert Data
Done

Drop Table:



3. Output (Working / Flow of Project)

Overall Flow



3.1 Database Setup using JCL (Job: P2CRETBL)

This phase involves initializing the DB2 database environment for the Bank Transaction Management System using **Job Control Language (JCL)**. A specialized JCL job named **P2CRETBL** was developed to automate the execution of SQL DDL statements within the IBM Z/OS mainframe ecosystem. The purpose of this job is to create the **TRANS** table, which serves as the central storage structure for all transaction records.

The job begins by creating a dedicated **tablespace**, which is the underlying storage container for the TRANS table in DB2. The use of a separate tablespace allows for controlled allocation of physical storage, ensures data independence, and facilitates better performance tuning during query execution.

3.1.1 Tablespace Creation

```
//P2CRETBL   JOB 1
//SQLEXEC   EXEC DB2JCL
//SQL.SYSIN DD *,SYMBOLS=CNVTSYS
--** SQL FOLLOWS ************************************
  SET CURRENT SCHEMA &SYSUID.;

  CREATE TABLESPACE &SYSUID.TS IN &SYSUID.
    USING STOGROUP ZXPUSER PRIQTY 20 SECQTY 20 ERASE NO
    LOCKSIZE PAGE LOCKMAX SYSTEM
    BUFFERPOOL BP0 CLOSE NO COMPRESS YES;
  COMMIT;
  CREATE TABLE &SYSUID.TRANS
               (TRANS_ID   INTEGER      GENERATED ALWAYS AS IDENTITY,
                ACCTNO     CHAR(8)      NOT NULL,
                TRANS_TYPE CHAR(1)      NOT NULL,
                AMOUNT     DECIMAL(9,2) NOT NULL,
                BALANCE    DECIMAL(9,2) NOT NULL,
                REFERENCE  CHAR(20)             ,
                STATUS     CHAR(1)      NOT NULL,
                USERID     CHAR(8)      NOT NULL,
                PRIMARY KEY(TRANS_ID),
                FOREIGN KEY(ACCTNO) REFERENCES &SYSUID.T(ACCTNO))
        IN &SYSUID..&SYSUID.TS;
  COMMIT;
--*************************************************
  CREATE INDEX &SYSUID.TX1
              ON &SYSUID.TRANS (ACCTNO ASC, TRANS_DATE DESC, TRANS_TIME
DES
              USING STOGROUP ZXPUSER PRIQTY 12 ERASE NO
              BUFFERPOOL BP0 CLOSE NO;
--*************************************************
  COMMIT;
```

3.1.2 Table Defination

The TRANS table is created with a carefully designed schema to capture key transactional data. The fields include:

- TRANS_ID – A unique, auto-incrementing identifier for each transaction.
- ACCTNO – The account number associated with the transaction.
- AMOUNT – The monetary value of the transaction.
- STATUS – Indicates the outcome of the transaction (e.g., SUCCESS, FAILED).
- TRANS_DATE and TRANS_TIME – Capture the date and time of the transaction for auditing and chronological tracking.

This schema is optimized for capturing the necessary attributes required to manage and monitor banking transactions in a secure and efficient manner.

3.2 Daily Account Snapshot (COBOL Program: P2CBL)

This component of the system is responsible for generating a daily snapshot of all account records in a human-readable report format. It is implemented using a COBOL program named **P2CBL**, which connects to the DB2 database and extracts data from the **Z52422T** table containing account details.

Input

The input to the COBOL program is the **Z52422T** DB2 table, which holds customer account information. This table typically includes fields such as account number, account limit, current balance, customer name, and additional comments or remarks.

Output

The program generates a structured and formatted report file called **REPORT-FILE**. Each line of the report displays the following details:

- Account Number
- Account Limit
- Account Balance
- Customer Name
- Comments

This output provides a concise view of all active accounts and their financial status.

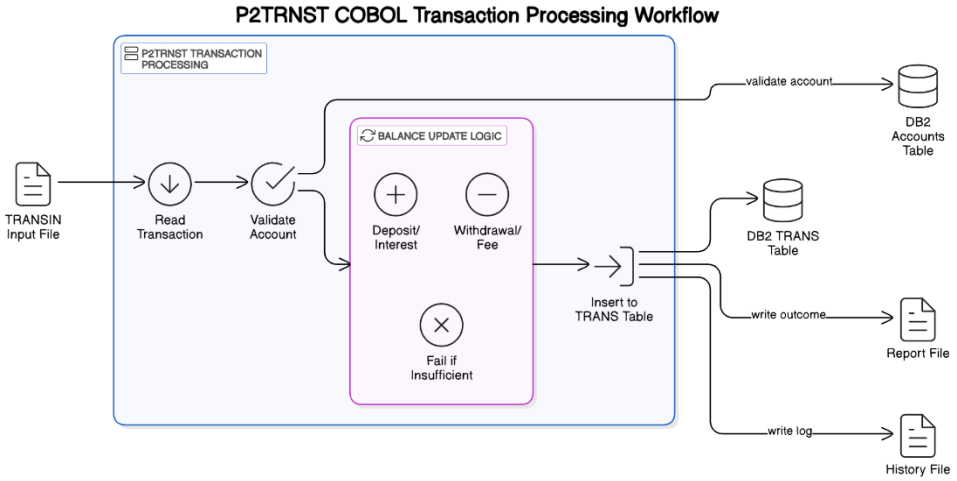
Process Description

- The program first establishes a connection with the DB2 subsystem.
- A **cursor** is declared and opened to fetch all records from the account table sequentially.
- Each record is retrieved one by one using a FETCH loop.
- The retrieved data is formatted into aligned fields suitable for report generation.
- Headers and section titles are written to improve readability, followed by detailed lines containing the account-specific information.

This COBOL program is essential for generating daily operational reports that assist in monitoring customer account status and are useful for both audit and internal review purposes.

3.3 Transaction Processing (COBOL Program: P2TRNST)

This module handles the core functionality of transaction execution within the Bank Transaction Management System. The COBOL program **P2TRNST** is responsible for processing transactions read from an input file, updating the account balance accordingly, logging the result to a report, and maintaining historical records. This simulates real-world banking operations such as deposits, withdrawals, and account adjustments.



Input

The program reads from a transaction input file named **TRANSIN**, which contains multiple transaction records. Each record includes:

- Account Number

- Transaction Type (D – Deposit, W – Withdrawal, I – Interest, F – Fee)
- Transaction Amount
- Transaction Reference Information

Outputs

- **Report File**: Captures the outcome of each transaction (success or failure) with appropriate messages and details.
- **History File**: Stores a persistent log of all processed transactions for future auditing and traceability.

Process Description

1. **Reading Transactions**: The program sequentially reads each transaction record from the TRANSIN file.
2. **Account Validation**: For every transaction, the program validates the existence of the account by checking against the **DB2 accounts table**.
3. **Balance Update Logic**: Based on the transaction type, the program adjusts the balance as follows:
    - **Deposit (D)** or **Interest (I)**: The account balance is **increased** by the specified amount.
    - **Withdrawal (W)** or **Fee (F)**: The account balance is **decreased**, provided sufficient funds are available. If not, the transaction fails.
4. **Logging in TRANS Table**: Each transaction is inserted into the **DB2 TRANS table** with a corresponding **status flag**:
    - C (Completed) – for successful transactions.
    - F (Failed) – for rejected or incomplete transactions.
5. **File Output Generation**: Transaction outcomes are written to the **Report File**, while a complete log is saved to the **History File**, ensuring data integrity and audit capability.

This COBOL routine ensures accurate and controlled transaction processing, similar to how real banking systems execute and monitor financial operations on mainframes.

3.4 Compilation and Execution (JCL)

This section outlines the use of **Job Control Language (JCL)** scripts to compile, bind, and execute the COBOL programs developed for the Bank Transaction Management System. Both compilation and runtime operations are handled through separate JCL jobs to ensure modularity, maintainability, and environment-specific control.

Compilation Jobs (P2CBLC, P2TRNC)

The compilation of COBOL programs—**P2CBL** (Daily Snapshot) and **P2TRNST** (Transaction Processing)—is managed using dedicated JCL scripts:

- These jobs invoke the **DB2 precompiler** to process embedded SQL within the COBOL source code.
- The COBOL compiler is then used to generate object modules.
- A **DB2 BIND** process follows, creating **DB2 Plans** that enable runtime program interaction with DB2 tables.
- Proper inclusion of **SYSLIB**, **DBRMLIB**, and **LOADLIB** ensures all required libraries and control blocks are linked during the process.

This compilation phase ensures all COBOL-DB2 dependencies are resolved and that the resulting executable is registered for secure execution within the DB2 environment.

Runtime Jobs (PRCBLR, P2TRNR)

For execution, separate runtime JCL jobs are defined:

- Programs are executed through the **IKJEFT01** utility under **TSO/E (Time Sharing Option Extensions)**, which facilitates DB2 interaction during runtime.
- Input datasets (e.g., transaction files) are passed, and output datasets are dynamically allocated using **Generation Data Groups (GDGs)** to maintain version-controlled output logs.

This approach leverages mainframe best practices for **versioning, automation, and historical record management**, reinforcing the system's reliability and audit readiness.

3.5 Automation (REXX Script: P2TRAN.REXX, P2SNAP.REXX)

To streamline the process of compiling and executing multiple jobs, a REXX script named **P2TRAN.REXX** is developed and used within the Z/OS environment. This script plays a key role in automating repetitive tasks and reducing manual intervention, especially during batch processing.

Functionality

The REXX script is designed to:

- **Submit JCL Jobs Sequentially**: It programmatically submits the required JCL scripts, including:
  - Compilation jobs for COBOL programs (e.g., P2CBLC, P2TRNC)
  - Runtime jobs (e.g., PRCBLR, P2TRNR)
- **Track Execution Flow**: Ensures jobs are submitted in the correct order so that dependencies—such as successful compilation before execution—are maintained.

- **Job Feedback and Monitoring**: Upon submission, the script returns and displays **JES Job IDs**, allowing users to monitor progress and review logs through SDSF or Zowe CLI.

Benefits

- **Efficiency**: Automates multi-step execution, saving time and ensuring consistency.
- **Error Reduction**: Minimizes the risk of human error in manual job submission.
- **Ease of Use**: Especially useful for new users or during demonstrations of the mini-project, enabling single-command execution of the full transaction processing workflow.

This REXX automation represents a real-world mainframe scripting practice, where batch job orchestration is essential for large-scale and time-sensitive systems such as banking operations.

3.6 OUTPUT (GDG)

1. Daily Account Snapshot

Dataset Name: Z52422.P2ACCT.SNPSOTS
Purpose:
Stores daily snapshots of account balances and related details.

Generated By:
COBOL program P2CBL (via PRCBLR JCL).

Structure:

- Fixed-block format (RECFM=FB)
- Record length: 120 bytes
- Includes headers and detailed rows with:
  - Account number
  - Account limit
  - Balance
  - Customer name
  - Comments

ACCOUNT SNAPSHOT

0ACCOUNT# LIMIT BALANCE LAST NAME FIRST NAME COMMENTS

-------- ---------- ---------- ---------- ---------- ----------

17891797 $10,000.00 $1,129,694.33 WASHINGTON George longed to retire to his fields at Mount Ve

17971801 $10,000.00 $704,691.60 ADAMS John retired to his farm in Quincy

18011809 $10,000.00 $444,687.57 JEFFERSON Thomas retired to Monticello

18091817 $10,000.00 $2,004,697.58 MADISON James retirement at Montpelier

18171825 $10,000.00 $711,203.95 MONROE James Russia must not encroach southward

18251829 $10,000.00 $659,189.91 ADAMS II John Quincy collapsed on the floor of the House from a

18291837 $10,000.00 $704,691.60 JACKSON Andrew that to the victors belong the spoils

18371841 $10,000.00 $704,691.60 VAN BUREN Martin independent treasury system

18411841 $10,000.00 $704,691.60 HARRISON William Henry ornate with classical allusions

18411845 $10,000.00 $704,691.60 TYLER John opposed the Missouri Compromise

18451849 $10,000.00 $3,954,697.45 POLK James health undermined from hard work

18491850 $10,000.00 $704,691.60 TAYLOR Zachary acted at times as though he were above par

18501853	$10,000.00	$704,691.60	FILLMORE	Millard	signed the Fugitive Slave Act
18531857	$10,000.00	$704,691.60	PIERCE	Franklin	he tried to persuade Spain to sell Cuba
18571861	$10,000.00	$704,691.60	BUCHANAN	James	widening rift over slavery

GDG Workflow:

- Each run generates a new GDG version (e.g., P2ACCT.SNPSOTS.G0001V00, G0002V00)
- JCL uses the (+1) syntax to create the next generation automatically

Use Cases:

- Auditing daily account statuses
- Performing trend analysis (e.g., tracking daily balance fluctuations)

2. Monthly Transaction Report

Dataset Name: Z52422.P2MNTHY.REPORTS
Purpose:
Logs the outcomes (success/failure) of all transactions processed within a month.

Generated By:
COBOL program P2TRNST (via P2TRNR JCL)

Structure:

- Record length: 132 bytes
- Includes:
  - Account number
  - Transaction type
  - Amount
  - Date and time
  - Transaction status
  - Reason for failure (if any)

```
TRANSACTION REPORT
0ACCOUNT   TYPE  AMOUNT       DATE       TIME     STATUS   REASON
17891797  D        $5,000.21  2025-05-12  03:00:51  C
18011809  W       $20,000.31  2025-05-12  03:00:51  C
18091817  D      $100,000.46  2025-05-12  03:00:51  C
18171825  I          $500.95  2025-05-12  03:00:51  C
18251829  F        $3,500.13  2025-05-12  03:00:51  C
18451849  D      $250,000.45  2025-05-12  03:00:51  C
18291832  W      $150,000.69  2025-05-12  03:00:51  F  ACCOUNT NOT FOUND
18651869  T       $30,000.39  2025-05-12  03:00:51  F  INVALID TRANSACTION
18771881  D        $1,000.24  2025-05-12  03:00:51  C
18811885  W        $5,000.69  2025-05-12  03:00:51  C
18931897  D       $25,000.24  2025-05-12  03:00:51  C
18971901  F        $1,500.65  2025-05-12  03:00:51  C
19231929  D        $8,000.95  2025-05-12  03:00:51  C
19291933  W      $100,000.54  2025-05-12  03:00:51  F  INSUFFICIENT FUNDS
$0.00  2025-05-12  03:00:51  F  ACCOUNT NOT FOUND
```

GDG Workflow:

- A new generation is created monthly (e.g., P2MNTHY.REPORTS.G0001V00)

Use Cases:

- Monthly operational reporting
- Identifying and analyzing frequent transaction failures
- Compliance and audit validation

3. Transaction History

Dataset Name: Z52422.P2TRNS.HISTORY
Purpose:
Maintains a permanent, detailed log of all processed transactions.

Generated By:
COBOL program P2TRNST (via P2TRNR JCL)

Structure:

- Record length: 100 bytes
- Contains raw transaction data:

- Account number
- Transaction type
- Amount
- Updated balance
- Date and time
- Status
- Reference information
- User ID

GDG Workflow:

- Each run appends a new GDG generation (e.g., P2TRNS.HISTORY.G0001V00)

Use Cases:

- Disaster recovery (to rebuild account state from historical logs)
- Forensic analysis of disputed or failed transactions

4. Conclusion and future scope

4.1 Conclusion

The **Bank Transaction Management System** developed using **IBM Mainframe technologies** successfully demonstrates how core banking operations—such as deposits, withdrawals, and balance updates—can be efficiently implemented and executed in a secure, scalable, and high-performance environment. Leveraging **COBOL for business logic**, **JCL for batch processing**, **DB2 for robust data management**, and **REXX for automation**, the system provides a comprehensive and reliable simulation of real-world transaction processing.

Through this project, we have gained hands-on experience in working with mainframe components like **GDG (Generation Data Groups)** for data versioning and audit trails, and developed a deeper understanding of how large-scale financial systems operate in enterprise environments. The project meets its objectives of ensuring data integrity, automating repetitive tasks, and maintaining clear audit records, which are essential in modern banking systems.

4.2 Future Scope

User Interface Integration:
Extend the system with a front-end interface (possibly web-based) to make the application more interactive and accessible to end-users.

Real-time Transaction Handling:
Incorporate CICS (Customer Information Control System) to handle online transactions in real time instead of batch processing only.

Fraud Detection Module:
Integrate analytics and pattern detection to identify potentially fraudulent or unusual transactions.

Scalability to Multibank Operations:
Modify the system to handle transactions across multiple banks or branches to simulate interbank operations.

Role-based Access Control:
Implement security features to allow access based on user roles such as admin, teller, or auditor.

Automated Backup and Recovery:
Enhance the backup mechanisms for disaster recovery and data loss prevention using automated scripts and cloud integration.

Performance Optimization:
Introduce indexing, advanced DB2 optimization techniques, or explore newer technologies like **z/OS Connect** for hybrid cloud support.

4. References

1. Amin, M. B., Alauddin, M. D., & Azad, M. M. (2012). *Business transaction processing system*. *International Journal of Computer Information Systems*, 4(2), 7–14. Retrieved May 11, 2025, from https://www.internationaljournalssrg.org/IJCSE/2025/Volume12-Issue1/IJCSE-V12I1P103.pdf

2. Caruso, E. (2022). *IBM mainframes: Still a viable option in a world of advancing technology* (Honors thesis). University of New Hampshire. Retrieved May 11, 2025, from https://scholars.unh.edu/cgi/viewcontent.cgi?article=1268&context=honors

3. Gill, R. (2025). *Building scalable batch processing for financial transactions*. *Turkish Journal of Computer and Mathematics Education*, 12(1), 113–121. Retrieved May 11, 2025, from https://turcomat.org/index.php/turkbilmat/article/view/14954

4. IBM. (2025). *IBM DB2 Documentation*. Retrieved May 11, 2025, from https://www.ibm.com/docs/en/db2/

5. IBM Z Xplore. (2025). *COBOL, REXX, and JCL Learning Modules*. Retrieved May 11, 2025, from https://ibmzxplore.influitive.com/channels/47

6. Rahmatian, S. (2003). *Transaction processing systems*. Craig School of Business, California State University, Fresno. Retrieved May 11, 2025, from https://zimmer.csufresno.edu/~sasanr/Teaching-Material/MIS/TPS/TPS.pdf