averiSource

CUSTUPD2.COB

Business Requirements

Concise REPORT

Project: **COBOL Modernization**

Package: AveriSource Analyze

Date: November 12, 2024

Table of Contents

1. Overview

1.1 High Level Summary

1.2 Key Business Objectives

1.3 Process Identifications

1.4 Connections

1.5 Most Used Reference Identifiers

2. Business Requirements

2.1 Business Rule Type

2.2 Business Rule Summary

3. Functional Requirements

3.1 MAIN

3.2 PROCESS-UPDATE

3.3 CREATE-NEW-ACCT-RECORD

3.4 DELETE-ACCT-RECORD

3.5 UPDATE-ACCT-RECORD

3.6 UPDATE-ACCT-PAYMENT-HISTORY

3.7 VALIDATE-RECORD

3.8 VALIDATE-ADDRESS

3.9 GENERATE-NEW-ACCOUNT

3.10 WRITE-ERROR-MESSAGE

3.11 FLATTEN-TABLE

3.12 SHUTDOWN

4. Input Output

4.1 Input/Output

4.2 Summary of Data Operation

4.3 Data Source Usage

5. Code Duplication

6. Source Code Dependencies

7. Disclaimer

1. Overview

Analyze

This Business Requirements Document (BRD) outlines the essential requirements for a new system or solution, capturing the business needs and objectives of the legacy program. It serves as a foundational guide for planning and executing the modernization effort.

1.1 High Level Program Summary

The system allows to read an update file, apply the update records to a VSAM file and a DB2 table, and perform various operations such as creating new customer records, updating existing customer records, deleting existing customer records, and updating payment history. It opens the necessary files (error reporting file, customer file, and customer updates file) and connects to the database. The main processing loop reads the update file and performs the corresponding action (create, update, delete, or payment history update) based on the update operator. It validates the input data, writes the updated records to the VSAM file and the DB2 table, and handles any errors that occur during the process. The system also generates new account numbers, validates addresses, flattens the payment history data for SQL usage, and commits the database changes before shutting down.

For details on the processes and connections in this program, please refer to Table 1.3 and Table 1.4.

1.2 Key Business Objectives

One line objective:
To update customer records in VSAM file and DB2 table based on input update file.

Detailed Objectives:
This COBOL program aims to achieve the following key objectives:

1. **File Operations**
   - Open and handle input/output operations for VSAM file, update file, and error file.
   - Read update records from the input file and apply changes to the VSAM file and DB2 table.

2. **Database Integration**
   - Connect to the DB2 database and perform SQL operations.
   - Insert, update, and delete records in the DB2 table based on the update operations.

3. **Record Processing**
   - Validate input update records for required fields and address format.
   - Create new customer records in both VSAM file and DB2 table.
   - Update existing customer records in VSAM file and DB2 table.
   - Delete existing customer records from VSAM file and DB2 table.
   - Update payment history for existing customer records in VSAM file and DB2 table.

4. **Error Handling**
  - Capture and log errors encountered during file operations, database operations, and record processing.
  - Write error messages to the error file for troubleshooting and auditing purposes.

5. **Utility Functions**
  - Generate new account numbers for new customer records.
  - Flatten the nested payment history structure for SQL operations.
  - Validate address format using an external routine.

6. **Commit and Shutdown**
  - Commit database changes upon successful completion of all updates.
  - Perform proper shutdown procedures, including closing files and terminating the program.

Overall, the program aims to maintain data integrity and consistency between the VSAM file and DB2 table by applying updates from an input file, while providing error handling and logging mechanisms for auditing and troubleshooting purposes.

1.3 Process Identification Details

Analyze ‣ Process Identification

The table below provides high-level information about the program's relationship to the entire system, detailing all the entry points associated with CUSTUPD2.

Entry Point	Translation Name	LOC	# of connections	Complexities	Other Programs	BPD Details		
CUSTUPD2.JCL	Users need to provide this	1527	72	51	ACCTGEN.COB , ADDRVAL.COB , CUSTUPD2.COB , MAKEACT2.COB	BR1	BF1.1	BA1.1.1

1.4 Connections

Analyze ‣ Flowcharts

The table below shows all the connections for **CUSTUPD2.COB**.

File	Type	Child	Child Type	Edge Type	Edge Operation Types
CUSTUPD2.COB	Program	ACCT	Dataset	ProgramToDataset	Delete,Read,ReadAndWrite,Write

CUSTUPD2.COB	Program	ACCTERRS	Dataset	ProgramToDataset	Write
CUSTUPD2.COB	Program	ACCTGEN.COB	Program	ProgramToProgram	Call
CUSTUPD2.COB	Program	ACCTREC	DBTable	ProgramToDBTable	ReadAndWrite
CUSTUPD2.COB	Program	ACCTUPDT	Dataset	ProgramToDataset	Read
CUSTUPD2.COB	Program	ADDRVAL.COB	Program	ProgramToProgram	Call
CUSTUPD2.COB	Program	CONSOLE	Console	ProgramToConsole	Write
CUSTUPD2.COB	Program	CUSTUPD2.COB	SourceFile	ProgramToModule	ContainedBy
CUSTUPD2.COB	SourceFile	ACCTREC.CPY	SourceFile	ModuleToModule	Includes
CUSTUPD2.COB	SourceFile	ACCTRECU.CPY	SourceFile	ModuleToModule	Includes
CUSTUPD2.COB	SourceFile	SQLCA.CPY	SourceFile	ModuleToModule	Includes
STEP02	BatchStep	CUSTUPD2.COB	Program	BatchStepToProgram	Reference

1.5 Most Used Reference Identifiers

Analyze ▸ Variable & Method Impact

The following table shows the top 10 reference identifiers with supporting information.

Name	Ref Type	Count	Translation
ACCTREC	Variable	86	
ERROR-MESSAGE	Variable	17	
WRITE-ERROR-MESSAGE	Method	15	
ACCTDO	Variable	14	
PAY-SUB	Variable	14	
BAL	Variable	12	
ACCTREC-REC	Variable	11	
ACCTFILE	File	11	
ACCT-STATUS	Variable	11	
PAMT	Variable	10	

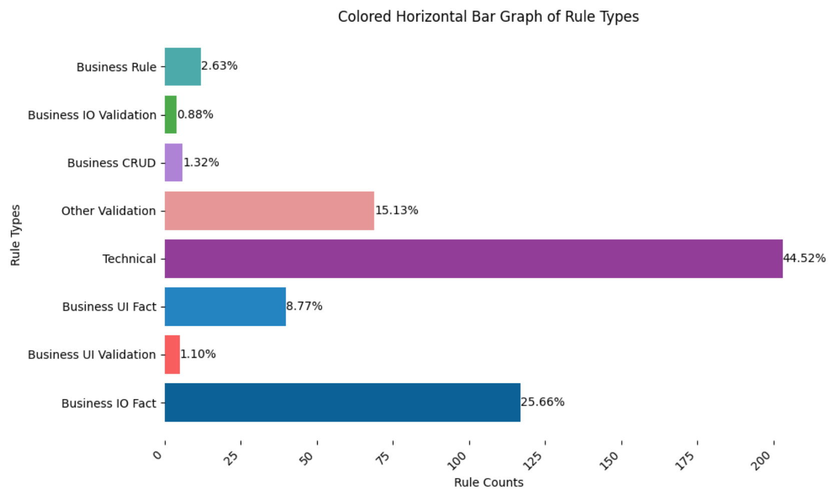
2. Business Rules

2.1 Business Rule Types

The Business Rule Types table provides a list of rule types in the **CUSTUPD2.COB** program, including the total number of rules and a breakdown by rule type.

Business Rule Type	Percentages	Rules
Business IO Fact	25.66	117
Business UI Validation	1.1	5
Business UI Fact	8.77	40
Technical	44.52	203
Other Validation	15.13	69
Business CRUD	1.32	6
Business IO Validation	0.88	4
Business Rule	2.63	12
Total	100	456

Rules Table Plot



Rules Bar Chart

2.2 Business Rule Summary

Data File Handling

This section outlines the rules for managing input, output, and data files used by the program. It covers the opening, reading, writing, and closing of files, as well as error handling related to file operations.

Database Interaction

This section describes the rules for interacting with the database. It covers establishing a connection to the database, executing SQL statements (such as INSERT, UPDATE, DELETE, and SELECT), handling SQL errors, and committing or rolling back transactions.

Record Processing

This section outlines the rules for processing customer account records. It covers creating new records, updating existing records, deleting records, and updating payment history information. It also includes rules for validating record data and generating new account numbers.

Error Handling

This section describes the rules for handling errors that may occur during program execution. It covers writing error messages to an error log file, displaying error messages on the console, and gracefully terminating the program in case of critical errors.

Program Flow Control

This section outlines the rules for controlling the flow of the program. It covers the main processing loop, conditional branching based on update operator values, and the sequence of operations performed for different types of updates (create, update, delete, payment history update).

Data Validation

This section describes the rules for validating input data. It covers checking for required fields, validating address information (potentially through external calls), and any other data validation rules that may be in place.

Utility Functions

This section outlines the rules for any utility functions used by the program, such as generating new account numbers, flattening nested data structures for database interaction, or performing string manipulation operations.

Program Initialization and Termination

This section describes the rules for initializing the program, such as opening files, establishing database connections, and setting up any necessary data structures. It also covers the rules for terminating the program, such as closing files, disconnecting from the database, and releasing any allocated resources.

3. Functional Requirements

The Functional Requirements section details the specific functions a system must perform to meet business needs. It describes the system's behavior, processing logic, and rules. Each section below represents a paragraph in the program.

3.1 MAIN

3.1.1 Summary

This method performs the following tasks:

1. Opens an error reporting file for output and checks for any errors.
2. Opens a customer file for input/output and checks for any errors.
3. Opens a customer updates file for input and checks for any errors.
4. Connects to a database and checks for any errors.
5. Enters a loop to process customer updates from the input file.
6. For each update, it reads the input file and performs the update processing.
7. After all updates are processed, it commits the database changes.
8. Finally, it performs a shutdown procedure.

3.1.2 Steps

1: **Opening Error Reporting File**: The program opens the error reporting file named "ACCTERR" for output. If there is an error opening the file, it displays an error message with the corresponding status code and stops the program execution.

2: **Opening Customer File**: The program opens the customer file named "ACCTFILE" for input-output operations. If there is an error opening the file, it displays an error message with the corresponding status code, writes the error message to the error reporting file, closes the error reporting file, and stops the program execution.

3: **Opening Customer Updates File**: The program opens the customer updates file named "ACCTINPT" for input. If there is an error opening the file, it displays an error message with the corresponding status code, writes the error message to the error reporting file, closes the error reporting file and the customer file, and stops the program execution.

4: **Connecting to the Database**: The program connects to the database named "ACCTDAT" using an SQL statement. If there is an error connecting to the database, it displays an error message with the corresponding error code, writes the error message to the error reporting file, closes all open files, and stops the program execution.

5: **Main Processing Loop**: The program enters a loop that continues until the "PROCESS-FLAG" is not equal to 0. Inside the loop, it reads a record from the customer updates file. If the end of the file is reached, it sets the "PROCESS-FLAG" to 1. If a record is successfully read, it increments the "UPDATE-COUNT" and performs the "PROCESS-UPDATE" procedure.

6: **Committing Database Changes**: After the main processing loop completes, the program commits any pending database changes using an SQL statement.

7: **Shutdown Procedure**: The program performs the "SHUTDOWN" procedure, which likely includes closing any open files and performing any necessary cleanup operations.

3.1.3 Rules

Error Handling

1. **File Open Error Handling**:
   - Description: Checks if the account error file (ACCTERR) opened successfully. If not, it displays an error message and stops the program execution.
   - Condition: ACCTERR-STATUS is not equal to 0 after opening the file.
   - Action: Display an error message with the ACCTERR-STATUS value and stop the program execution.

2. **File Open Error Handling**:
   - Description: Checks if the account file (ACCTFILE) opened successfully for input-output operations. If not, it displays an error message, writes the error to the error file, and stops the program execution.
   - Condition: ACCT-STATUS is not equal to 0 after opening the file.
   - Action: Display an error message with the ACCT-STATUS value, construct an error message string, perform the WRITE-ERROR-MESSAGE routine, close the error file, and stop the program execution.

3. **File Open Error Handling**:
   - Description: Checks if the account updates file (ACCTINPT) opened successfully for input operations. If not, it displays an error message, writes the error to the error file, closes the open files, and stops the program execution.
   - Condition: ACCTINPT-STATUS is not equal to 0 after opening the file.
   - Action: Display an error message with the ACCTINPT-STATUS value, construct an error message string, perform the WRITE-ERROR-MESSAGE routine, close the error file and account file, and stop the program execution.

4. **Database Connection Error Handling**:
   - Description: Checks if the connection to the ACCTDAT database was successful. If not, it displays an error message, writes the error to the error file, closes all open files, and stops the program execution.
   - Condition: SQLCODE is not equal to 0 after executing the database connection statement.
   - Action: Display an error message with the SQLERRMC value, move the SQLERRMC value to the ERROR-MESSAGE variable, perform the WRITE-ERROR-MESSAGE routine, close the error file, account file, and account updates file, and stop the program execution.

Validation Rule

1. **End of File Validation**:
   - Description: Checks if the end of the account updates file (ACCTINPT) has been reached during the main processing loop.
   - Condition: End of file condition is encountered while reading the ACCTINPT file.
   - Action: Set the PROCESS-FLAG to 1 to indicate the end of the processing loop.

Data Processing

1. **Update Count**:

- Description: Increments the UPDATE-COUNT variable for each record read from the account updates file (ACCTINPT).
- Condition: A record is successfully read from the ACCTINPT file.
- Action: Increment the UPDATE-COUNT variable by 1.

2. **Process Update**:
   - Description: Performs the update processing logic for each record read from the account updates file (ACCTINPT).
   - Condition: A record is successfully read from the ACCTINPT file, and the PROCESS-FLAG is 0.
   - Action: Perform the PROCESS-UPDATE routine.

Database Operation

1. **Database Commit**:
   - Description: Commits the database transactions if the program execution reaches this point without any errors.
   - Condition: The program execution reaches this point without encountering any errors.
   - Action: Execute the SQL COMMIT statement to commit the database transactions.

Program Termination

1. **Shutdown Routine**:
   - Description: Performs the shutdown routine after the main processing loop completes.
   - Condition: The main processing loop completes.
   - Action: Perform the SHUTDOWN routine.

3.2 PROCESS-UPDATE

3.2.1 Summary

The method evaluates the value of the UPDATE-OPERATOR variable and performs different actions based on its value. If the value is "C", it creates a new account record. If the value is "D", it deletes an account record. If the value is "U", it updates an account record. If the value is "P", it updates the payment history of an account. If the value is none of the above, it sets an error message and writes it.

3.2.2 Steps

1: **Evaluating the Update Operator**: The method evaluates the value of the UPDATE-OPERATOR variable to determine the appropriate action to perform.

2: **Creating a New Account Record**: If the UPDATE-OPERATOR is "C", the method performs the CREATE-NEW-ACCT-RECORD procedure to create a new account record.

3: **Deleting an Account Record**: If the UPDATE-OPERATOR is "D", the method performs the DELETE-ACCT-RECORD procedure to delete an existing account record.

4: **Updating an Account Record**: If the UPDATE-OPERATOR is "U", the method performs the UPDATE-ACCT-RECORD procedure to update an existing account record.

5: **Updating Account Payment History**: If the UPDATE-OPERATOR is "P", the method performs the UPDATE-ACCT-PAYMENT-HISTORY procedure to update the payment history of an account.

6: **Handling Invalid Update Operator**: If the UPDATE-OPERATOR is not "C", "D", "U", or "P", the method moves an error message to the ERROR-MESSAGE variable and performs the WRITE-ERROR-MESSAGE procedure to handle the invalid update operator.

7: **Exiting the Method**: After executing the appropriate procedure based on the UPDATE-OPERATOR value, the method exits.

3.2.3 Rules

Validation Rules

1. **Check for Valid Update Operator**:
   - Description: This rule checks if the value of the UPDATE-OPERATOR field is valid.
   - Condition: The UPDATE-OPERATOR field must be one of the following values: 'C', 'D', 'U', or 'P'.
   - Action: If the UPDATE-OPERATOR field has any other value, an error message is set, and the WRITE-ERROR-MESSAGE procedure is performed.

Business Rules

1. **Create New Account Record**:
   - Description: This rule creates a new account record.
   - Condition: The UPDATE-OPERATOR field is 'C'.
   - Action: The CREATE-NEW-ACCT-RECORD procedure is performed.

2. **Delete Account Record**:
   - Description: This rule deletes an existing account record.
   - Condition: The UPDATE-OPERATOR field is 'D'.
   - Action: The DELETE-ACCT-RECORD procedure is performed.

3. **Update Account Record**:
   - Description: This rule updates an existing account record.
   - Condition: The UPDATE-OPERATOR field is 'U'.
   - Action: The UPDATE-ACCT-RECORD procedure is performed.

4. **Update Account Payment History**:
   - Description: This rule updates the payment history of an account.
   - Condition: The UPDATE-OPERATOR field is 'P'.
   - Action: The UPDATE-ACCT-PAYMENT-HISTORY procedure is performed.

3.3 CREATE-NEW-ACCT-RECORD

3.3.1 Summary

This method creates a new account record. It generates a new account number, validates the record, writes it to a file, and inserts it into a SQL database table. If any errors occur during the process, it displays an error message and performs a shutdown procedure.

3.3.2 Steps

1: **Creating a New Account Record**: This method is responsible for creating a new account record in the system.

2: **Generating a New Account Number**: The method calls another procedure named "GENERATE-NEW-ACCOUNT" to obtain a new account number for the new record.

3: **Validating the New Record**: The method checks the value of the "VALIDATE-FLAG" variable. If it is zero, it proceeds to the next step.

4: **Moving Data to the Account Record**: The method moves the data from the "ACCTREC-REC" variable to the "ACCTREC" variable and the new account number to the "ACCTDO" variable.

5: **Validating the Record**: The method calls the "VALIDATE-RECORD" procedure to validate the record. If the validation is successful (VALIDATE-FLAG is zero), it proceeds to the next step.

6: **Writing the Record to a File**: The method attempts to write the "ACCTREC" record to a file. If the write operation fails, it moves an error message to the "ERROR-MESSAGE" variable, calls the "WRITE-ERROR-MESSAGE" procedure, and performs a shutdown operation.

7: **Flattening the Table**: If the record is successfully written to the file, the method calls the "FLATTEN-TABLE" procedure to flatten the subscripted portion of the record into individual data items for host variable usage.

8: **Inserting the Record into a SQL Database Table**: The method uses an SQL INSERT statement to insert the new record into the "ACCTREC" table in the database. It populates the values for various columns using host variables.

9: **Handling SQL Errors**: If the SQL INSERT operation fails (SQLCODE is not zero), the method displays an error message with the SQL error message code (SQLERRMC), moves the error message to the "ERROR-MESSAGE" variable, and calls the "WRITE-ERROR-MESSAGE" procedure.

10: **Exiting the Method**: After completing all the steps, the method exits.

3.3.3 Rules

Validation Rule

1. **Validate Account Record**:
   - Description: Validates the account record before writing it to the file and database.

- Condition: After generating a new account number and before writing the record.
- Action: Performs the VALIDATE-RECORD procedure to validate the account record. If the validation is successful (VALIDATE-FLAG = 0), it proceeds with writing the record.

Error Handling

1. **File Write Error Handling**:
   - Description: Handles errors that occur while writing the account record to the file.
   - Condition: After attempting to write the account record to the file.
   - Action: If the ACCT-STATUS is not 0 (indicating an error), it moves an error message to ERROR-MESSAGE, performs the WRITE-ERROR-MESSAGE procedure, and performs the SHUTDOWN procedure.

2. **Database Insert Error Handling**:
   - Description: Handles errors that occur while inserting the account record into the database.
   - Condition: After attempting to insert the account record into the database.
   - Action: If the SQLCODE is not 0 (indicating an error), it displays the error message from SQLERRMC, moves the error message to ERROR-MESSAGE, and performs the WRITE-ERROR-MESSAGE procedure.

Data Manipulation

1. **Flatten Table**:
   - Description: Flattens the subscripted portion of the account record into individual data items for host variable usage.
   - Condition: After successfully writing the account record to the file and before inserting it into the database.
   - Action: Performs the FLATTEN-TABLE procedure to flatten the subscripted portion of the account record.

Database Operation

1. **Insert Account Record**:
   - Description: Inserts the account record into the ACCTREC table in the database.
   - Condition: After successfully writing the account record to the file and flattening the subscripted portion.
   - Action: Executes an SQL INSERT statement to insert the account record into the ACCTREC table, using host variables to pass the data values.

3.4 DELETE-ACCT-RECORD

3.4.1 Summary

The method deletes an existing record from a file and a corresponding row from a database table. It first moves the account number to be deleted into a variable, then attempts to delete the record from the file. If the file deletion fails, it writes an error message. Next, it executes a SQL statement to delete the row from the database table with the same account number. If the database deletion fails, it displays an error message and writes it to an error log.

3.4.2 Steps

1: **Moving Account Details to Output Area**: The account details to be deleted are moved from the input area (ACCTDU) to the output area (ACCTDO).

2: **Deleting Record from File**: The record with the account details stored in ACCTDO is deleted from the ACCTFILE.

3: **Checking File Deletion Status**: If the deletion from the file was unsuccessful (ACCT-STATUS is not 0), an error message is constructed and written using the WRITE-ERROR-MESSAGE routine.

4: **Deleting Row from Database Table**: A SQL DELETE statement is executed to remove the row from the ACCTREC table in the database, where the ACCTDO value matches the value in ACCTDU.

5: **Checking Database Deletion Status**: If the deletion from the database was unsuccessful (SQLCODE is not 0), an error message is displayed, and the SQL error message (SQLERRMC) is moved to the ERROR-MESSAGE area and written using the WRITE-ERROR-MESSAGE routine.

6: **Exiting the Routine**: The routine exits after completing the deletion process.

3.4.3 RULES

Error Handling

1. **Handle Delete Failure from File**:
   - Description: This rule handles the case when the deletion of a record from the account file fails.
   - Condition: If the ACCT-STATUS variable is not equal to 0 after attempting to delete a record from the ACCTFILE.
   - Action: It moves the error message "ERROR! - DELETE RECORD FAILED: " to the ERROR-MESSAGE variable and performs the WRITE-ERROR-MESSAGE routine.

2. **Handle Delete Failure from Database**:
   - Description: This rule handles the case when the deletion of a row from the DB2 table fails.
   - Condition: If the SQLCODE variable is not equal to 0 after attempting to delete a row from the ACCTREC table.
   - Action: It displays the error message "ERROR DELETING FROM DATABASE: " along with the SQLERRMC variable, which contains the error message from the database. It also moves the SQLERRMC value to the ERROR-MESSAGE variable and performs the WRITE-ERROR-MESSAGE routine.

Data Manipulation

1. **Delete Account Record from File**:
   - Description: This rule deletes an existing record from the account file.
   - Condition: Always executed.
   - Action: It moves the value of ACCTDU to ACCTDO and then deletes the corresponding record from the ACCTFILE.

2. **Delete Account Record from Database**:
   - Description: This rule deletes an existing row from the DB2 table.
   - Condition: Always executed.

- Action: It executes an SQL DELETE statement to remove the row from the ACCTREC table where the ACCTDO value matches the ACCTDU value.

3.5 UPDATE-ACCT-RECORD

3.5.1 Summary

This method updates an account record in both a VSAM file and a SQL database table. It first reads the record from the VSAM file using a key to ensure it exists. If the record exists, it performs validation checks. If the validation is successful, it rewrites the record in the VSAM file and then updates the corresponding row in the SQL table with the new data. If any errors occur during the update process, appropriate error messages are generated.

3.5.2 Steps

1: **Reading Account Record**: The method attempts to read an account record from the ACCTFILE file using the key ACCTDU. If the record is not found, an error message is set and written.

2: **Validating Record**: If the account record is found, it is moved to the ACCTREC variable, and the VALIDATE-RECORD procedure is performed. If the validation fails (VALIDATE-FLAG is not 0), the method exits without updating the record.

3: **Updating VSAM File**: If the record is valid, the method attempts to rewrite (update) the ACCTREC record in the ACCT VSAM file. If the rewrite operation fails, an error message is set and written.

4: **Updating SQL Table**: After successfully updating the VSAM file, the method updates the corresponding row in the ACCTREC SQL table. It uses an SQL UPDATE statement with a WHERE clause to update various columns of the row matching the ACCTDO key. If the SQL update fails, an error message is displayed and written.

5: **Error Handling**: Throughout the method, if any error occurs during file operations or SQL updates, appropriate error messages are set and written using the WRITE-ERROR-MESSAGE procedure.

6: **Exiting**: After all operations are completed (successfully or with errors), the method exits.

3.5.3 Rules

Validation Rule

1. **Record Existence Validation**:
   - Description: Checks if the account record exists before attempting an update.
   - Condition: When attempting to update an account record.
   - Action: Reads the account record from the ACCTFILE file using the account key (ACCTDU). If the record does not exist (ACCT-STATUS not equal to 0), an error message is generated and written.

2. **Record Validation**:
   - Description: Validates the account record data before updating.

- Condition: After reading the existing account record and before updating it.
- Action: Performs the VALIDATE-RECORD routine. If the validation fails (VALIDATE-FLAG is not 0), the update process is aborted.

Error Handling

1. **VSAM File Update Error Handling**:
   - Description: Handles errors that occur during the update of the VSAM file.
   - Condition: After attempting to rewrite the account record in the ACCTFILE VSAM file.
   - Action: If the rewrite operation fails (ACCT-STATUS is not 0), an error message is generated and written.

2. **SQL Update Error Handling**:
   - Description: Handles errors that occur during the update of the SQL database table.
   - Condition: After attempting to update the ACCTREC table in the SQL database.
   - Action: If the SQL update operation fails (SQLCODE is not 0), the SQL error message (SQLERRMC) is displayed, moved to the ERROR-MESSAGE variable, and written using the WRITE-ERROR-MESSAGE routine.

Data Update

1. **VSAM File Update**:
   - Description: Updates the account record in the VSAM file.
   - Condition: After validating the record and before updating the SQL database table.
   - Action: Rewrites the ACCTREC record in the ACCTFILE VSAM file.

2. **SQL Database Update**:
   - Description: Updates the account record in the SQL database table.
   - Condition: After successfully updating the VSAM file and before exiting the routine.
   - Action: Updates the ACCTREC table in the SQL database with the new account record data using an SQL UPDATE statement.

3.6 UPDATE-ACCT-PAYMENT-HISTORY

3.6.1 Summary

This method updates an existing account record's payment history. It first reads the existing account record from a file. If the record is found, it checks for an open payment history slot. If all slots are filled, it cycles the payment history by shifting the older entries down. It then adds the new payment information and computes the new account balance. Finally, it updates the account record in both the VSAM file and the corresponding row in a DB2 table.

3.6.2 Steps

1: **Initializing Variables**: The method starts by moving the value of the variable `ACCTDU` to `ACCTDO`, which is likely used to identify the account record to be updated.

2: **Reading Account Record**: The method reads an existing account record from the `ACCTFILE` file.

3: **Checking Account Status**: If the account record is successfully read (indicated by `ACCT-STATUS = 0`), the method proceeds to the next step.

4: **Finding Open Payment History Slot**: The method checks the payment history slots (`ACCTREC(1)`, `ACCTREC(2)`, and `ACCTREC(3)`) for an empty slot by looking for a space in the `BAL` field. If an empty slot is found, the corresponding subscript (`PAY-SUB`) is set to 1, 2, or 3.

5: **Cycling Payment History**: If no empty slot is found (`PAY-SUB = 0`), the method cycles the payment history by shifting the values from `ACCTREC(2)` to `ACCTREC(3)`, `ACCTREC(1)` to `ACCTREC(2)`, and sets `PAY-SUB` to 1.

6: **Adding New Payment Information**: The method copies the payment information from `ACCTREC-REC(1)` to the appropriate payment history slot (`ACCTREC(PAY-SUB)`), including the billing month, day, year, amount, payment month, day, year, and amount.

7: **Computing New Balance**: The method computes the new account balance by subtracting the new payment amount (`PAMT OF ACCTREC-REC(1)`) from the old balance (`BAL OF ACCTREC(1)`), and stores the new balance in the appropriate payment history slot (`BAL OF ACCTREC(PAY-SUB)`).

8: **Updating VSAM File**: The method rewrites the updated account record (`ACCTREC`) to the VSAM file. If the rewrite operation fails, an error message is generated and written.

9: **Flattening Table**: The method performs a `FLATTEN-TABLE` operation, which likely prepares the data for updating the database table.

10: **Updating Database Table**: The method executes an SQL `UPDATE` statement to update the corresponding columns in the `ACCTREC` table with the new payment history information. If the update operation fails, an error message is displayed.

11: **Error Handling**: If the initial account record read fails (`ACCT-STATUS` is not 0), an error message is generated and written, indicating an attempt to update the payment history of a non-existent account.

12: **Exiting Method**: The method exits after completing the update process or encountering an error.

3.6.3 Rules

Data Validation Rules

1. **Check for Existing Account Record**:
   - Description: Verifies if the account record exists in the ACCTFILE before attempting to update the payment history.
   - Condition: After reading the ACCTFILE.
   - Action: If ACCT-STATUS is not equal to 0, it means the account record does not exist, and an error message is displayed.

2. **Check for Open Payment History Slot**:
   - Description: Checks if there is an available slot in the payment history array (ACCTREC) to store the new payment information.

- Condition: After successfully reading the account record from ACCTFILE.
- Action: Iterates through the payment history array (ACCTREC) and assigns the first available slot index to PAY-SUB. If no slot is available, PAY-SUB remains 0.

Data Manipulation Rules

1. **Cycle Payment History**:
   - Description: If no payment history slot is available, it cycles the payment history by shifting the existing entries down and creating an open slot at the beginning.
   - Condition: If PAY-SUB is 0 after checking for an open payment history slot.
   - Action: Shifts the payment history entries down by one position, creating an open slot at the beginning (ACCTREC(1)).

2. **Update Payment History**:
   - Description: Updates the payment history slot with the new payment information.
   - Condition: After finding an available payment history slot or creating one by cycling the payment history.
   - Action: Copies the payment information from ACCTREC-REC(1) to the corresponding fields in the payment history slot (ACCTREC(PAY-SUB)).

3. **Compute New Balance**:
   - Description: Calculates the new account balance based on the previous balance and the new payment amount.
   - Condition: After updating the payment history slot with the new payment information.
   - Action: Subtracts the new payment amount (PAMT OF ACCTREC-REC(1)) from the previous balance (BAL OF ACCTREC(1)) and stores the result in NEW-BALANCE, which is then moved to the corresponding payment history slot (BAL OF ACCTREC(PAY-SUB)).

Data Update Rules

1. **Update VSAM File**:
   - Description: Updates the account record in the VSAM file (ACCTFILE) with the new payment history.
   - Condition: After updating the payment history and computing the new balance.
   - Action: Performs a REWRITE operation on the ACCTREC record in the ACCTFILE. If the REWRITE operation fails (ACCT-STATUS is not 0), an error message is displayed.

2. **Update Database Table**:
   - Description: Updates the corresponding row in the DB2 table (ACCTREC) with the new payment history.
   - Condition: After successfully updating the VSAM file.
   - Action: Executes an SQL UPDATE statement to update the row in the ACCTREC table with the new payment history information. If the UPDATE operation fails (SQLCODE is not 0), an error message is displayed.

Error Handling Rules

1. **Handle VSAM File Update Error**:
   - Description: Handles errors that occur during the REWRITE operation on the VSAM file (ACCTFILE).
   - Condition: If ACCT-STATUS is not 0 after the REWRITE operation.
   - Action: Displays an error message indicating the failure to update the payment history in the VSAM file.

2. **Handle Database Update Error**:
   - Description: Handles errors that occur during the UPDATE operation on the DB2 table (ACCTREC).
   - Condition: If SQLCODE is not 0 after the SQL UPDATE statement.
   - Action: Displays an error message containing the SQL error message code (SQLERRMC).

3.7 VALIDATE-RECORD

3.7.1 Summary

The method performs validation checks on various fields. It first checks if any of the required fields are blank. If any required field is blank, it sets an error message and a validation flag. If all required fields are present, it calls another method to validate the address field.

3.7.2 Steps

1: **Initializing Validation Flag**: The method starts by moving the value 0 to the VALIDATE-FLAG variable, indicating that no validation errors have been encountered yet.

2: **Validating Required Fields**: The method checks if any of the following fields are blank (spaces): ACCTDO, SNAMEDO, FNAMEDO, TELDO, ADDR1DO, ADDR3DO, AUTH1DO, CARDSDO, IMODO, IDAYDO, IYRDO, RSNDO, CCODEDO, APPRDO, SCODE1DO, STATDO, or LIMITDO. If any of these fields are blank, it performs the following steps:

   a. **Setting Error Message**: The method moves the string "REQUIRED FIELD MISSING! " to the ERROR-MESSAGE variable.
   b. **Writing Error Message**: The method calls the WRITE-ERROR-MESSAGE procedure to handle the error message.
   c. **Setting Validation Flag**: The method sets the VALIDATE-FLAG to 1, indicating that a validation error has occurred.

3: **Validating Address**: If none of the required fields are blank, the method calls the VALIDATE-ADDRESS procedure to perform additional validation on the address fields.

4: **Exiting the Method**: After executing the appropriate validation steps, the method exits using the EXIT statement.

3.7.3 Rules

Validation Rule

1. **Required Field Validation**:
   - Description: This rule checks if any of the required fields are blank or empty.
   - Condition: If any of the fields ACCTDO, SNAMEDO, FNAMEDO, TELDO, ADDR1DO, ADDR3DO, AUTH1DO, CARDSDO, IMODO, IDAYDO, IYRDO, RSNDO, CCODEDO, APPRDO, SCODE1DO, STATDO, or LIMITDO are blank or empty spaces.
   - Action: If the condition is true, it sets an error message "REQUIRED FIELD MISSING!" and performs the WRITE-ERROR-MESSAGE procedure. It also sets the VALIDATE-FLAG to 1, indicating a validation failure.

Control Flow

1. **Address Validation**:
   - Description: If all required fields are not blank, it performs additional validation on the address.
   - Condition: If none of the required fields are blank or empty spaces.
   - Action: It calls the VALIDATE-ADDRESS procedure to perform additional address validation.

3.8 VALIDATE-ADDRESS

3.8.1 Summary

The method validates an address by calling an external program or subroutine named "ADDRVAL". It passes four parameters to the called program: VALIDATE-FLAG, ADDR1DO, ADDR2DO, and ADDR3DO. The VALIDATE-FLAG parameter likely indicates whether the address validation was successful or not, while the other three parameters (ADDR1DO, ADDR2DO, and ADDR3DO) represent different parts of the address being validated, such as street address, city, and state/zip code. After calling the external program, the method simply exits.

3.8.2 Steps

1: **Calling External Program**: The method calls an external program named "ADDRVAL" to validate the address.

2: **Passing Parameters**: The method passes four parameters to the external program:
- VALIDATE-FLAG: A flag indicating whether the address is valid or not.
- ADDR1DO: The first line of the address.
- ADDR2DO: The second line of the address.
- ADDR3DO: The third line of the address.

3: **Exiting the Method**: After calling the external program, the method exits, indicating the end of the address validation process.

3.8.3 Rules

Validation Rule

1. **Address Validation**:
   - Description: This rule validates the address provided by the user.
   - Condition: The rule is triggered when the program needs to validate the address.
   - Action: The program calls an external program or subroutine named "ADDRVAL" and passes the following parameters:
     - VALIDATE-FLAG: A flag or indicator to determine if the address is valid or not.
     - ADDR1DO, ADDR2DO, ADDR3DO: These are likely variables containing different parts of the address (e.g., street address, city, state, zip code) to be validated.

The external program "ADDRVAL" performs the actual validation logic and sets the VALIDATE-FLAG accordingly. The calling program can then take appropriate action based on the value of VALIDATE-FLAG.

3.9 GENERATE-NEW-ACCOUNT

3.9.1 Summary

The method generates a new account number by first initializing the new account number variable to spaces and a validation flag to 0. It then retrieves the current highest account number from a file by reading the file in reverse order. If the file read is successful, it calls a subroutine named "ACCTGEN" with the validation flag and the current highest account number as input parameters to generate the new account number. If the file read fails, it sets the validation flag to 1 and closes the file.

3.9.2 Steps

1: **Initializing Variables**: The method starts by initializing the variables `NEW-ACCT-NUM` and `VALIDATE-FLAG` to spaces and 0, respectively.

2: **Setting the Highest Account Number**: The value "99999" is assigned to the variable `ACCTDO`, which is likely used to represent the highest account number.

3: **Positioning the File Pointer**: The `START` statement is used to position the file pointer in the `ACCTFILE` at or before the record with the key value equal to `ACCTDO`.

4: **Reading the Previous Record**: The `READ` statement is used to read the previous record from the `ACCTFILE`. This step is likely performed to retrieve the highest account number currently in use.

5: **Checking File Status**: The method checks the value of `ACCT-STATUS` to determine if the previous read operation was successful. If the status is not 0 (indicating an error), the `VALIDATE-FLAG` is set to 1, and the `ACCTFILE` is closed.

6: **Calling the Account Generation Routine**: If the previous read operation was successful, the method moves the value of `ACCTDO` to `KEY-NUM` and calls a subroutine named "ACCTGEN" with the `VALIDATE-FLAG`, `KEY-NUM`, and `NEW-ACCT-NUM` as arguments. This subroutine is likely responsible for generating a new account number based on the provided inputs.

7: **Exiting the Method**: The method exits after executing the previous steps.

3.9.3 Rules

Validation Rule

1. **Check Account Status**:
   - Description: This rule checks the status of the account file after reading the previous record.
   - Condition: If the account status is not equal to 0 (successful read).
   - Action: Set the VALIDATE-FLAG to 1 and close the account file.

Data Transformation Rule

1. **Generate New Account Number**:
   - Description: This rule generates a new account number based on the current highest account number.
   - Condition: If the account status is 0 (successful read).
   - Action: Move the current highest account number to KEY-NUM and call the "ACCTGEN" program, passing VALIDATE-FLAG, KEY-NUM, and NEW-ACCT-NUM as parameters.

File Operation Rule

1. **Read Previous Account Record**:
   - Description: This rule reads the previous record from the account file.
   - Condition: After starting the account file with a key value less than or equal to "99999".
   - Action: Read the previous record from the account file.

Data Initialization Rule

1. **Initialize Variables**:
   - Description: This rule initializes variables before processing.
   - Condition: At the beginning of the GENERATE-NEW-ACCOUNT section.
   - Action: Move spaces to NEW-ACCT-NUM and move 0 to VALIDATE-FLAG.

3.10 WRITE-ERROR-MESSAGE

3.10.1 SUMMARY

The method writes an error message to a file or display. It first displays the error message, then constructs a record containing the error message, the update operator, the record number, the update count, and the customer account number. This record is written to a file. If there is an error writing to the file, it displays an "ABORT" message and performs a shutdown procedure.

3.10.2 STEPS

1: **Displaying Error Message**: The error message stored in the `ERROR-MESSAGE` variable is displayed on the console or output device.

2: **Clearing Customer Error Record**: The `CUSTERR-RECORD` variable, which is used to store the customer error record, is cleared by moving spaces into it.

3: **Formatting Update Count**: The `UPDATE-COUNT` variable, which holds the count of updates performed, is formatted and stored in the `UPDATE-COUNT-DISPLAY` variable for display purposes.

4: **Building Customer Error Record**: The `CUSTERR-RECORD` variable is populated by concatenating various pieces of information using the `STRING` statement. The components concatenated are:
  - `ERROR-MESSAGE`: The error message
  - `UPDATE-OPERATOR`: The operator who performed the update
  - The literal string ", RECORD NUMBER: "
  - `UPDATE-COUNT-DISPLAY`: The formatted update count

- The literal string ", CUSTOMER-ACCT IS: "
- `ACCTDU`: The customer account number

5: **Writing Customer Error Record**: The `CUSTERR-RECORD` variable, which now contains the complete customer error record, is written to a file or output device.

6: **Checking Write Status**: After writing the customer error record, the `ACCTERR-STATUS` variable is checked. If it is not equal to 0, it indicates an error occurred while writing to the error file.

7: **Error Handling**: If an error occurred while writing to the error file, a message "ABORT -UNABLE TO WRITE TO ERROR FILE!" is displayed on the console or output device, and the `SHUTDOWN` procedure is performed, which likely handles the error or terminates the program.

8: **Exiting Procedure**: The `WRITE-ERROR-MESSAGE` procedure is exited, and control is returned to the calling routine.

3.10.3 Rules

Error Handling

1. **Write Error Message**:
   - Description: This rule handles the writing of error messages to a log file or display.
   - Condition: An error condition is encountered during program execution.
   - Action:
     1. Display the error message.
     2. Clear the error record buffer.
     3. Format the error message with relevant details like the update count, operator, record number, customer account, and error message.
     4. Write the formatted error message to the error log file.
     5. Check if the write operation was successful. If not, display an "ABORT" message and perform a shutdown procedure.

Validation Rule

1. **Check Write Status**:
   - Description: This rule validates the status of the write operation to the error log file.
   - Condition: After attempting to write the error message to the log file.
   - Action: If the write status is not equal to 0 (indicating an error), display an "ABORT" message and perform a shutdown procedure.

3.11 FLATTEN-TABLE

3.11.1 Summary

The method FLATTEN-TABLE moves various account details from a data structure ACCTREC to individual variables. It copies the balance (BAL), billing month (BMO), billing day (BDAY), billing year (BYR), billing amount (BAMT), payment month (PMO), payment day (PDAY), payment year (PYR), and payment amount (PAMT) from the first three records in ACCTREC to corresponding variables with suffixes -1, -2, and -3 respectively.

3.11.2 Steps

1: **Moving Account Balance**: The balance (BAL) of the first account record (ACCTREC(1)) is moved to the variable BAL-1.

2: **Moving Beginning Month**: The beginning month (BMO) of the first account record (ACCTREC(1)) is moved to the variable BMO-1.

3: **Moving Beginning Day**: The beginning day (BDAY) of the first account record (ACCTREC(1)) is moved to the variable BDAY-1.

4: **Moving Beginning Year**: The beginning year (BYR) of the first account record (ACCTREC(1)) is moved to the variable BYR-1.

5: **Moving Beginning Amount**: The beginning amount (BAMT) of the first account record (ACCTREC(1)) is moved to the variable BAMT-1.

6: **Moving Posting Month**: The posting month (PMO) of the first account record (ACCTREC(1)) is moved to the variable PMO-1.

7: **Moving Posting Day**: The posting day (PDAY) of the first account record (ACCTREC(1)) is moved to the variable PDAY-1.

8: **Moving Posting Year**: The posting year (PYR) of the first account record (ACCTREC(1)) is moved to the variable PYR-1.

9: **Moving Posting Amount**: The posting amount (PAMT) of the first account record (ACCTREC(1)) is moved to the variable PAMT-1.

10: **Moving Account Balance**: The balance (BAL) of the second account record (ACCTREC(2)) is moved to the variable BAL-2.

11: **Moving Beginning Month**: The beginning month (BMO) of the second account record (ACCTREC(2)) is moved to the variable BMO-2.

12: **Moving Beginning Day**: The beginning day (BDAY) of the second account record (ACCTREC(2)) is moved to the variable BDAY-2.

13: **Moving Beginning Year**: The beginning year (BYR) of the second account record (ACCTREC(2)) is moved to the variable BYR-2.

14: **Moving Beginning Amount**: The beginning amount (BAMT) of the second account record (ACCTREC(2)) is moved to the variable BAMT-2.

15: **Moving Posting Month**: The posting month (PMO) of the second account record (ACCTREC(2)) is moved to the variable PMO-2.

16: **Moving Posting Day**: The posting day (PDAY) of the second account record (ACCTREC(2)) is moved to the variable PDAY-2.

17: **Moving Posting Year**: The posting year (PYR) of the second account record (ACCTREC(2)) is moved to the variable PYR-2.

18: **Moving Posting Amount**: The posting amount (PAMT) of the second account record (ACCTREC(2)) is moved to the variable PAMT-2.

19: **Moving Account Balance**: The balance (BAL) of the third account record (ACCTREC(3)) is moved to the variable BAL-3.

20: **Moving Beginning Month**: The beginning month (BMO) of the third account record (ACCTREC(3)) is moved to the variable BMO-3.

21: **Moving Beginning Day**: The beginning day (BDAY) of the third account record (ACCTREC(3)) is moved to the variable BDAY-3.

22: **Moving Beginning Year**: The beginning year (BYR) of the third account record (ACCTREC(3)) is moved to the variable BYR-3.

23: **Moving Beginning Amount**: The beginning amount (BAMT) of the third account record (ACCTREC(3)) is moved to the variable BAMT-3.

24: **Moving Posting Month**: The posting month (PMO) of the third account record (ACCTREC(3)) is moved to the variable PMO-3.

25: **Moving Posting Day**: The posting day (PDAY) of the third account record (ACCTREC(3)) is moved to the variable PDAY-3.

26: **Moving Posting Year**: The posting year (PYR) of the third account record (ACCTREC(3)) is moved to the variable PYR-3.

27: **Moving Posting Amount**: The posting amount (PAMT) of the third account record (ACCTREC(3)) is moved to the variable PAMT-3.

28: **Exiting the Method**: The method FLATTEN-TABLE exits after completing the data movement operations.

3.11.3 Rules

Data Transformation Rule

1. **Flatten Table Rule**:
   - Description: This rule flattens a table or array structure by moving the values from the source table/array to individual target fields.
   - Condition: When the data needs to be transformed from a table/array structure to individual fields.
   - Action: The rule moves the values from the source table/array (ACCTREC) to individual target fields (BAL-1, BMO-1, BDAY-1, BYR-1, BAMT-1, PMO-1, PDAY-1, PYR-1, PAMT-1, BAL-2, BMO-2, BDAY-2, BYR-2, BAMT-2, PMO-2, PDAY-2, PYR-2, PAMT-2, BAL-3, BMO-3, BDAY-3, BYR-3, BAMT-3, PMO-3, PDAY-3, PYR-3, PAMT-3) using the MOVE statement.

3.12 SHUTDOWN

3.12.1 Summary

The method performs the following tasks:

The method named "SHUTDOWN" is responsible for closing three files: ACCTFILE, ACCTINPT, and ACCTERR. After closing these files, it terminates the program execution by issuing the "STOP RUN" statement.

3.12.2 Steps

1: **Closing Files**: The method is closing three files: ACCTFILE, ACCTINPT, and ACCTERR. These files were likely opened earlier in the program for reading or writing data related to accounts.

2: **Terminating Program**: After closing the files, the method issues the STOP RUN statement, which terminates the execution of the COBOL program.

3.12.3 Rules

Error Handling

1. **File Closure**:
   - Description: This rule ensures that all open files are properly closed before the program terminates.
   - Condition: When the program is about to terminate.
   - Action: Close the ACCTFILE, ACCTINPT, and ACCTERR files.

Program Termination

1. **Stop Program Execution**:
   - Description: This rule terminates the program execution.
   - Condition: After all necessary operations have been completed.
   - Action: Stop the program from running further.

4. Inputs and Outputs

Analyze ‣ Sequence Diagram

4.1 Inputs

1. ACCTFILE: The VSAM file containing customer account records. It is opened for input-output and accessed using various file operations like READ, WRITE, REWRITE, and DELETE.

2. ACCTINPT: The sequential file containing customer account update records. It is opened for input and read to process updates.

3. ACCTERR: The sequential file used for error logging. It is opened for output, and error messages are written to this file.

4. ACCTREC: The copybook containing the record layout for the ACCTFILE VSAM file. It is included in the DATA DIVISION and used to define the structure of customer account records.

5. ACCTREC-UPDT: The copybook containing the record layout for the ACCTINPT update file. It is included in the DATA DIVISION and used to define the structure of customer account update records.

6. SQLCA: The SQL Communication Area copybook, which is included in the DATA DIVISION and used for handling SQL-related errors and warnings.

7. User input: The program does not explicitly prompt for user input. However, it processes customer account update records from the ACCTINPT file, which can be considered user input data.

8. Function parameters: The program calls two external functions, "ADDRVAL" and "ACCTGEN," passing parameters to validate addresses and generate new account numbers, respectively.

9. Database query: The program connects to the ACCTDAT database and performs various SQL operations, including INSERT, UPDATE, DELETE, and COMMIT, to interact with the ACCTREC table.

The program reads customer account update records from the ACCTINPT file, processes them based on the update operator (create, update, delete, or payment history update), and applies the changes to both the ACCTFILE VSAM file and the ACCTREC table in the ACCTDAT database. It also performs validations, error handling, and logging throughout the execution.

4.2 Outputs

Based on the provided COBOL source code, here is a comprehensive list of all outputs produced during execution, along with a brief description of when and why each output is generated:

1. **Error Message Output**: A message indicating an error opening or accessing files, connecting to the database, or performing operations. These messages are displayed and written to the error file (ACCTERR) when an error condition is encountered.

2. **Database Connection Error**: A message indicating an error connecting to the database. This output is generated when the SQL CONNECT statement fails, and the error message is displayed and written to the error file.

3. **Database Operation Error**: A message indicating an error during database operations, such as inserting, updating, or deleting records. These messages are displayed and written to the error file when an SQL statement fails.

4. **Record Validation Error**: A message indicating that a record failed validation due to missing required fields or invalid data. These messages are displayed and written to the error file during the record validation process.

5. **File I/O Error**: A message indicating an error opening, reading, writing, or closing files. These messages are displayed and written to the error file when a file operation fails.

6. **Update Count Display**: The number of records processed during the update operation. This output is displayed and written to the error file for each record processed, along with other relevant information.

7. **Database Commit Message**: A message indicating that the database changes have been committed successfully. This output is not explicitly shown in the provided code but may be added for debugging or logging purposes.

8. **Program Termination Message**: A message indicating that the program has terminated normally or abnormally. This output is not explicitly shown in the provided code but may be added for debugging or logging purposes.

9. **Debug or Trace Messages**: Additional messages or outputs may be added throughout the code for debugging or tracing purposes, such as displaying variable values or program flow information.

It's important to note that the provided code does not include any explicit DISPLAY statements for outputting data to the console or terminal. However, the code does include logic for writing error messages and other relevant information to the error file (ACCTERR) during various stages of execution.

4.3 Summary of Data Operations

Discover ‣ Data Operations

The table below shows information on CRUD statements categorized by the **CUSTUPD2.COB** program.

Total Operations	Create	Read	Update	Delete	Other	Data Sources
24	3	4	4	2	11	4

4.4 Data Source Usage

Discover ‣ Data Source Usage

The following table lists the data sources used by the **CUSTUPD2.COB** program.

Data Source	Program	Total Data Operations
ACCTREC	**CUSTUPD2.COB**	4
ACCTINPT	**CUSTUPD2.COB**	4
ACCTERR	**CUSTUPD2.COB**	6
ACCTFILE	**CUSTUPD2.COB**	13

5.0 Code Duplication

Analyze ‣ Code Block & File Comparison

The table below shows the **top 5 programs and blocks** with **90% or more similar code** with **CUSTUPD2.COB**. For additional code duplication information, please see the AveriSource Platform.

Overview			Code Block 1			Code Block 2		
#	**Similarity**	**Differences**	**Code Block**	**File**	**Block Line**	**Code Block**	**File**	**Block Line**
1	100	0	VALIDATE-RECORD	CUSTUPD2.COB	539	VALIDATE-RECORD	CUSTUPD3.cbl	603
2	100	0	VALIDATE-RECORD	CUSTUPD2.COB	539	VALIDATE-RECORD	CUSTUPDT.COB	219
3	100	0	GENERATE-NEW-ACCOUNT	CUSTUPD2.COB	576	GENERATE-NEW-ACCOUNT	CUSTUPDT.COB	256
4	100	0	WRITE-ERROR-MESSAGE	CUSTUPD2.COB	593	WRITE-ERROR-MESSAGE	CUSTUPD3.cbl	656
5	100	0	FLATTEN-TABLE	CUSTUPD2.COB	612	FLATTEN-TABLE	MAKEACT2.COB	553

6.0 Source Code Dependencies

Discover ‣ Dependencies

The following technical dependencies and assumptions for the **CUSTUPD2.COB** program:

Technical Dependencies
- **VSAM File**: The program reads from and writes to a VSAM file named ACCTFILE, which stores customer account records.
- **Sequential Input File**: The program reads update records from a sequential input file named ACCTINPT.
- **Sequential Error File**: The program writes error messages to a sequential file named ACCTERR.
- **DB2 Database**: The program connects to a DB2 database named ACCTDAT and performs insert, update, and delete operations on a table named ACCTREC.
- **Copybooks**: The program uses copybooks ACCTREC.CPY, ACCTRECU.CPY, and SQLCA.CPY to define record layouts and SQL communication area.
- **External Programs/Subroutines**: The program calls external programs/subroutines named ADDRVAL and ACCTGEN for address validation and account number generation, respectively.

Assumptions
- **Database Connection**: The program assumes that the DB2 database connection details are externalized or hardcoded.
- **External Program Availability**: The program assumes that the external programs ADDRVAL and ACCTGEN are available and accessible.
- **File Locations**: The program assumes that the file locations for ACCTFILE, ACCTINPT, and ACCTERR are defined and accessible.
- **Copybook Availability**: The program assumes that the copybooks ACCTREC.CPY, ACCTRECU.CPY, and SQLCA.CPY are available and accessible.
- **Record Layouts**: The program assumes that the record layouts defined in the copybooks match the actual data structures in the VSAM file and DB2 table.
- **Update Operator Values**: The program assumes that the update operator values ("C", "U", "D", "P") are valid and correctly interpreted.
- **Error Handling**: The program assumes that error handling is sufficient and appropriate actions are taken in case of errors.

7.0 Disclaimer

Code Completeness

It is assumed that the analyzed codebase is complete and includes all necessary components to understand the full scope of the business logic.

Functional Integrity

It is assumed that the analyzed code accurately reflects the intended business logic and functionality without undocumented shortcuts or workarounds.

Consistency of Business Rules

It is assumed that the business rules embedded in the codebase are consistent with current business processes and do not require significant modification.

No Hidden Business Logic

It is assumed that there is no hidden business logic or critical functionality embedded in comments or inactive sections of the COBOL code.