Modernization assessment - CARDDEMO

Prepared 19 November 2026 · Selected: job POSTTRAN · 23 reviewed summaries

A sample assessment of AWS CardDemo, an open-source mainframe credit card application (Apache License 2.0, Copyright Amazon.com, Inc. or its affiliates), at commit 59cc6c2 of https://github.com/aws-samples/aws-mainframe-modernization-carddemo. No client code is included.

1. Executive summary
2. Inventory
3. Jobs and transactions
4. Modernization roadmap
5. Job and transaction details
6. Data access
7. Data dictionary
8. Findings
9. Appendix: sources and method

Executive summary

1	0	1	697
batch jobs	transactions / CICS flows	programs in scope	lines of source

6	0
datasets	findings to resolve

This assessment covers 1 batch job(s) and 0 online transaction(s) or CICS flow(s), 1 application program(s), and 697 lines of source. Approach: 1 refactor (suggested, to confirm).

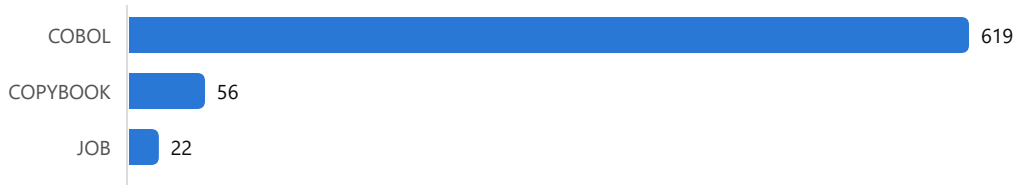
What they do

- **POSTTRAN** - POSTTRAN is the daily posting run. It takes the day's card transactions, rejects the ones that fail basic checks (unknown card, missing account, over the credit limit, account expired), and posts the rest: each is added to the transaction master file, and its amount is added to the account's balance and to the account's running balance for that transaction category. Rejected transactions go to a new rejects file with the reason, for someone to follow up.

Inventory

What the selected jobs and transactions reach: 1 COBOL program(s) and 0 assembler module(s), 5 copybook(s), 1 job(s), 0 PROC(s), 0 CICS transaction(s) or flow(s). Everything loaded holds 44 COBOL program(s), 62 copybook(s), 46 job(s), and 25 CICS transaction(s).

Lines of source by type (excluding comments and blanks)



Type	Code	Comment
COBOL	619	59
COPYBOOK	56	30
JOB	22	23

COBOL programs by kind



No Db2, IMS, or MQ use.

Largest programs (lines of code)



Most complex programs

Decision points: IF, WHEN, and UNTIL conditions - roughly the number of paths to test. Depth: the longest chain of PERFORMs from the first paragraph.

Program	Decision points	IF	EVALUATE	Loops	PERFORM depth	Paragraphs	GO TO
CBTRN02C	49	48	0	1	4	26	0

Jobs and transactions

What each job or transaction reaches - its programs, copybooks, screens, and data - and how much source that is. Shared programs and copybooks count in every scope that uses them.

Lines of source in scope



Name	Kind	programs	copybooks	screens (mapsets)	datasets	Db2 tables	IMS segments	Lines	Approach
POSTTRAN	job	1	5		6			697	Refactor suggested

Modernization roadmap

Approach	Meaning	Count
Rehost	Move as is to a mainframe-compatible runtime first; change the code later.	0
Refactor	Rewrite the logic on the target platform (services, SQL, a reporting or ETL tool).	1
Expose as API	Keep the business logic, put it behind a service interface, and replace the 3270 screen.	0
Retire	Not needed on the target platform, or no longer used.	0

1 of 1 approach(es) are suggestions from the facts below, to confirm with you before planning.

Refactor (1)

Name	Why
POSTTRAN suggested	1 program(s), 697 lines: small enough to rewrite as a batch service, query, or report on the target platform.

Job and transaction details

Job POSTTRAN

Refactor suggested - 1 program(s), 697 lines: small enough to rewrite as a batch service, query, or report on the target platform.

1	1	1	1
jobs	steps	programs	system programs

5	6	12	697
copybooks	datasets	record layouts	lines of source



What it does

POSTTRAN is the daily posting run. It takes the day's card transactions, rejects the ones that fail basic checks (unknown card, missing account, over the credit limit, account expired), and posts the rest: each is added to the transaction master file, and its amount is added to the account's balance and to the account's running balance for that transaction category. Rejected transactions go to a new rejects file with the reason, for someone to follow up.

Step by step

1. **Post the daily transactions (STEP15, CBTRN02C).** Reads the daily transaction file in sequence. For each transaction it looks up the card in the cross-reference to find the account, reads the account, and checks the credit limit and expiration date. Valid transactions update the transaction category balance (creating it for a first transaction in a category), the account's current balance and current-cycle credit or debit total, and are written to the transaction master with the posting time. Invalid ones are written to the rejects file with a reason code (100 unknown card, 101 account not found, 102 over limit, 103 after account expiration). The step ends with return code 4 if anything was rejected.

What it needs

- The day's transactions in `AWS.M2.CARDDEMO.DALYTRAN.PS`, delivered by whatever captures card activity upstream (not in the source provided).
- The card cross-reference (`CARDXREF.VSAM.KSDS`), the account master (`ACCTDATA.VSAM.KSDS`), and the category balances (`TCATBALF.VSAM.KSDS`).
- The transaction master (`TRANSACT.VSAM.KSDS`) in a state the program can open for output - it's opened OUTPUT although the JCL allocates the existing file (DISP=SHR), which for a VSAM file works only if it's empty or defined as reusable.

What it produces

- New records on the transaction master - read later by the transaction report (TRANREPT), the backup and combine jobs, and the statements (CREASTMT).
- Updated account balances and cycle totals - used by the interest run (INTCALC), the statements, and the online account screens.
- Updated category balances - the basis of the interest calculation (INTCALC).
- `AWS.M2.CARDDEMO.DALYREJS(+1)`, a new generation of rejected transactions with reasons, and return code 4 when there are any.

Notes for modernization

- **Posting rules to confirm with the business:** the credit-limit check uses the current cycle's credits minus debits plus the new amount, not the account balance; and when a transaction is both over limit and past expiration, only the expiration reason is recorded.
- **Partial updates on failure:** an account update that fails is silently ignored after the category balance has been updated and while the transaction is still written; a failure part-way through the run leaves some transactions posted and others not, with no restart point. A modern version would post each transaction atomically (one database transaction) or make the run restartable.
- **Opening the master for output** depends on how the VSAM file is defined; confirm the actual run procedure (the scheduler definitions in the source aren't analyzed yet).
- A natural first candidate for a service: "validate and post a card transaction" is a clear unit of business logic that online and batch could share.

Data access

Which programs create (C), read (R), update (U), or delete (D) each file, Db2 table, and IMS segment. Batch files are matched to their datasets through the JCL, CICS files through the CSD, so online and batch use of the same file share a row.

Data	CBTRN02C
AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS file	RU
AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS file	R
AWS.M2.CARDDEMO.DALYREJS file	C
AWS.M2.CARDDEMO.DALYTRAN.PS file	R
AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS file	CRU
AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS file	C

Data dictionary

The records kept in files and Db2 tables: what each field means (from reviewed summaries, where there are any) and, from the source, which programs set each field. A program sets a field when it moves, computes, or accepts a value into it; reading or writing the whole record is under "stored in".

CVACT01Y: ACCOUNT-RECORD

- Stored in ACCTFILE: looked up by CBTRN01C
- Stored in AWS.M2.CARDDEMO.ACCTDATA.IMPORT: written by CBIMPORT
- Stored in AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS: looked up by COACCT01, COACTVWC, COPAUA0C, COPAUS0C; read by CBACT01C, CBEXPORT; updated by CBACT04C, CBTRN02C, COACTUPC, COBIL00C

One record per credit card account: its balance, limits, key dates, and current billing-cycle totals. Kept in the account master file (AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS, keyed by account ID), 300 bytes long.

Fields

Field	Meaning	Notes
ACCOUNT-RECORD	The whole account record	Read by the account view, update, and bill payment screens and by the batch jobs; rewritten by posting (POSTTRAN), interest (INTCALC), bill payment (CB00), and the account update screen (CAUP)
ACCT-ID	Account number, 11 digits	The file's key. Other records point to it: the card record (CARD-ACCT-ID) and the card cross-reference (XREF-ACCT-ID)
ACCT-ACTIVE-STATUS	Whether the account is active, one character	Probably Y or N; the copybook defines no values. Set by the import job; the account update screen changes it through its own copy of the record (ACCT-UPDATE-RECORD)
ACCT-CURR-BAL	Current balance, signed, two decimals (zoned decimal, not packed)	Probably the amount the customer owes. Maintained by three processes: posting adds each posted transaction (POSTTRAN), the monthly interest run adds the interest (INTCALC), and an online bill payment subtracts the payment (CB00)
ACCT-CREDIT-LIMIT	Credit limit, signed, two decimals	Posting rejects a transaction that would take the account over it (CBTRN02C). Set only by the import job and the account update screen
ACCT-CASH-CREDIT-LIMIT	Cash advance limit, signed, two decimals	Used online (authorization, account view and update) and printed by the batch account report (CBACT01C) and the export; posting doesn't check it
ACCT-OPEN-DATE	Date the account was opened, 10 characters	Probably YYYY-MM-DD, kept as text
ACCT-EXPIRAION-DATE	Date the account expires, 10 characters	Probably YYYY-MM-DD, kept as text. Posting rejects transactions after it. The name is misspelled ("EXPIRAION") in the source
ACCT-REISSUE-DATE	Date the account (or its cards) was last reissued, 10 characters	Probably YYYY-MM-DD, kept as text. Only displayed and exported
ACCT-CURR-CYC-CREDIT	Total credits in the current billing cycle, signed, two decimals	Posting adds credit transactions; the monthly interest run resets it to zero when it closes the cycle
ACCT-CURR-CYC-DEBIT	Total debits in the current billing cycle, signed, two decimals	Posting adds debit transactions; the monthly interest run resets it to zero when it closes the cycle
ACCT-ADDR-ZIP	ZIP code, 10 characters	Probably the billing ZIP code. Set only by the import job and read only by the export job - but see the notes: the account update screen overwrites it
ACCT-GROUP-ID	Disclosure group, 10 characters	The interest run uses it to look up the account's interest rate in the disclosure group file (INTCALC)

Notes for modernization

- **Data defect:** the account update screen (COACTUPC, transaction CAUP) writes the record back from its own layout, ACCT-UPDATE-RECORD, which has no ZIP code. Every account update puts the group ID into ACCT-ADDR-ZIP (bytes

103-112) and blanks ACCT-GROUP-ID (bytes 113-122), which the interest run then uses to look up the rate. Confirm with the client whether production has this fix; check the data for group IDs in the ZIP field.
- The account update screen maintains this record through its own copy (ACCT-UPDATE-RECORD), so the field-by-field table doesn't show it as setting these fields. Treat CAUP as a maintainer of every field except the ZIP code.
- Amounts are zoned decimal with an implied two decimals (S9(10)V99): convert to a decimal type, not text. Dates are text and need validating before conversion.
- 178 bytes of the 300-byte record are unused FILLER.

Field	PIC	Bytes	Values	Set by
ACCOUNT-RECORD		1-300		
ACCT-ID	9(11)	1-11		CBIMPORT (CBIMPORT), CBTRN01C
ACCT-ACTIVE-STATUS	X(01)	12-12		CBIMPORT (CBIMPORT)
ACCT-CURR-BAL	S9(10)V99	13-24		CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)
ACCT-CREDIT-LIMIT	S9(10)V99	25-36		CBIMPORT (CBIMPORT)
ACCT-CASH-CREDIT-LIMIT	S9(10)V99	37-48		CBIMPORT (CBIMPORT)
ACCT-OPEN-DATE	X(10)	49-58		CBIMPORT (CBIMPORT)
ACCT-EXPIRAION-DATE	X(10)	59-68		CBIMPORT (CBIMPORT)
ACCT-REISSUE-DATE	X(10)	69-78		CBIMPORT (CBIMPORT)
ACCT-CURR-CYC-CREDIT	S9(10)V99	79-90		CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN)
ACCT-CURR-CYC-DEBIT	S9(10)V99	91-102		CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN)
ACCT-ADDR-ZIP	X(10)	103-112		CBIMPORT (CBIMPORT)
ACCT-GROUP-ID	X(10)	113-122		CBIMPORT (CBIMPORT)

CVACT03Y: CARD-XREF-RECORD

- Stored in AWS.M2.CARDDEMO.CARDXREF.IMPORT: written by CBIMPORT
- Stored in AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX.PATH: looked up by COACTUPC, COACTVWC, COBIL00C, COPAUS0C, COTRN02C
- Stored in AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS: looked up by CBACT04C, CBTRN02C, CBTRN03C, COPAUA0C, COTRN02C; read by CBACT03C, CBEXPORT
- Stored in XREFFILE: looked up by CBTRN01C

The card cross-reference: one record per card, linking the card number to its customer and account. Kept in AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS (keyed by card number), 50 bytes long, with an alternate index path (CARDXREF.VSAM.AIX.PATH) that the online screens use to go from an account to its cards.

Fields

Field	Meaning	Notes
CARD-XREF-RECORD	The whole cross-reference record	Read by posting (POSTTRAN), interest (INTCALC), the transaction report (TRANREPT), and most online screens; written only by the import job
XREF-CARD-NUM	Card number, 16 characters	The file's key. Several programs move a card number into it only to look the record up (authorization COPAUA0C, the add-transaction screen COTRN02C, and CBTRN01C, a transaction validation program no job in the source runs) - they don't change the file
XREF-CUST-ID	Customer number, 9 digits	Points to the customer record (CUST-ID)
XREF-ACCT-ID	Account number, 11 digits	Points to the account record (ACCT-ID). Probably the key of the alternate index, since the account and bill payment screens look cards up by account. The add-transaction screen sets it only to search by account

Notes for modernization

- This file is how every process gets from a card to its account and customer: posting, interest, statements, authorizations, and the online screens all depend on it. In a relational design it becomes foreign keys on the card table, not a separate file.
- Only the import job writes it. Nothing in the source keeps it in step with the card and account files when cards are added or moved, so check how the client maintains it.
- 14 bytes of the 50-byte record are unused FILLER.

Field	PIC	Bytes	Values	Set by
CARD-XREF-RECORD		1-50		
XREF-CARD-NUM	X(16)	1-16		CBIMPORT (CBIMPORT), CBTRN01C, COPAUA0C (transaction CP00), COTRN02C (transaction CT02)
XREF-CUST-ID	9(09)	17-25		CBIMPORT (CBIMPORT)
XREF-ACCT-ID	9(11)	26-36		CBIMPORT (CBIMPORT), COTRN02C (transaction CT02)

CVTRA01Y: TRAN-CAT-BAL-RECORD

- Stored in AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS: read by CBACT04C; updated by CBTRN02C

Field	PIC	Bytes	Values	Set by
TRAN-CAT-BAL-RECORD		1-50		
TRAN-CAT-KEY		1-17		
TRANCAT-ACCT-ID	9(11)	1-11		CBTRN02C (POSTTRAN)
TRANCAT-TYPE-CD	X(02)	12-13		CBTRN02C (POSTTRAN)
TRANCAT-CD	9(04)	14-17		CBTRN02C (POSTTRAN)
TRAN-CAT-BAL	S9(09)V99	18-28		CBTRN02C (POSTTRAN)

CVTRA05Y: TRAN-RECORD

- Stored in AWS.M2.CARDDEMO.SYSTRAN: written by CBACT04C
- Stored in AWS.M2.CARDDEMO.TRANSACT.DALY: read by CBTRN03C
- Stored in AWS.M2.CARDDEMO.TRANSACT.IMPORT: written by CBIMPORT
- Stored in AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS: looked up by COTRN01C; read by CBEXPORT, COTRN00C; updated by COBIL00C, COTRN02C; written by CBTRN02C

One record per card transaction: what kind it is, the amount, the merchant, the card, and when it happened and was processed. Kept in the transaction master file (AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS, keyed by transaction ID), 350 bytes long. The same layout is used for the daily transaction file the report reads (TRANSACT.DALY) and the system transactions the interest run writes (SYSTRAN).

Fields

Field	Meaning	Notes
TRAN-RECORD	The whole transaction record	Written by posting (POSTTRAN), the add-transaction screen (CT02), and bill payment (CB00); the interest run (INTCALC) writes its interest transactions to the separate SYSTRAN file
TRAN-ID	Transaction ID, 16 characters	The master file's key. Posting copies it from the daily transaction; the add-transaction and bill payment screens read the highest ID on file and add 1; the interest run builds one from its run date (PARM) and a counter. The transaction list screen (CT00) sets it only to start browsing
TRAN-TYPE-CD	Transaction type, 2 characters	Probably a code from the transaction type file (CVTRA03Y). The interest run writes '01', bill payment writes '02'
TRAN-CAT-CD	Transaction category within the type, 4 digits	Probably a code from the transaction category file (CVTRA04Y). The interest run writes 5, bill payment writes 2

TRAN-SOURCE	Where the transaction came from, 10 characters	The interest run writes 'System', bill payment writes 'POS TERM'
TRAN-DESC	Description, 100 characters	Bill payment writes 'BILL PAYMENT - ONLINE'; the interest run builds a description that includes the account
TRAN-AMT	Amount, signed, two decimals (zoned decimal)	Bill payment records the whole current balance as the payment
TRAN-MERCHANT-ID	Merchant number, 9 digits	
TRAN-MERCHANT-NAME	Merchant name, 50 characters	
TRAN-MERCHANT-CITY	Merchant city, 50 characters	
TRAN-MERCHANT-ZIP	Merchant ZIP code, 10 characters	
TRAN-CARD-NUM	Card the transaction was made with, 16 characters	Posting uses it to find the account through the card cross-reference
TRAN-ORIG-TS	When the transaction happened, 26 characters	Probably a Db2-style timestamp, YYYY-MM-DD-HH.MM.SS.nnnnnn
TRAN-PROC-TS	When it was processed, 26 characters	Probably the same timestamp format; the transaction report (TRANREPT) selects transactions whose first 10 characters (the date) fall in its date range

Notes for modernization

- **Two ways in:** posting validates daily transactions against the card, account, and credit limit before writing them, but the add-transaction screen (CT02) writes straight to the master file without those checks and without updating the account balance. Confirm with the client whether online-added transactions are meant to bypass posting.
- Transaction type, category, and source are hard-coded literals in the interest run and bill payment. Map them to the type and category reference data in the target, not to constants.
- Transaction IDs are made in three different ways (copied from the daily file, highest-plus-one, run date plus counter). Highest-plus-one can collide when two users add transactions at the same time. The target needs one ID scheme.
- Amounts are zoned decimal with two implied decimals; timestamps are text.
- 20 bytes of the 350-byte record are unused FILLER.

Field	PIC	Bytes	Values	Set by
TRAN-RECORD		1-350		
TRAN-ID	X(16)	1-16		CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00), COTRN00C (transaction CT00), COTRN02C (transaction CT02)
TRAN-TYPE-CD	X(02)	17-18		CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)
TRAN-CAT-CD	9(04)	19-22		CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)
TRAN-SOURCE	X(10)	23-32		CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)
TRAN-DESC	X(100)	33-132		CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)
TRAN-AMT	S9(09)V99	133-143		CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00), COTRN02C (transaction CT02)
TRAN-MERCHANT-ID	9(09)	144-152		CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)

TRAN-MERCHANT-NAME	X(50)	153-202	CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)
TRAN-MERCHANT-CITY	X(50)	203-252	CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)
TRAN-MERCHANT-ZIP	X(10)	253-262	CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)
TRAN-CARD-NUM	X(16)	263-278	CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)
TRAN-ORIG-TS	X(26)	279-304	CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)
TRAN-PROC-TS	X(26)	305-330	CBACT04C (INTCALC), CBIMPORT (CBIMPORT), CBTRN02C (POSTTRAN), COBIL00C (transaction CB00)

CVTRA06Y: DALYTRAN-RECORD

- Stored in AWS.M2.CARDDEMO.DALYTRAN.PS: read by CBTRN02C
- Stored in DALYTRAN: read by CBTRN01C

Field	PIC	Bytes	Values	Set by
DALYTRAN-RECORD		1-350		
DALYTRAN-ID	X(16)	1-16		
DALYTRAN-TYPE-CD	X(02)	17-18		
DALYTRAN-CAT-CD	9(04)	19-22		
DALYTRAN-SOURCE	X(10)	23-32		
DALYTRAN-DESC	X(100)	33-132		
DALYTRAN-AMT	S9(09)V99	133-143		
DALYTRAN-MERCHANT-ID	9(09)	144-152		
DALYTRAN-MERCHANT-NAME	X(50)	153-202		
DALYTRAN-MERCHANT-CITY	X(50)	203-252		
DALYTRAN-MERCHANT-ZIP	X(10)	253-262		
DALYTRAN-CARD-NUM	X(16)	263-278		
DALYTRAN-ORIG-TS	X(26)	279-304		
DALYTRAN-PROC-TS	X(26)	305-330		

Findings

Check **2 job(s) the schedule doesn't make wait for the data they read**

Compared the client's scheduler definitions with which jobs write and read each dataset and Db2 table.

- In schedule ID 030, nothing makes CREASTMT wait for POSTTRAN, though it reads what POSTTRAN writes: they can run in either order, or at the same time. (AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS, AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS)
- In schedule ID 030, nothing makes READACCT wait for POSTTRAN, though it reads what POSTTRAN writes: they can run in either order, or at the same time. (AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS)

Check **5 dataset(s) in scope are also used by jobs outside it**

Moving these datasets affects the listed jobs too: migrate them together, or keep the data in sync during the transition.

- AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS - also used by ACCTFILE, CBEXPORT, CREASTMT, INTCALC, READACCT
- AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS - also used by CBEXPORT, CREASTMT, INTCALC, READXREF, TRANREPT, XREFFILE
- AWS.M2.CARDDEMO.DALYREJS - also used by DALYREJS
- AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS - also used by INTCALC, PRTCATBL, TCATBALF
- AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS - also used by CBEXPORT, COMBTRAN, CREASTMT, TRANBKP, TRANFILE, TRANIDX, TRANREPT

Note **2 job(s) reading data written in another scheduler flow**

Compared the client's scheduler definitions with which jobs write and read each dataset and Db2 table.

- PRTCATBL reads what POSTTRAN writes, but they're in different flows (schedule ID 030, schedule ID 031): the order depends on when each flow runs - confirm it with the client. (AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS)
- POSTTRAN reads what TCATBALF writes, but they're in different flows (schedule ID 032, schedule ID 030): the order depends on when each flow runs - confirm it with the client. (AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS)

Note **From the review of job POSTTRAN**

- **Posting rules to confirm with the business:** the credit-limit check uses the current cycle's credits minus debits plus the new amount, not the account balance; and when a transaction is both over limit and past expiration, only the expiration reason is recorded.
- **Partial updates on failure:** an account update that fails is silently ignored after the category balance has been updated and while the transaction is still written; a failure part-way through the run leaves some transactions posted and others not, with no restart point. A modern version would post each transaction atomically (one database transaction) or make the run restartable.
- **Opening the master for output** depends on how the VSAM file is defined; confirm the actual run procedure (the scheduler definitions in the source aren't analyzed yet).
- A natural first candidate for a service: "validate and post a card transaction" is a clear unit of business logic that online and batch could share.

Note **From the review of program CBTRN02C**

- **One reason per reject:** when a transaction is both over limit and past expiration, only reason 103 is kept (the second check overwrites the first).
- **A failed account update is silently ignored:** if the account rewrite fails, the program sets reason 109 but doesn't reject or stop - by then the category balance has already been updated and the transaction is still written, so the files can disagree.
- **The transaction master is opened for OUTPUT,** which for a VSAM KSDS only works when the file is empty or defined as reusable; the scheduling around POSTTRAN has to make sure of that, or the open fails.
- The over-limit rule uses cycle credits and debits rather than the balance - worth confirming with the business that this is the intended rule before carrying it over.
- The error message for closing the rejects file shows the cross-reference file's status (copy-and-paste slip in 9300-DALYREJS-CLOSE).
- Posting is a read-validate-update loop over keyed files with no restart logic: a failure part-way leaves some transactions posted and others not. A modern version would post each transaction in one database transaction, or make the batch restartable.

Appendix: sources and method

About the source

A sample assessment of AWS CardDemo, an open-source mainframe credit card application (Apache License 2.0, Copyright Amazon.com, Inc. or its affiliates), at commit 59cc6c2 of https://github.com/aws-samples/aws-mainframe-modernization-carddemo. No client code is included.

Libraries, in search order

#	Library
1	APP

How this report was made

- The inventory, scopes, data access, flow diagrams, walkthroughs, and findings are built from the source alone - JCL, PROCs, programs, copybooks, BMS maps, DDL, and the CICS CSD - with no AI.
- The plain-English summaries were drafted by Claude from the source and the facts above, then reviewed and corrected by us; only reviewed summaries appear.
- Roadmap approaches marked suggested follow simple rules from the facts (size, IMS use, data access, utility-only jobs) and are a starting point for discussion.