Modernization assessment - CARDDEMO

Prepared 19 November 2026 · All members of the libraries provided · 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

46	25	45	41,347
batch jobs	transactions / CICS flows	programs in scope	lines of source

71	3
datasets	findings to resolve

This assessment covers 46 batch job(s) and 25 online transaction(s) or CICS flow(s), 45 application program(s), and 41,347 lines of source. Approach: 42 refactor, 17 expose as API, 7 rehost, 5 retire (suggested, to confirm). 3 finding(s) should be resolved before estimating: 1 member(s) referenced but not in the source; 1 program(s) use ALTER; 1 file(s) a program uses that its job doesn't define.

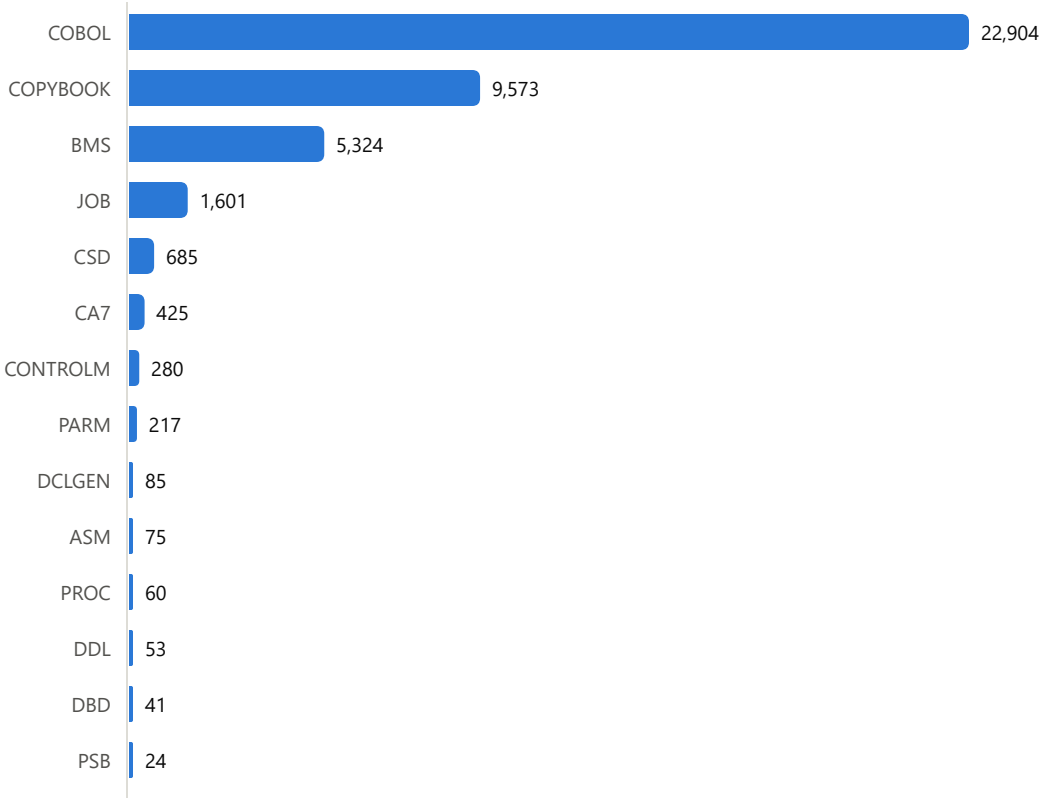
What they do

- **CREASTMT** - CREASTMT produces the customer statements. It rebuilds a copy of the transaction master keyed by card number, clears out the previous run's statements, and then prints a statement for every card - customer name and address, account ID, current balance, FICO score, and the card's transactions with a total - as both a plain-text file and an HTML file.
- **INTCALC** - INTCALC is the monthly interest run. For every account it charges interest on the account's balance in each transaction category, at the annual rate set for the account's disclosure group (or the default group), and writes each charge as a system-generated interest transaction. It adds the account's total interest to its current balance and resets the account's current-cycle credit and debit totals, closing the billing cycle. The interest transactions go to a new file; a later job (COMBTRAN) loads them into the transaction master.
- **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.
- **TRANREPT** - TRANREPT produces the Daily Transaction Report. It takes a backup copy of the transaction master file, keeps only the transactions processed within a date range, sorts them by card number, and prints a report listing each transaction with its account, type, category, source, and amount, with totals per card, per page, and overall. Each run creates new generations of the backup, the filtered extract, and the report, so earlier runs are kept.
- **CAUP** - CAUP lets a user look up an account and change it: the account's status, dates, credit limits, and balances, and the account holder's name, address, phone numbers, SSN, date of birth, FICO score, and EFT account. Changes are validated field by field, shown back for confirmation, and saved to the account and customer records together, with protection against two people updating the same account at once.
- **CC00** - CC00 is how every CardDemo user starts: the sign-on screen. The user types a user ID and password; the program checks them against the user security file and sends administrators to the admin menu and everyone else to the main menu, carrying their identity and role to every screen after it. PF3 ends the session.
- **CM00** - CM00 is the main menu for regular users. It lists the eleven things a user can do - view or update accounts, list, view, or update credit cards, list, view, or add transactions, run transaction reports, pay bills, and view pending authorizations - and starts the chosen function. Each function returns here when the user presses PF3, and PF3 here returns to sign-on.
- **CT02** - CT02 lets a user enter a card transaction by hand. The user picks the card (by account or card number), types the transaction details - type, category, source, description, amount, dates, merchant - confirms with Y, and the transaction is written to the transaction master with the next transaction number.

Inventory

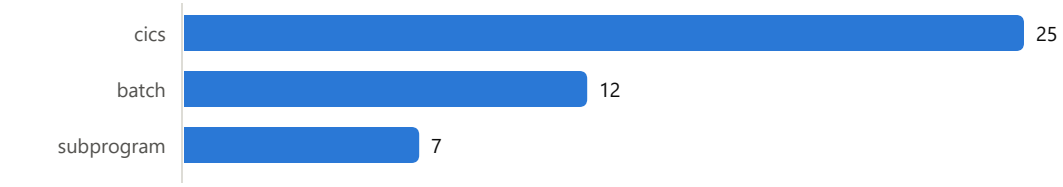
Everything loaded: 44 COBOL program(s) and 2 assembler module(s), 62 copybook(s), 46 job(s), 2 PROC(s), 25 CICS transaction(s) defined.

Lines of source by type (excluding comments and blanks)

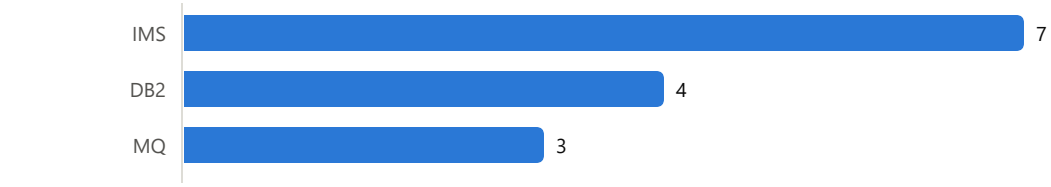


Type	Code	Comment
COBOL	22,904	3,906
COPYBOOK	9,573	989
BMS	5,324	430
JOB	1,601	1,304
CSD	685	0
CA7	425	0
CONTROLM	280	0
PARM	217	0
DCLGEN	85	106
ASM	75	39
PROC	60	54
DDL	53	0
DBD	41	94
PSB	24	64

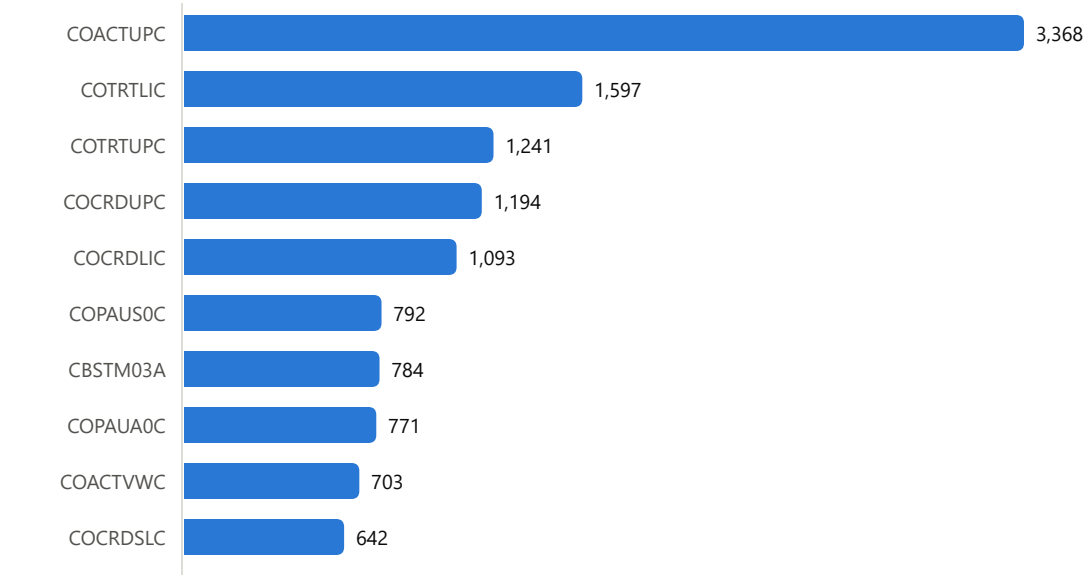
COBOL programs by kind



COBOL programs using



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.

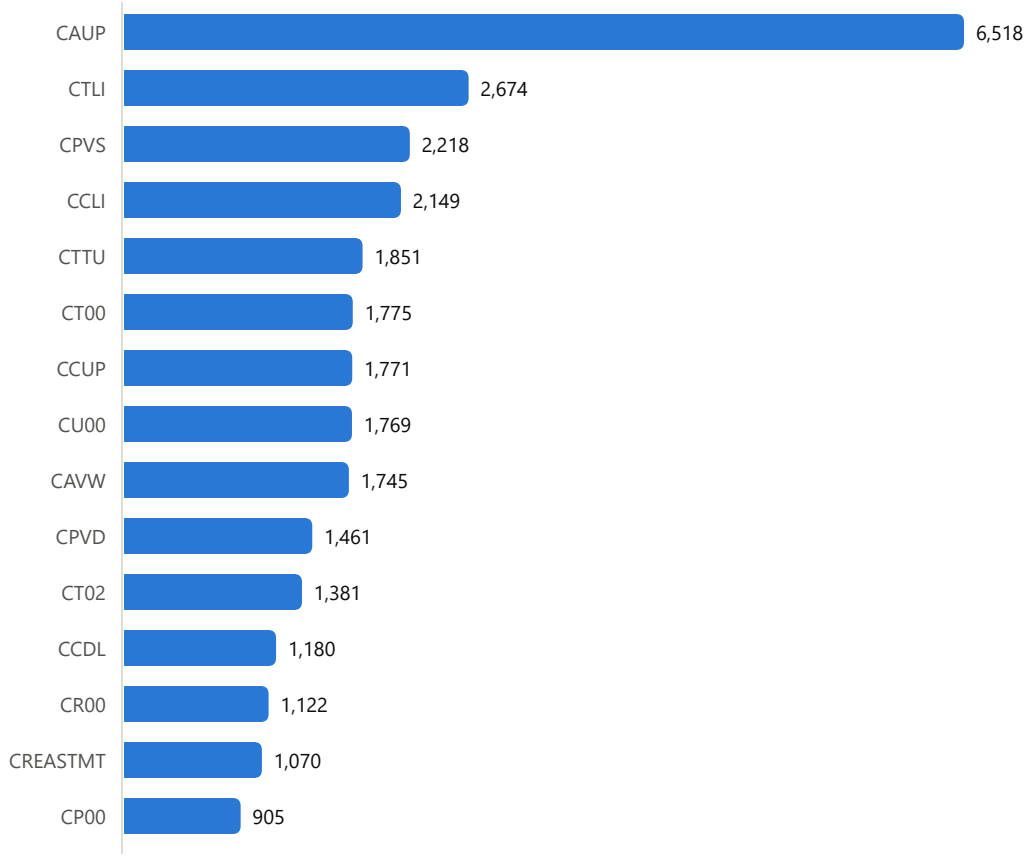
Program	Decision points	IF	EVALUATE	Loops	PERFORM depth	Paragraphs	GO TO
COACTUPC	420	274	11	0	4	101	66
COTRTUPC	174	51	14	0	4	65	23
COTRTLIC	168	86	17	6	5	58	28
COCRDUPC	152	72	9	0	3	47	21
COCRDLIC	124	59	10	3	3	41	16
COCRDSLC	73	33	5	0	3	36	9
COUSR00C	73	25	8	4	5	16	0
COTRN00C	72	26	8	4	5	16	0

COACTVWC	67	28	6	0	3	37	9
COTRN02C	64	14	13	0	6	18	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
CAUP	transaction	2	16	1	3			6,518	Expose as API suggested
CTLI	transaction	1	11	1		1		2,674	Expose as API suggested
CPVS	transaction	1	12	1	3		2	2,218	Rehost suggested
CCLI	transaction	1	9	1	1			2,149	Expose as API suggested
CTTU	transaction	1	11	1		1		1,851	Expose as API suggested
CT00	transaction	1	6	1	1			1,775	Expose as API suggested
CCUP	transaction	1	11	1	1			1,771	Expose as API suggested
CU00	transaction	1	6	1	1			1,769	Expose as API suggested
CAVW	transaction	1	13	1	3			1,745	Expose as API suggested
CPVD	transaction	2	8	1		1	2	1,461	Rehost suggested
CT02	transaction	2	8	1	3			1,381	Expose as API suggested
CCDL	transaction	1	11	1	2			1,180	Expose as API suggested
CR00	transaction	2	6	1				1,122	Refactor suggested
CREASTMT	job	2	4		8			1,070	Refactor suggested
CP00	transaction	1	8		3		2	905	Rehost suggested
CT01	transaction	1	6	1	1			831	Expose as API suggested
CB00	transaction	1	8	1	3			774	Expose as API suggested

CM00	transaction	1	7	1				751	Refactor suggested
TRANREPT	job	1	5		8			708	Refactor suggested
POSTTRAN	job	1	5		6			697	Refactor suggested
CA00	transaction	1	7	1				695	Refactor suggested
CU02	transaction	1	6	1	1			683	Expose as API suggested
INTCALC	job	1	5		6			622	Refactor suggested
CU03	transaction	1	6	1	1			603	Expose as API suggested
CC00	transaction	1	6	1	1			581	Expose as API suggested
CU01	transaction	1	6	1	1			573	Expose as API suggested
CBEXPORT	job	1	6		6			564	Refactor suggested
CDRA	transaction	1	1		1			514	Expose as API suggested
CBIMPORT	job	1	6		6			505	Refactor suggested
READACCT	job	2	2		4			497	Refactor suggested
CDRD	transaction	1						409	Refactor suggested
LOADPADB	job	1	4		2		2	340	Rehost suggested
CBPAUP0J	job	1	2				2	337	Rehost suggested
UNLDPADB	job	1	4		4		2	312	Rehost suggested
UNLDGSAM	job	1	6		4		4	311	Rehost suggested
CREADB21	job							242	Refactor suggested
MNTTRDB2	job	1			1	1		200	Refactor suggested
READCUST	job	1	1		1			159	Refactor suggested
READCARD	job	1	1		1			146	Refactor suggested
READXREF	job	1	1		1			144	Refactor suggested
CBADMCDJ	job				1			121	Refactor suggested
TRANEXTR	job				4	2		83	Refactor suggested
CARDFILE	job				4			81	Refactor suggested
TRANFILE	job				4			81	Refactor suggested
ESDSRRDS	job				3			68	Refactor suggested
XREFFILE	job				4			68	Refactor suggested
TRANBKP	job				3			65	Refactor suggested
PRTCATBL	job				3			63	Refactor suggested
DEFGDGD	job				6			52	Refactor suggested
CUSTFILE	job				2			48	Refactor suggested
DUSRSECJ	job				2			48	Refactor suggested
DEFGDGB	job				6			41	Retire suggested
ACCTFILE	job				2			36	Refactor suggested
DISCGRP	job				2			36	Refactor suggested
TCATBALF	job				2			36	Refactor suggested
TRANCATG	job				2			36	Refactor suggested
TRANTYPE	job				2			36	Refactor suggested
WAITSTEP	job	2						35	Refactor suggested
DBPAUTP0	job				3			32	Refactor suggested
TRANIDX	job				3			30	Refactor suggested
COMBTRAN	job				4			27	Refactor suggested
DEFCUST	job				2			24	Retire suggested
INTRDRJ1	job				3			14	Refactor suggested

TXT2PDF1	job	2	14	Refactor suggested
FTPJCL	job		13	Refactor suggested
CLOSEFIL	job		12	Refactor suggested
OPENFIL	job		12	Refactor suggested
DALYREJS	job	1	10	Retire suggested
REPTFILE	job	1	10	Retire suggested
INTRDRJ2	job	2	9	Refactor suggested
CDV1	transaction		0	Retire suggested

Modernization roadmap

Approach	Meaning	Count
Rehost	Move as is to a mainframe-compatible runtime first; change the code later.	7
Refactor	Rewrite the logic on the target platform (services, SQL, a reporting or ETL tool).	42
Expose as API	Keep the business logic, put it behind a service interface, and replace the 3270 screen.	17
Retire	Not needed on the target platform, or no longer used.	5

71 of 71 approach(es) are suggestions from the facts below, to confirm with you before planning.

Rehost (7)

Name	Why
CBPAUP0J suggested	Uses IMS databases; rehost first and plan the data migration separately.
CP00 suggested	905 lines with IMS or heavy logic: rehost, then expose services.
CPVD suggested	1,461 lines with IMS or heavy logic: rehost, then expose services.
CPVS suggested	2,218 lines with IMS or heavy logic: rehost, then expose services.
LOADPADB suggested	Uses IMS databases; rehost first and plan the data migration separately.
UNLDGSAM suggested	Uses IMS databases; rehost first and plan the data migration separately.
UNLDPADB suggested	Uses IMS databases; rehost first and plan the data migration separately.

Refactor (42)

Name	Why
ACCTFILE suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
CA00 suggested	Screen and navigation only (no data access): rebuild in the new user interface.
CARDFILE suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
CBADMCDJ suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
CBEXPORT suggested	1 program(s), 564 lines: small enough to rewrite as a batch service, query, or report on the target platform.
CBIMPORT suggested	1 program(s), 505 lines: small enough to rewrite as a batch service, query, or report on the target platform.
CDRD suggested	Screen and navigation only (no data access): rebuild in the new user interface.
CLOSEFIL suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
CM00 suggested	Screen and navigation only (no data access): rebuild in the new user interface.
COMBTRAN suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
CR00 suggested	Screen and navigation only (no data access): rebuild in the new user interface.
CREADB21 suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
CREASTMT suggested	2 program(s), 1,070 lines: small enough to rewrite as a batch service, query, or report on the target platform.
CUSTFILE suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
DBPAUTP0 suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
DEFGDGD suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
DISCGRP suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
DUSRSECJ suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
ESDSRRDS suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
FTPJCL suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
INTCALC suggested	1 program(s), 622 lines: small enough to rewrite as a batch service, query, or report on the target platform.
INTRDRJ1 suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
INTRDRJ2 suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

MNTTRDB2 suggested	1 program(s), 200 lines: small enough to rewrite as a batch service, query, or report on the target platform.
OPENFIL suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
POSTTRAN suggested	1 program(s), 697 lines: small enough to rewrite as a batch service, query, or report on the target platform.
PRTCATBL suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
READACCT suggested	2 program(s), 497 lines: small enough to rewrite as a batch service, query, or report on the target platform.
READCARD suggested	1 program(s), 146 lines: small enough to rewrite as a batch service, query, or report on the target platform.
READCUST suggested	1 program(s), 159 lines: small enough to rewrite as a batch service, query, or report on the target platform.
READXREF suggested	1 program(s), 144 lines: small enough to rewrite as a batch service, query, or report on the target platform.
TCATBALF suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
TRANBKP suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
TRANCATG suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
TRANEXTR suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
TRANFILE suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
TRANIDX suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
TRANREPT suggested	1 program(s), 708 lines: small enough to rewrite as a batch service, query, or report on the target platform.
TRANTYPE suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
TXT2PDF1 suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.
WAITSTEP suggested	2 program(s), 35 lines: small enough to rewrite as a batch service, query, or report on the target platform.
XREFFILE suggested	Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

Expose as API (17)

Name	Why
CAUP suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CAVW suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CB00 suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CC00 suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CCDL suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CCLI suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CCUP suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CDRA suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CT00 suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CT01 suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CT02 suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CTLI suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CTTU suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CU00 suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

CU01 suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CU02 suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.
CU03 suggested	Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

Retire (5)

Name	Why
CDV1 suggested	PROGRAM COCRDSEC isn't in the source: confirm with the client whether it's still used, and retire it if not.
DALYREJS suggested	Only IBM utilities that define and delete data sets - housekeeping a database-backed platform doesn't need.
DEFCUST suggested	Only IBM utilities that define and delete data sets - housekeeping a database-backed platform doesn't need.
DEFGDGB suggested	Only IBM utilities that define and delete data sets - housekeeping a database-backed platform doesn't need.
REPTFILE suggested	Only IBM utilities that define and delete data sets - housekeeping a database-backed platform doesn't need.

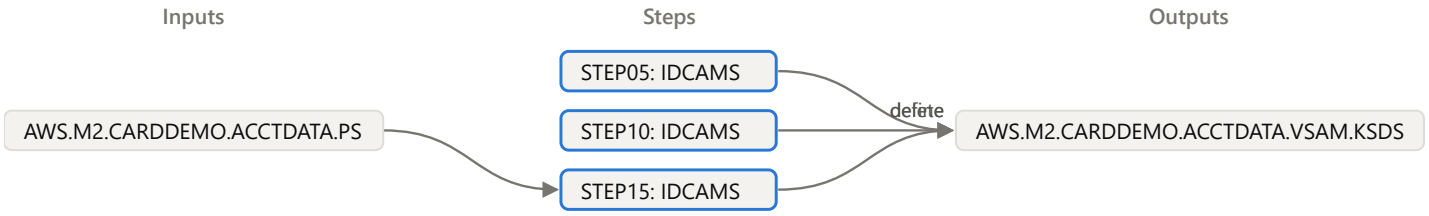
Job and transaction details

Job ACCTFILE

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	3	1	2
jobs	steps	system programs	datasets

36
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job ACCTFILE

Step 1 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE ACCOUNT VSAM FILE IF ONE ALREADY EXISTS
- Deletes: AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS
- Control statements: `DELETE AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS CLUSTER IF MAXCC LE 08 THEN SET MAXCC = 0`

Step 2 - STEP10: runs IDCAMS (IBM utility)

- Comment: DEFINE ACCOUNT VSAM FILE
- Defines: AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS
- Control statements: `DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS) CYLINDERS(1 5) VOLUMES(AWSHJ1 ) KEYS(11 0) RECORDSIZE(300 300) SHAREOPTIONS(2 3) ERASE`

Step 3 - STEP15: runs IDCAMS (IBM utility)

- Comment: COPY DATA FROM FLAT FILE TO VSAM FILE
- Copies AWS.M2.CARDDEMO.ACCTDATA.PS into AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS
- Reads: AWS.M2.CARDDEMO.ACCTDATA.PS (DD ACCTDATA)
- Writes: AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS (DD ACCTVSAM)
- Control statements: `REPRO INFILE(ACCTDATA) OUTFILE(ACCTVSAM)`

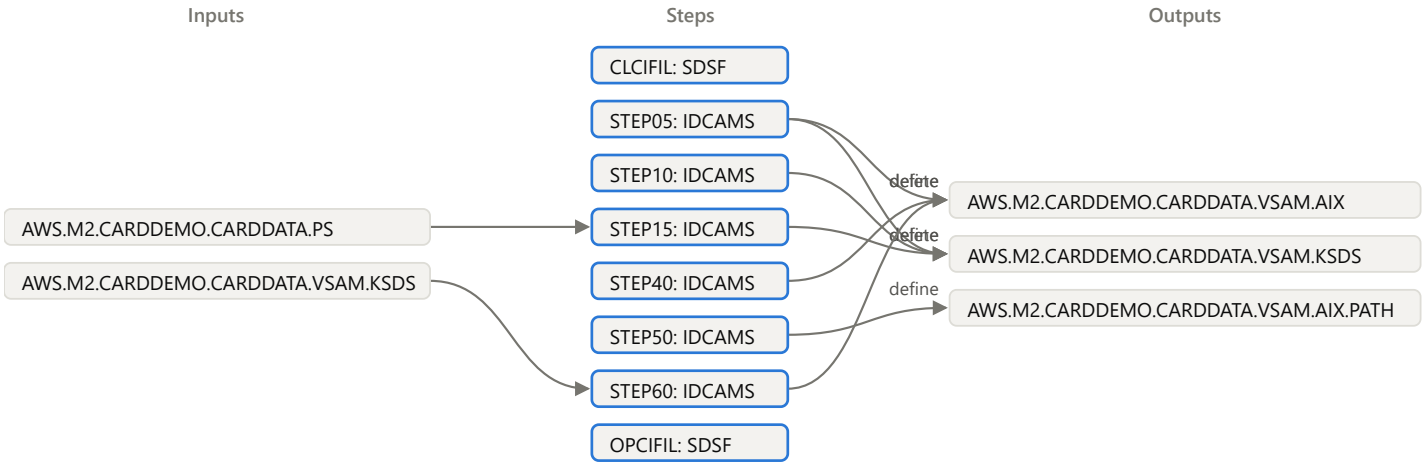
Inputs from outside the job: AWS.M2.CARDDEMO.ACCTDATA.PS **Outputs it leaves behind:** AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS

Job CARDFILE

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	8	2	4
jobs	steps	system programs	datasets

81
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job CARDFILE

Step 1 - CLCIFIL: runs SDSF (IBM utility)

- Comment: Close files in CICS region

Step 2 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE CARD DATA VSAM FILE IF ONE ALREADY EXISTS
- Deletes: AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX, AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS
- Control statements: `DELETE AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS CLUSTER IF MAXCC LE 08 THEN SET MAXCC = 0 DELETE AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX ALTERNATEINDEX IF MAXCC LE 08 THEN SET MAXCC = 0`

Step 3 - STEP10: runs IDCAMS (IBM utility)

- Comment: DEFINE CARD DATA VSAM FILE
- Defines: AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS
- Control statements: `DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS) CYLINDERS(1 5) VOLUMES(AWSHJ1 ) KEYS(16 0) RECORDSIZE(150 150) SHAREOPTIONS(2 3) ERASE`

Step 4 - STEP15: runs IDCAMS (IBM utility)

- Comment: COPY DATA FROM FLAT FILE TO VSAM FILE
- Copies AWS.M2.CARDDEMO.CARDDATA.PS into AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS
- Reads: AWS.M2.CARDDEMO.CARDDATA.PS (DD CARDDATA)

- Writes: AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS (DD CARDVSAM)
- Control statements: `REPRO INFILE(CARDDATA) OUTFILE(CARDVSAM)`

Step 5 - STEP40: runs IDCAMS (IBM utility)

- Comment: CREATE ALTERNATE INDEX ON ACCT ID
- Defines: AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX
- Control statements: `DEFINE ALTERNATEINDEX (NAME(AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX)-
  RELATE(AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS) KEYS(11 16) NONUNIQUEKEY UPGRADE RECORDSIZE(150,150)
  VOLUMES(AWSHJ1) CYLINDERS(5,1))`

Step 6 - STEP50: runs IDCAMS (IBM utility)

- Comment: DEFINE PATH IS USED TO RELATE THE ALTERNATE INDEX TO BASE CLUSTER
- Defines: AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX.PATH
- Control statements: `DEFINE PATH (NAME(AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX.PATH)
  PATHENTRY(AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX))`

Step 7 - STEP60: runs IDCAMS (IBM utility)

- Comment: BUILD ALTERNATE INDEX CLUSTER
- BLDINDEX AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS into AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX
- Reads: AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS
- Writes: AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX
- Control statements: `BLDINDEX INDATASET(AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS)
  OUTDATASET(AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX)`

Step 8 - OPCIFIL: runs SDSF (IBM utility)

- Comment: Open files in CICS region

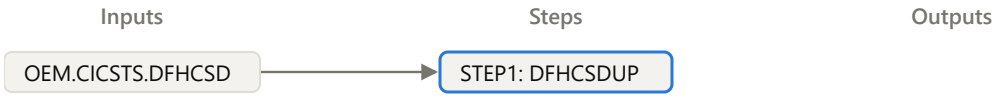
Inputs from outside the job: AWS.M2.CARDDEMO.CARDDATA.PS **Outputs it leaves behind:** AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX

Job CBADMCDJ

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	1	1	1
jobs	steps	system programs	datasets

121
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job CBADMCDJ

Step 1 - STEP1: runs DFHCSDUP (IBM utility)

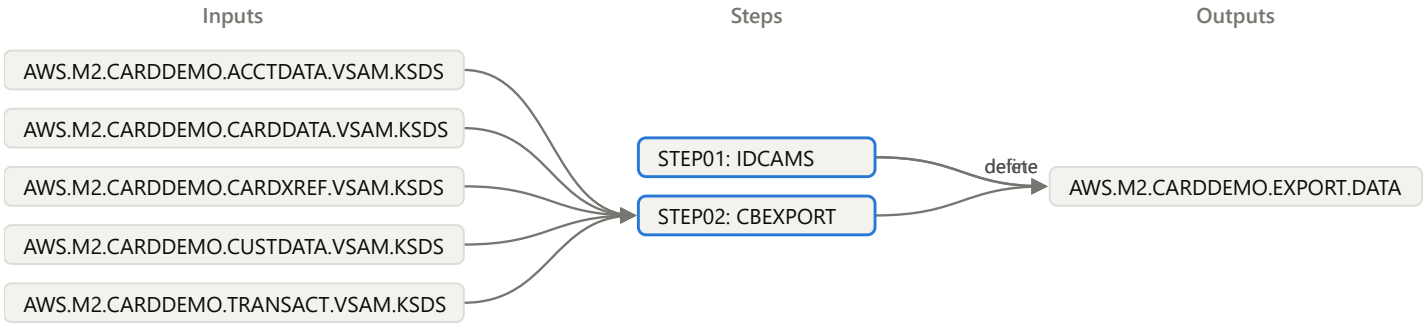
- Comment: Create Resources for Card Demo application / SET PARMS FOR THIS JOB:
- Uses: OEM.CICSTS.DFHCSD (DD DFHCSD)

Inputs from outside the job: none **Outputs it leaves behind:** none

Job CBEXPORT

Refactor suggested - 1 program(s), 564 lines: small enough to rewrite as a batch service, query, or report on the target platform.

1	2	1	2
jobs	steps	programs	system programs
6	**6**	**6**	**564**
copybooks	datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job CBEXPORT

Step 1 - STEP01: runs IDCAMS (IBM utility)

- Comment: EXPORT CUSTOMER DATA FROM VSAM FILES TO MULTI-RECORD EXPORT FILE / FOR BRANCH MIGRATION OR DATA TRANSFER PURPOSES / STEP 1: DEFINE VSAM CLUSTER FOR EXPORT FILE
- Defines: AWS.M2.CARDDEMO.EXPORT.DATA
- Deletes: AWS.M2.CARDDEMO.EXPORT.DATA
- Control statements: `DELETE AWS.M2.CARDDEMO.EXPORT.DATA CLUSTER PURGE SET MAXCC = 0 DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.EXPORT.DATA) INDEXED KEYS(4 28) RECORDSIZE(500 500) CYLINDERS(10 5) FREESPACE(10 10)`

Step 2 - STEP02: runs CBEXPORT

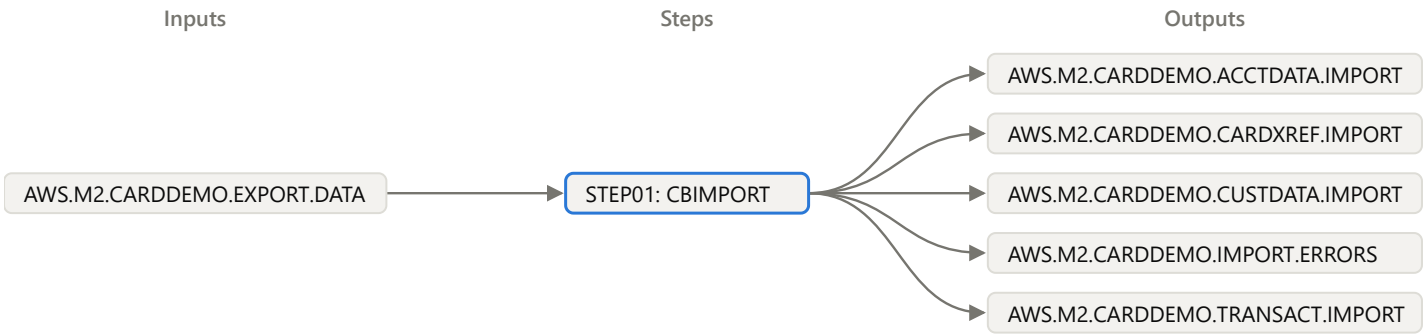
- Comment: STEP 2: RUN EXPORT PROGRAM
- CBEXPORT's comments: Program : CBEXPORT.CBL / Application : CardDemo / Type : BATCH COBOL Program / Function : Export Customer Data for Branch Migration
- Reads: AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS (DD ACCTFILE), AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS (DD CARDFILE), AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS (DD XREFFILE), AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS (DD CUSTFILE), AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS (DD TRANSACT)
- Writes: AWS.M2.CARDDEMO.EXPORT.DATA (DD EXPFILE)

Inputs from outside the job: AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS, AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS, AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS, AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS, AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS **Outputs it leaves behind:** AWS.M2.CARDDEMO.EXPORT.DATA

Job CBIMPORT

Refactor suggested - 1 program(s), 505 lines: small enough to rewrite as a batch service, query, or report on the target platform.

1 jobs	1 steps	1 programs	1 system programs
6 copybooks	6 datasets	6 record layouts	505 lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job CBIMPORT

Step 1 - STEP01: runs CBIMPORT

- Comment: IMPORT CUSTOMER DATA FROM MULTI-RECORD EXPORT FILE AND SPLIT / INTO SEPARATE NORMALIZED FILES FOR TARGET SYSTEM
- CBIMPORT's comments: Program : CBIMPORT.CBL / Application : CardDemo / Type : BATCH COBOL Program / Function : Import Customer Data from Branch Migration Export
- Reads: AWS.M2.CARDDEMO.EXPORT.DATA (DD EXPFILE)
- Creates: AWS.M2.CARDDEMO.ACCTDATA.IMPORT (DD ACCTOUT), AWS.M2.CARDDEMO.CARDXREF.IMPORT (DD XREFOUT), AWS.M2.CARDDEMO.CUSTDATA.IMPORT (DD CUSTOUT), AWS.M2.CARDDEMO.IMPORT.ERRORS (DD ERROUT), AWS.M2.CARDDEMO.TRANSACT.IMPORT (DD TRNXOUT)

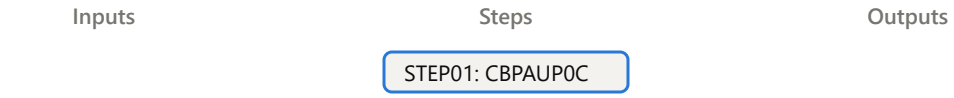
Inputs from outside the job: AWS.M2.CARDDEMO.EXPORT.DATA **Outputs it leaves behind:** AWS.M2.CARDDEMO.ACCTDATA.IMPORT, AWS.M2.CARDDEMO.CARDXREF.IMPORT, AWS.M2.CARDDEMO.CUSTDATA.IMPORT, AWS.M2.CARDDEMO.IMPORT.ERRORS, AWS.M2.CARDDEMO.TRANSACT.IMPORT

Job CBPAUP0J

Rehost suggested - Uses IMS databases; rehost first and plan the data migration separately.

1	1	1	2
jobs	steps	programs	copybooks

2	337
IMS segments	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job CBPAUP0J

Step 1 - STEP01: runs CBPAUP0C, through DFSRRC00

- Comment: EXECUTE IMS PROGRAM TO DELETE EXPIRED AUTHORIZATIONS
- CBPAUP0C's comments: Program : CBPAUP0C.CBL / Application : CardDemo - Authorization Module / Type : BATCH COBOL IMS Program / Function : Delete Expired Pending Authoriation Messages
- Reads and deletes from IMS segment PAUTDTL1
- Reads and deletes from IMS segment PAUTSUM0

Inputs from outside the job: none **Outputs it leaves behind:** none

Job CLOSEFIL

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	1	1	12
jobs	steps	system programs	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job CLOSEFIL

Step 1 - CLCIFIL: runs SDSF (IBM utility)

- Comment: Close files in CICS region

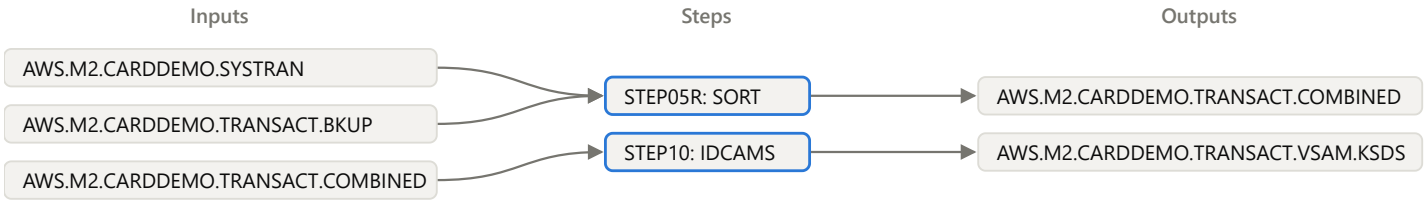
Inputs from outside the job: none **Outputs it leaves behind:** none

Job COMBTRAN

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	2	2	4
jobs	steps	system programs	datasets

27
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job COMBTRAN

Step 1 - STEP05R: runs SORT (IBM utility)

- Comment: Sort current transaction file and system generated transactions
- Sorts AWS.M2.CARDDEMO.TRANSACT.BKUP into AWS.M2.CARDDEMO.TRANSACT.COMBINED
- Sorts AWS.M2.CARDDEMO.SYSTRAN into AWS.M2.CARDDEMO.TRANSACT.COMBINED
- Reads: AWS.M2.CARDDEMO.SYSTRAN (DD SORTIN), AWS.M2.CARDDEMO.TRANSACT.BKUP (DD SORTIN)
- Creates: AWS.M2.CARDDEMO.TRANSACT.COMBINED (DD SORTOUT)
- Sort keys: TRAN-ID (ascending, positions 1-16).
- Control statements: `SORT FIELDS=(TRAN-ID,A)`

Step 2 - STEP10: runs IDCAMS (IBM utility)

- Comment: Load combined file to transaction master
- Copies AWS.M2.CARDDEMO.TRANSACT.COMBINED into AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
- Reads: AWS.M2.CARDDEMO.TRANSACT.COMBINED (DD TRANSACT)
- Writes: AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS (DD TRANVSAM)
- Control statements: `REPRO INFILE(TRANSACT) OUTFILE(TRANVSAM)`

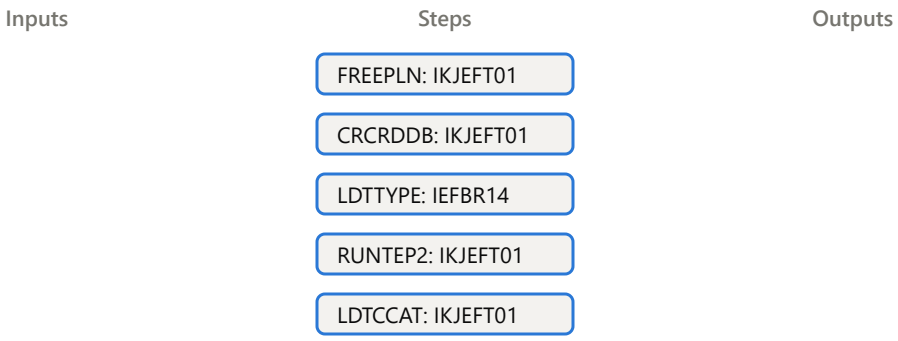
Inputs from outside the job: AWS.M2.CARDDEMO.SYSTRAN, AWS.M2.CARDDEMO.TRANSACT.BKUP **Outputs it leaves behind:** AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS

Job CREADB21

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	5	2	6
jobs	steps	system programs	control cards

242
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job CREADB21

From the job's comments: Create CARDDEMO Database in DAZ1 subsystem (SSID=DAZ1) / You can run the SQLs linked here through SPUFI / If you prefer JCL, you can run them from this job / You may need to bind the utilities used first though / See AWS.M2.CARDDEMO.JCL.UTIL(BINDTIAD) to see how / SET PARMS FOR THIS JOB: / STEP 00 : Free existing plans and packages / : It ends with RC 8 if not existing.

Step 1 - FREEPLN: runs IKJEFT01

- Comment: STEP 00 : Free existing plans and packages / : It ends with RC 8 if not existing. / : So dont run it if creating new database
- Reads control card DB2FREE (DD SYSTSIN)

Step 2 - CRCRDDB: runs IKJEFT01

- Comment: STEP 10 : Use Utility DSNTIAD to create the database / This uses an existing STOGROUP AWST1STG / You would have to create it if not available
- Reads control card DB2TIAD1 (DD SYSTSIN)
- Reads control card DB2CREAT (DD SYSIN)

Step 3 - LDTTYPE: runs IEFBR14 (IBM utility)

- Comment: STEP 20 : Load the transaction Type table / using DSNTEP4 utility

Step 4 - RUNTEP2: runs IKJEFT01

- Reads control card DB2TEP41 (DD SYSTSIN)
- Reads control card DB2LTTYP (DD SYSIN)

Step 5 - LDTCCAT: runs IKJEFT01

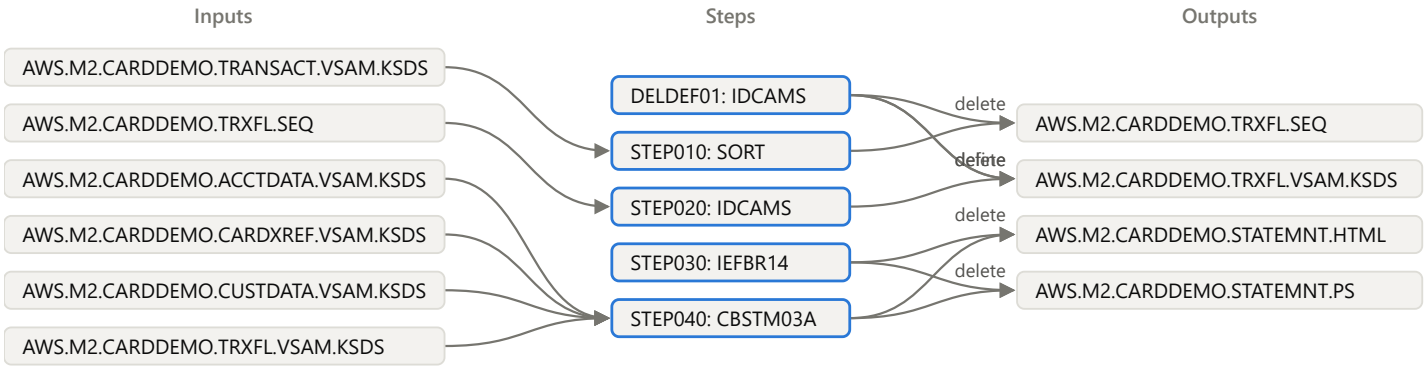
- Comment: STEP 30 : Load Transaction Type Category table / using DSNTEP4 utility
- Reads control card DB2TEP41 (DD SYSTSIN)
- Reads control card DB2LTCAT (DD SYSIN)

Inputs from outside the job: none **Outputs it leaves behind:** none

Job CREASTMT

Refactor suggested - 2 program(s), 1,070 lines: small enough to rewrite as a batch service, query, or report on the target platform.

1 jobs	**5** steps	**2** programs	**4** system programs
4 copybooks	**8** datasets	**23** record layouts	**1,070** lines of source



What it does

CREASTMT produces the customer statements. It rebuilds a copy of the transaction master keyed by card number, clears out the previous run's statements, and then prints a statement for every card - customer name and address, account ID, current balance, FICO score, and the card's transactions with a total - as both a plain-text file and an HTML file.

Step by step

1. **Clear the work files (DELDEF01, IDCAMS).** Deletes the previous card-keyed transaction files (`TRXFL.SEQ` and `TRXFL.VSAM.KSDS`) and defines a new, empty VSAM file keyed by card number and transaction ID (32-byte key, 350-byte records).
2. **Re-key the transactions by card (STEP010, SORT).** Sorts the whole transaction master by card number (TRAN-CARD-NUM) then transaction ID (TRAN-ID), and moves the card number to the front of each record so it becomes part of the key, writing `TRXFL.SEQ` .
3. **Load the card-keyed file (STEP020, IDCAMS REPRO).** Copies the sorted file into the new VSAM file `TRXFL.VSAM.KSDS` .
4. **Remove last run's statements (STEP030, IEFBR14).** Deletes the previous text and HTML statement files.
5. **Print the statements (STEP040, CBSTM03A with its file-access subroutine CBSTM03B).** For each card in the cross-reference, reads the customer and account and writes a statement with the card's transactions and their total, in text (`STATEMNT.PS`) and HTML (`STATEMNT.HTML`).

What it needs

- The transaction master (`TRANSACT.VSAM.KSDS`) after posting and the interest load, the card cross-reference, the customer file (`CUSTDATA.VSAM.KSDS`), and the account master.

- To run after the cycle's posting (POSTTRAN), interest (INTCALC), and combine (COMBTRAN) jobs, so balances and transactions are complete.

What it produces

- `AWS.M2.CARDDEMO.STATEMNT.PS` - text statements, one per card.
- `AWS.M2.CARDDEMO.STATEMNT.HTML` - the same statements as HTML.
- `TRXFL.SEQ` and `TRXFL.VSAM.KSDS` - work files, rebuilt each run.

Notes for modernization

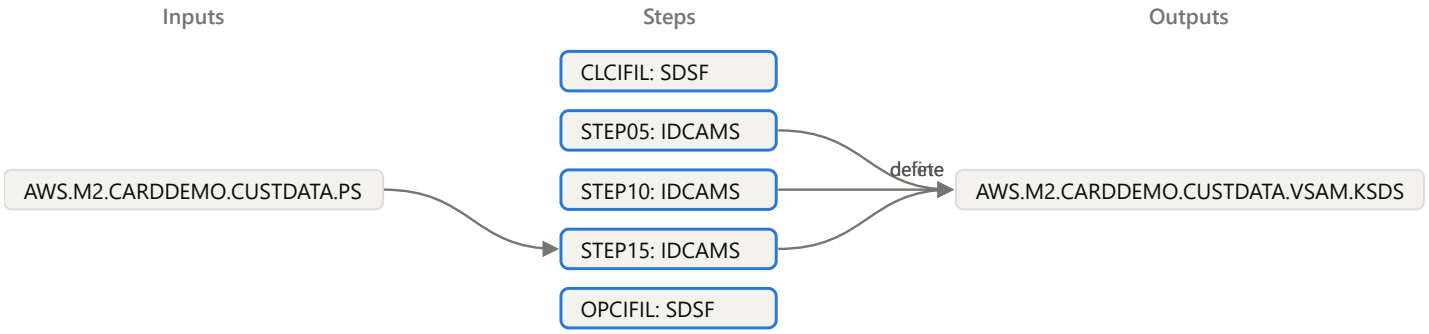
- **Only works for demo-sized data:** the statement program loads every transaction into a table of 51 cards x 10 transactions per card with no overflow check, so a realistic number of cards or transactions corrupts memory rather than failing cleanly.
- **No statement period:** every transaction ever posted to the master appears on the statement; there's no cycle start and end date. Real statements need one - confirm the intended cycle with the business.
- **One statement per card,** not per account: a customer with two cards on one account gets two statements, each showing the full account balance.
- The re-keying steps (delete, define, sort, load) exist only to read transactions in card order; with a database, a query ordered by card replaces steps 1-3.
- The statement program uses z/OS-specific control-block access and ALTER/GO TO (deliberately, to exercise modernization tools); both are removed in a rewrite. Statement layout and HTML generation map naturally to a templating or document-generation tool.

Job CUSTFILE

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	5	2	2
jobs	steps	system programs	datasets

48
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job CUSTFILE

Step 1 - CLCIFIL: runs SDSF (IBM utility)

- Comment: Close files in CICS region

Step 2 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE CUSTOMER VSAM FILE IF ONE ALREADY EXISTS
- Deletes: AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS
- Control statements: `DELETE AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS CLUSTER IF MAXCC LE 08 THEN SET MAXCC = 0`

Step 3 - STEP10: runs IDCAMS (IBM utility)

- Comment: DEFINE CUSTOMER VSAM FILE
- Defines: AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS
- Control statements: `DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS) CYLINDERS(1 5) VOLUMES(AWSHJ1 ) KEYS(9 0) RECORDSIZE(500 500) SHAREOPTIONS(2 3) ERASE`

Step 4 - STEP15: runs IDCAMS (IBM utility)

- Comment: COPY DATA FROM FLAT FILE TO VSAM FILE
- Copies AWS.M2.CARDDEMO.CUSTDATA.PS into AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS
- Reads: AWS.M2.CARDDEMO.CUSTDATA.PS (DD CUSTDATA)
- Writes: AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS (DD CUSTVSAM)
- Control statements: `REPRO INFILE(CUSTDATA) OUTFILE(CUSTVSAM)`

Step 5 - OPCIFIL: runs SDSF (IBM utility)

- Comment: Open files in CICS region

Inputs from outside the job: AWS.M2.CARDDEMO.CUSTDATA.PS **Outputs it leaves behind:** AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS

Job DALYREJS

Retire suggested - Only IBM utilities that define and delete data sets - housekeeping a database-backed platform doesn't need.

1	1	1	1
jobs	steps	system programs	datasets

10
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job DALYREJS

Step 1 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE TRANSACATION MASTER VSAM FILE IF ONE ALREADY EXISTS
- Defines: AWS.M2.CARDDEMO.DALYREJS
- Control statements: `DEFINE GENERATIONDATAGROUP (NAME(AWS.M2.CARDDEMO.DALYREJS) LIMIT(5) SCRATCH )`

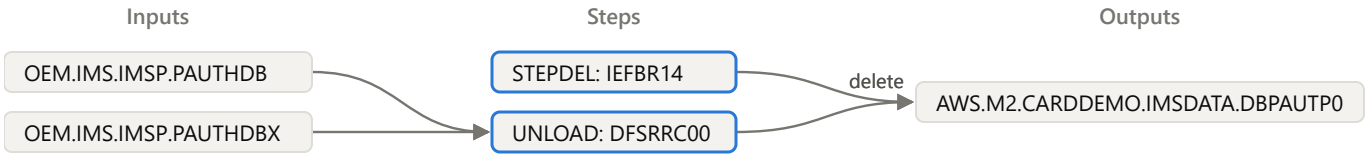
Inputs from outside the job: none **Outputs it leaves behind:** none

Job DBPAUTP0

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	2	2	3
jobs	steps	system programs	datasets

32
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job DBPAUTP0

Step 1 - STEPDEL: runs IEFBR14 (IBM utility)

- Comment: DELETE OUTPUT DATASET
- Deletes: AWS.M2.CARDDEMO.IMSDATA.DBPAUTP0 (DD SYSUT1)

Step 2 - UNLOAD: runs DFSRRC00

- Comment: DOWNLOAD DBD DBPAUTP0
- Deletes: AWS.M2.CARDDEMO.IMSDATA.DBPAUTP0 (DD DFSURGU1)
- Uses: OEM.IMS.IMSP.PAUTHDB (DD DDPAUTP0), OEM.IMS.IMSP.PAUTHDBX (DD DDPAUTX0)

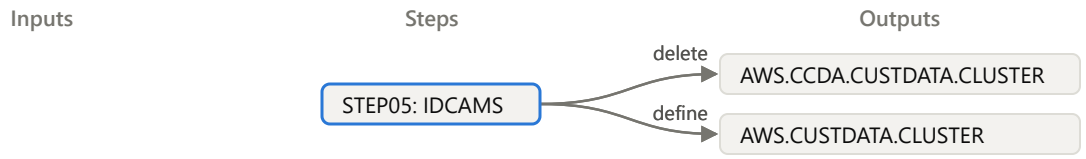
Inputs from outside the job: none **Outputs it leaves behind:** none

Job DEFCUST

Retire suggested - Only IBM utilities that define and delete data sets - housekeeping a database-backed platform doesn't need.

1	1	1	2
jobs	steps	system programs	datasets

24
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job DEFCUST

Step 1 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE CUSTOMER VSAM FILE IF ONE ALREADY EXISTS
- Defines: AWS.CUSTDATA.CLUSTER
- Deletes: AWS.CCDA.CUSTDATA.CLUSTER
- Control statements: `DELETE AWS.CCDA.CUSTDATA.CLUSTER CLUSTER`

Step 2 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE CUSTOMER VSAM FILE IF ONE ALREADY EXISTS
- Defines: AWS.CUSTDATA.CLUSTER
- Deletes: AWS.CCDA.CUSTDATA.CLUSTER
- Control statements: `DEFINE CLUSTER (NAME(AWS.CUSTDATA.CLUSTER) CYLINDERS(1 5) KEYS(10 0) RECORDSIZE(500 500) SHAREOPTIONS(1 4) ERASE INDEXED )`

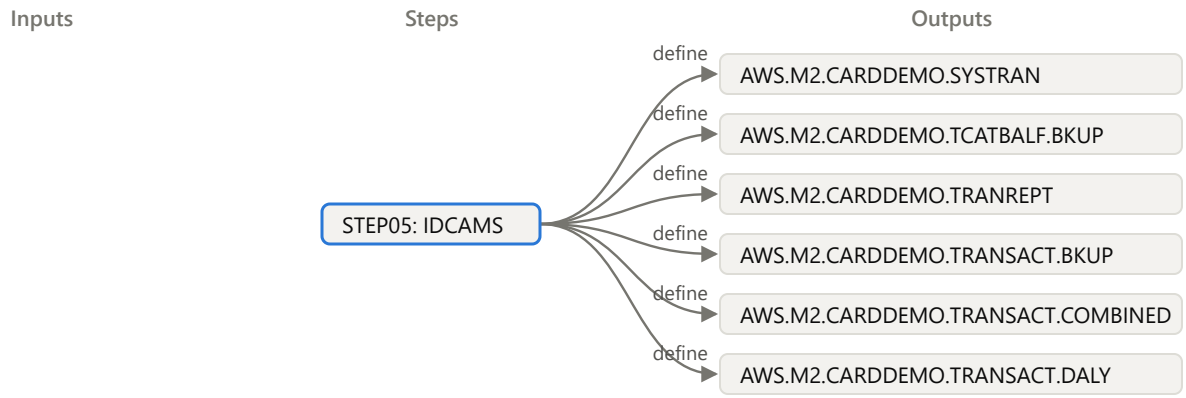
Inputs from outside the job: none **Outputs it leaves behind:** none

Job DEFGDGB

Retire suggested - Only IBM utilities that define and delete data sets - housekeeping a database-backed platform doesn't need.

1	1	1	6
jobs	steps	system programs	datasets

41
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job DEFGDGB

Step 1 - STEP05: runs IDCAMS (IBM utility)

- Comment: DEFINE GDG BASES NEEDED BY CARDDEMO PROJECT
- Defines: AWS.M2.CARDDEMO.SYSTRAN, AWS.M2.CARDDEMO.TCATBALF.BKUP, AWS.M2.CARDDEMO.TRANREPT, AWS.M2.CARDDEMO.TRANSACT.BKUP, AWS.M2.CARDDEMO.TRANSACT.COMBINED, AWS.M2.CARDDEMO.TRANSACT.DALY
- Control statements: `DEFINE GENERATIONDATAGROUP (NAME(AWS.M2.CARDDEMO.TRANSACT.BKUP) LIMIT(5) SCRATCH ) IF LASTCC=12 THEN SET MAXCC=0 DEFINE GENERATIONDATAGROUP (NAME(AWS.M2.CARDDEMO.TRANSACT.DALY)`

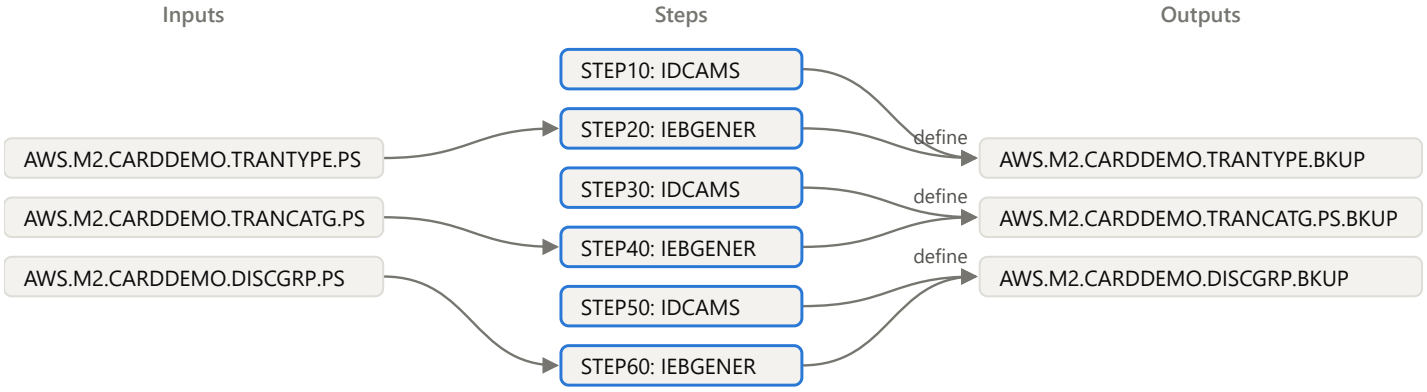
Inputs from outside the job: none **Outputs it leaves behind:** none

Job DEFGDGD

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	6	2	6
jobs	steps	system programs	datasets

52
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job DEFGDGD

Step 1 - STEP10: runs IDCAMS (IBM utility)

- Comment: RESTART=STEP30 / This jcl will create GDGs and load first generation for / Transaction reference data / Define GDG for Transaction Type
- Defines: AWS.M2.CARDDEMO.TRANTYPE.BKUP
- Control statements: `DEFINE GENERATIONDATAGROUP (NAME(AWS.M2.CARDDEMO.TRANTYPE.BKUP) LIMIT(5) SCRATCH )`

Step 2 - STEP20: runs IEBGENER (IBM utility)

- Comment: Create the first generation of GDG Transaction Type
- Copies AWS.M2.CARDDEMO.TRANTYPE.PS into AWS.M2.CARDDEMO.TRANTYPE.BKUP
- Reads: AWS.M2.CARDDEMO.TRANTYPE.PS (DD SYSUT1)
- Creates: AWS.M2.CARDDEMO.TRANTYPE.BKUP (DD SYSUT2)

Step 3 - STEP30: runs IDCAMS (IBM utility)

- Comment: Transaction Category type
- Defines: AWS.M2.CARDDEMO.TRANCATG.PS.BKUP
- Control statements: `DEFINE GENERATIONDATAGROUP (NAME(AWS.M2.CARDDEMO.TRANCATG.PS.BKUP) LIMIT(5) SCRATCH )`

Step 4 - STEP40: runs IEBGENER (IBM utility)

- Comment: Create the first generation of GDG Transaction Category Type
- Copies AWS.M2.CARDDEMO.TRANCATG.PS into AWS.M2.CARDDEMO.TRANCATG.PS.BKUP
- Reads: AWS.M2.CARDDEMO.TRANCATG.PS (DD SYSUT1)

- Creates: AWS.M2.CARDDEMO.TRANCATG.PS.BKUP (DD SYSUT2)

Step 5 - STEP50: runs IDCAMS (IBM utility)

- Comment: Disclosure Group
- Defines: AWS.M2.CARDDEMO.DISCGRP.BKUP
- Control statements: `DEFINE GENERATIONDATAGROUP (NAME(AWS.M2.CARDDEMO.DISCGRP.BKUP) LIMIT(5) SCRATCH )`

Step 6 - STEP60: runs IEBGENER (IBM utility)

- Comment: Create the first generation of GDG disclosure group
- Copies AWS.M2.CARDDEMO.DISCGRP.PS into AWS.M2.CARDDEMO.DISCGRP.BKUP
- Reads: AWS.M2.CARDDEMO.DISCGRP.PS (DD SYSUT1)
- Creates: AWS.M2.CARDDEMO.DISCGRP.BKUP (DD SYSUT2)

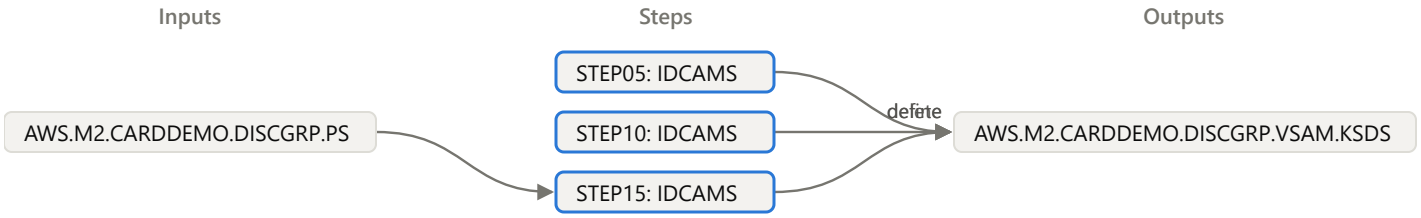
Inputs from outside the job: AWS.M2.CARDDEMO.DISCGRP.PS, AWS.M2.CARDDEMO.TRANCATG.PS, AWS.M2.CARDDEMO.TRANTYPE.PS **Outputs it leaves behind:** AWS.M2.CARDDEMO.DISCGRP.BKUP, AWS.M2.CARDDEMO.TRANCATG.PS.BKUP, AWS.M2.CARDDEMO.TRANTYPE.BKUP

Job DISCGRP

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	3	1	2
jobs	steps	system programs	datasets

36
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job DISCGRP

Step 1 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE DISCLOSURE GROUP VSAM FILE IF ONE ALREADY EXISTS
- Deletes: AWS.M2.CARDDEMO.DISCGRP.VSAM.KSDS
- Control statements: `DELETE AWS.M2.CARDDEMO.DISCGRP.VSAM.KSDS CLUSTER SET MAXCC = 0`

Step 2 - STEP10: runs IDCAMS (IBM utility)

- Comment: DEFINE DISCLOSURE GROUP VSAM FILE
- Defines: AWS.M2.CARDDEMO.DISCGRP.VSAM.KSDS
- Control statements: `DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.DISCGRP.VSAM.KSDS) CYLINDERS(1 5) VOLUMES(AWSHJ1 ) KEYS(16 0) RECORDSIZE(50 50) SHAREOPTIONS(2 3) ERASE`

Step 3 - STEP15: runs IDCAMS (IBM utility)

- Comment: COPY DATA FROM FLAT FILE TO VSAM FILE
- Copies AWS.M2.CARDDEMO.DISCGRP.PS into AWS.M2.CARDDEMO.DISCGRP.VSAM.KSDS
- Reads: AWS.M2.CARDDEMO.DISCGRP.PS (DD DISCGRP)
- Writes: AWS.M2.CARDDEMO.DISCGRP.VSAM.KSDS (DD DISCVSAM)
- Control statements: `REPRO INFILE(DISCGRP) OUTFILE(DISCVSAM)`

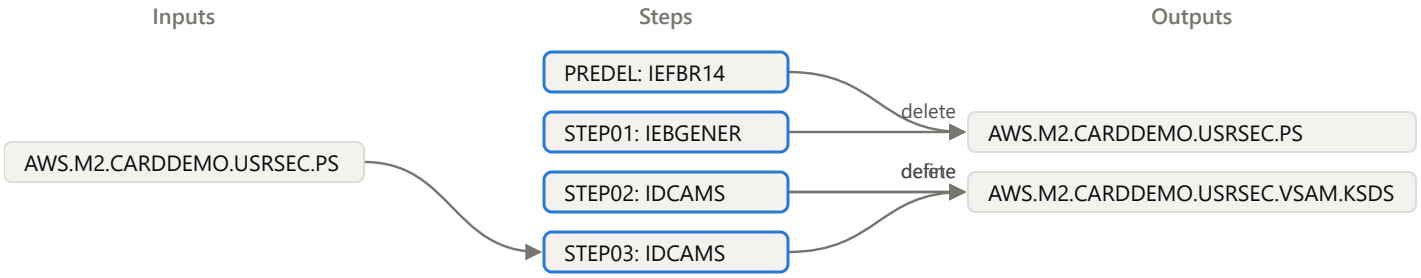
Inputs from outside the job: AWS.M2.CARDDEMO.DISCGRP.PS **Outputs it leaves behind:** AWS.M2.CARDDEMO.DISCGRP.VSAM.KSDS

Job DUSRSECJ

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	4	3	2
jobs	steps	system programs	datasets

48
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job DUSRSECJ

Step 1 - PREDEL: runs IEFBR14 (IBM utility)

- Comment: PRE DELETE STEP
- Deletes: AWS.M2.CARDDEMO.USRSEC.PS (DD DD01)

Step 2 - STEP01: runs IEBGENER (IBM utility)

- Comment: CREATE USER SECURITY FILE (PS) FROM IN-STREAM DATA
- Copies (in-stream SYSUT1) into AWS.M2.CARDDEMO.USRSEC.PS
- Creates: AWS.M2.CARDDEMO.USRSEC.PS (DD SYSUT2)

Step 3 - STEP02: runs IDCAMS (IBM utility)

- Comment: DEFINE VSAM FILE FOR USER SECURITY
- Defines: AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS
- Deletes: AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS
- Control statements: `DELETE AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS SET MAXCC = 0 DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS) KEYS(8,0) RECORDSIZE(80,80) REUSE INDEXED TRACKS(45,15)`

Step 4 - STEP03: runs IDCAMS (IBM utility)

- Comment: COPY USER SECURITY DATA FROM PS TO VSAM FILE
- Copies AWS.M2.CARDDEMO.USRSEC.PS into AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS
- Reads: AWS.M2.CARDDEMO.USRSEC.PS (DD IN)
- Writes: AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS (DD OUT)
- Control statements: `REPRO INFILE(IN) OUTFILE(OUT)`

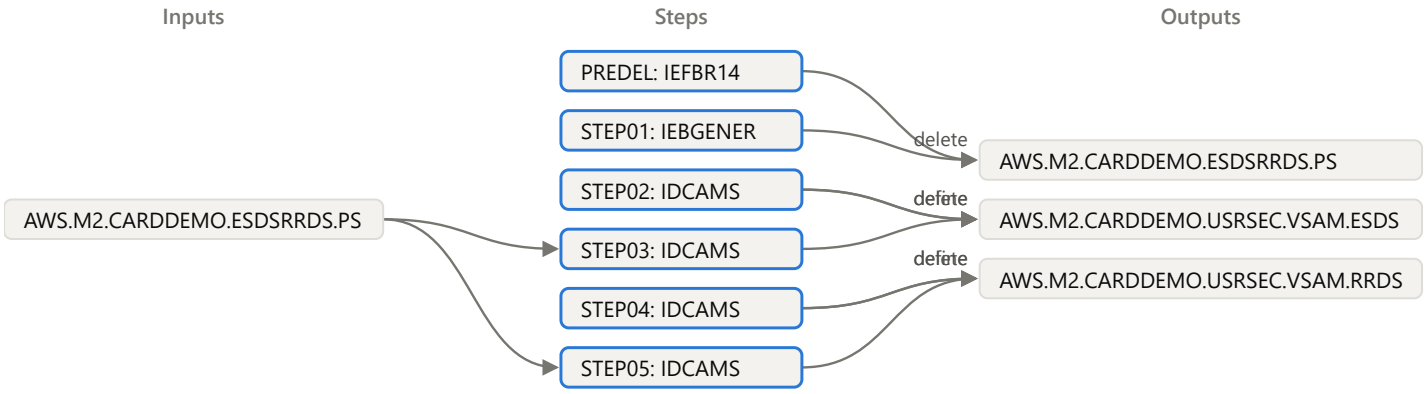
Inputs from outside the job: none **Outputs it leaves behind:** AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS

Job ESDSRRDS

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	6	3	3
jobs	steps	system programs	datasets

68
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job ESDSRRDS

Step 1 - PREDEL: runs IEFBR14 (IBM utility)

- Comment: PRE DELETE STEP
- Deletes: AWS.M2.CARDDEMO.ESDSRRDS.PS (DD DD01)

Step 2 - STEP01: runs IEBGENER (IBM utility)

- Comment: CREATE USER SECURITY FILE (PS) FROM IN-STREAM DATA
- Copies (in-stream SYSUT1) into AWS.M2.CARDDEMO.ESDSRRDS.PS
- Creates: AWS.M2.CARDDEMO.ESDSRRDS.PS (DD SYSUT2)

Step 3 - STEP02: runs IDCAMS (IBM utility)

- Comment: DEFINE VSAM FILE FOR USER SECURITY (ESDS)
- Defines: AWS.M2.CARDDEMO.USRSEC.VSAM.ESDS
- Deletes: AWS.M2.CARDDEMO.USRSEC.VSAM.ESDS
- Control statements: `DELETE AWS.M2.CARDDEMO.USRSEC.VSAM.ESDS SET MAXCC = 0 DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.USRSEC.VSAM.ESDS) RECORDSIZE(80,80) REUSE NONINDEXED TRACKS(45,15) FREESPACE(10,15)`

Step 4 - STEP03: runs IDCAMS (IBM utility)

- Comment: COPY USER SECURITY DATA FROM PS TO VSAM FILE(ESDS)
- Copies AWS.M2.CARDDEMO.ESDSRRDS.PS into AWS.M2.CARDDEMO.USRSEC.VSAM.ESDS
- Reads: AWS.M2.CARDDEMO.ESDSRRDS.PS (DD IN)
- Writes: AWS.M2.CARDDEMO.USRSEC.VSAM.ESDS (DD OUT)

- Control statements: `REPRO INFILE(IN) OUTFILE(OUT)`

Step 5 - STEP04: runs IDCAMS (IBM utility)

- Comment: DEFINE VSAM FILE FOR USER SECURITY (RRDS)
- Defines: AWS.M2.CARDDEMO.USRSEC.VSAM.RRDS
- Deletes: AWS.M2.CARDDEMO.USRSEC.VSAM.RRDS
- Control statements: `DELETE AWS.M2.CARDDEMO.USRSEC.VSAM.RRDS SET MAXCC = 0 DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.USRSEC.VSAM.RRDS) RECORDSIZE(80,80) REUSE NUMBERED TRACKS(45,15) FREESPACE(10,15)`

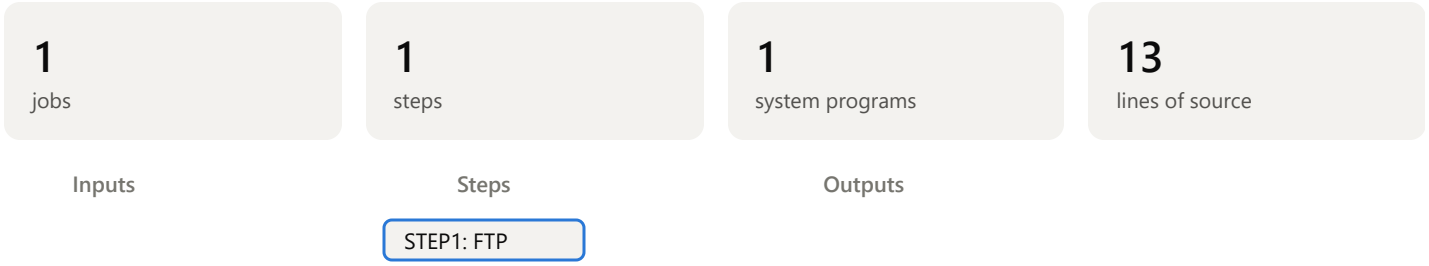
Step 6 - STEP05: runs IDCAMS (IBM utility)

- Comment: COPY USER SECURITY DATA FROM PS TO VSAM FILE(ESDS)
- Copies AWS.M2.CARDDEMO.ESDSRRDS.PS into AWS.M2.CARDDEMO.USRSEC.VSAM.RRDS
- Reads: AWS.M2.CARDDEMO.ESDSRRDS.PS (DD IN)
- Writes: AWS.M2.CARDDEMO.USRSEC.VSAM.RRDS (DD OUT)
- Control statements: `REPRO INFILE(IN) OUTFILE(OUT)`

Inputs from outside the job: none **Outputs it leaves behind:** AWS.M2.CARDDEMO.USRSEC.VSAM.ESDS, AWS.M2.CARDDEMO.USRSEC.VSAM.RRDS

Job FTPJCL

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job FTPJCL

Step 1 - STEP1: runs FTP (IBM utility)

- Comment: FTP JOB TO RECEIVE A FILE / MY.FTP.SERVER.COM IS THE FTP SERVER NAME(SUBJECT TO CHANGE FOR SITE) / USER ID - EITHER SCHEDULER ID OR TSO USER ID / PASSWORD OF THE USER / FOLDER TO FTP THE MAINFRAME FILE / PUT COMMAND FOR SENDING MAINFRAME FILE TO TXT / CHANGE THE PUT COMMAND TO GET TO RECEIVE FILE INTO MAINFRAME

Inputs from outside the job: none **Outputs it leaves behind:** none

Job INTCALC

Refactor suggested - 1 program(s), 622 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	10	622
copybooks	datasets	record layouts	lines of source



What it does

INTCALC is the monthly interest run. For every account it charges interest on the account's balance in each transaction category, at the annual rate set for the account's disclosure group (or the default group), and writes each charge as a system-generated interest transaction. It adds the account's total interest to its current balance and resets the account's current-cycle credit and debit totals, closing the billing cycle. The interest transactions go to a new file; a later job (COMBTRAN) loads them into the transaction master.

Step by step

1. **Calculate interest (STEP15, CBACT04C, with the run date as its PARM).** Reads the category balances in account order. For each account it reads the account and a card number from the cross-reference; for each of the account's categories it looks up the annual rate for the account's group, transaction type, and category (falling back to the DEFAULT group) and, if the rate isn't zero, computes a month's interest (balance x rate / 1200) and writes an interest transaction - type 01, category 05, source "System", ID made of the run date and a sequence number. When it moves on to the next account, it adds the interest total to the previous account's current balance and sets its cycle credit and debit totals to zero.

What it needs

- The category balances built up by daily posting (POSTTRAN, `TCATBALF.VSAM.KSDS`), the account master, the card cross-reference, and the disclosure group rates (`DISCGRP.VSAM.KSDS`), including DEFAULT group rates.
- The run date, coded in the JCL as `PARM='2022071800'` - it has to be changed for every run, or every month's interest transactions get IDs starting with the same date.
- To run after the month's posting is complete.

What it produces

- `AWS.M2.CARDDEMO.SYSTRAN(+1)`, a new generation of interest transactions, which COMBTRAN sorts together with the transaction master's backup and loads into the master.

- Updated account balances (interest added) and reset cycle totals - seen by the next statements (CREASTMT) and the online account screens.

Notes for modernization

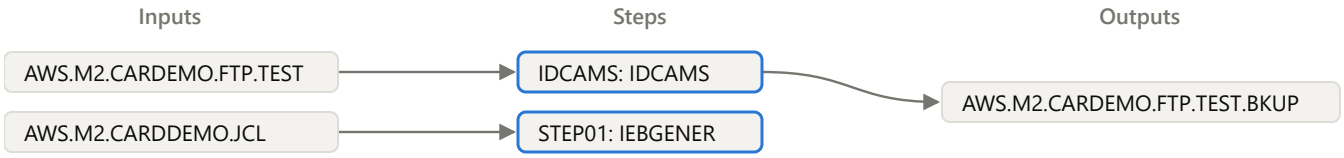
- **The last account in the file is never updated:** its interest transactions are written but the interest isn't added to its balance and its cycle isn't reset (a loop branch the program can't reach) - the transactions and balances disagree for that account every month.
- **Interest is charged on running totals that nothing resets:** the category balances are only ever added to by posting; neither this job nor the posting job clears them after interest is charged, so each month's interest is calculated on all activity to date, not the month's balance. Confirm with the business how the balance subject to interest should be defined - this is a likely defect, not just a modernization detail.
- **Hard-coded run date** in the JCL; on a modern scheduler this is the business date passed in automatically.
- Fees are a placeholder in the program (not implemented).
- Interest posting is split across three jobs (INTCALC writes, COMBTRAN merges and reloads the master); a modern design would post the interest directly, in the same database transaction as the balance update.

Job INTRDRJ1

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	2	2	3
jobs	steps	system programs	datasets

14
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job INTRDRJ1

Step 1 - IDCAMS: runs IDCAMS (IBM utility)

- Comment: THIS INTRDR JOB WILL TRIGGER ANOTHER JCL
- Copies AWS.M2.CARDEMO.FTP.TEST into AWS.M2.CARDEMO.FTP.TEST.BKUP
- Reads: AWS.M2.CARDEMO.FTP.TEST (DD IN)
- Writes: AWS.M2.CARDEMO.FTP.TEST.BKUP (DD OUT)
- Control statements: `REPRO IFILE(IN) OFILE(OUT)`

Step 2 - STEP01: runs IEBGENER (IBM utility)

- Reads: AWS.M2.CARDDEMO.JCL (DD SYSUT1)

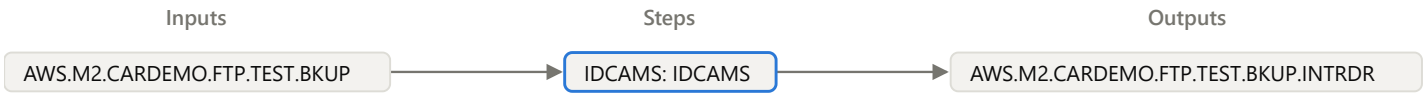
Inputs from outside the job: AWS.M2.CARDDEMO.JCL, AWS.M2.CARDEMO.FTP.TEST **Outputs it leaves behind:** AWS.M2.CARDEMO.FTP.TEST.BKUP

Job INTRDRJ2

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	1	1	2
jobs	steps	system programs	datasets

9
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job INTRDRJ2

Step 1 - IDCAMS: runs IDCAMS (IBM utility)

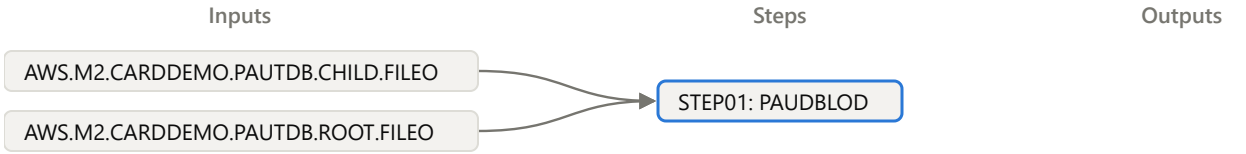
- Comment: THIS JOB IS TO CREATE PHYSICAL VSAM FILE FOR IMS DEMODB / AND INDEX DEMODX
- Copies AWS.M2.CARDEMO.FTP.TEST.BKUP into AWS.M2.CARDEMO.FTP.TEST.BKUP.INTRDR
- Reads: AWS.M2.CARDEMO.FTP.TEST.BKUP (DD IN)
- Writes: AWS.M2.CARDEMO.FTP.TEST.BKUP.INTRDR (DD OUT)
- Control statements: `REPRO IFILE(IN) OFILE(OUT)`

Inputs from outside the job: AWS.M2.CARDEMO.FTP.TEST.BKUP **Outputs it leaves behind:** AWS.M2.CARDEMO.FTP.TEST.BKUP.INTRDR

Job LOADPADB

Rehost suggested - Uses IMS databases; rehost first and plan the data migration separately.

1 jobs	1 steps	1 programs	1 system programs
4 copybooks	2 datasets	2 IMS segments	2 record layouts
340 lines of source			



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job LOADPADB

Step 1 - STEP01: runs PAUDBLOD, through DFSRRC00

- Comment: YLIBS JCLLIB ORDER=VIPINGP.CNTL.PROCLIB / EXECUTE IMS PROGRAM
- Reads: AWS.M2.CARDDEMO.PAUTDB.CHILD.FILEO (DD INFILE2), AWS.M2.CARDDEMO.PAUTDB.ROOT.FILEO (DD INFILE1)
- Adds records to IMS segment PAUTDTL1
- Adds records to and reads IMS segment PAUTSUM0

Inputs from outside the job: AWS.M2.CARDDEMO.PAUTDB.CHILD.FILEO, AWS.M2.CARDDEMO.PAUTDB.ROOT.FILEO **Outputs it leaves behind:** none

Job MNTTRDB2

Refactor suggested - 1 program(s), 200 lines: small enough to rewrite as a batch service, query, or report on the target platform.

1	1	1	1
jobs	steps	programs	DCLGENs
1	**1**	**2**	**200**
datasets	Db2 tables	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job MNTTRDB2

From the job's comments: USE THIS JOB TO MAINTAIN DB2 TRANSACTION TABLE, INSERTING, / UPDATING OR REMOVING RECORDS IN BATCH. / INPFILE - THE INPUT FILE TO USE FOR UPDATES. THE ALLOWED VALUES / ARE: / COLUMN 1 - A - ADD / D - DELETE / U - UPDATE / COMMENT

Step 1 - STEP1: runs COBTUPDT, through IKJEFT01

- Comment: USE THIS JOB TO MAINTAIN DB2 TRANSACTION TABLE, INSERTING, / UPDATING OR REMOVING RECORDS IN BATCH. / INPFILE - THE INPUT FILE TO USE FOR UPDATES. THE ALLOWED VALUES / ARE: / COLUMN 1 - A - ADD / D - DELETE / U - UPDATE / COMMENT / COLUMNS 2-3 - TRANSACTION TYPE. NUMERIC VALUE / COLUMNS 4-53 - TRANSACTION DESCRIPTION
- COBTUPDT's comments: Program: COBTUPDT.CBL / Layer: Business logic / Function: Update Transaction type based on user input
- Reads: INPFILE (DD INPFILE)
- Adds records to and updates and deletes from Db2 table TRANSACTION_TYPE

Inputs from outside the job: INPFILE **Outputs it leaves behind:** none

Job OPENFIL

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	1	1	12
jobs	steps	system programs	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job OPENFIL

Step 1 - OPCIFIL: runs SDSF (IBM utility)

- Comment: Open files in CICS region

Inputs from outside the job: none **Outputs it leaves behind:** none

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.

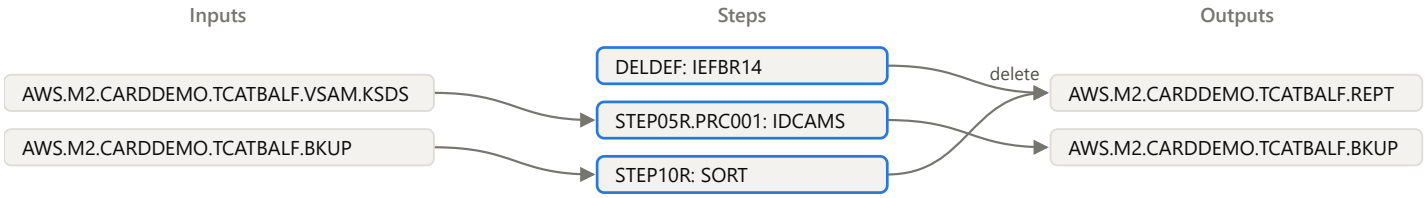
Notes for modernization

- **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.

Job PRTCATBL

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1 jobs	1 PROCs	3 steps	3 system programs
1 control cards	3 datasets	63 lines of source	



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job PRTCATBL

Step 1 - DELDEF: runs IEFBR14 (IBM utility)

- Deletes: AWS.M2.CARDDEMO.TCATBALF.REPT (DD THEFILE)

Step 2 - STEP05R.PRC001 (in PROC REPROC): runs IDCAMS (IBM utility)

- Comment: Unload the processed transaction category balance file / REPRO UTILITY TO LOAD OR UNLOAD VSAM FILE
- Copies AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS into AWS.M2.CARDDEMO.TCATBALF.BKUP
- Reads: AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS (DD FILEIN)
- Creates: AWS.M2.CARDDEMO.TCATBALF.BKUP (DD FILEOUT)
- Control statements: `REPRO INFILE(FILEIN) OUTFILE(FILEOUT)`
- Reads control card REPROCT (DD SYSIN)

Step 3 - STEP10R: runs SORT (IBM utility)

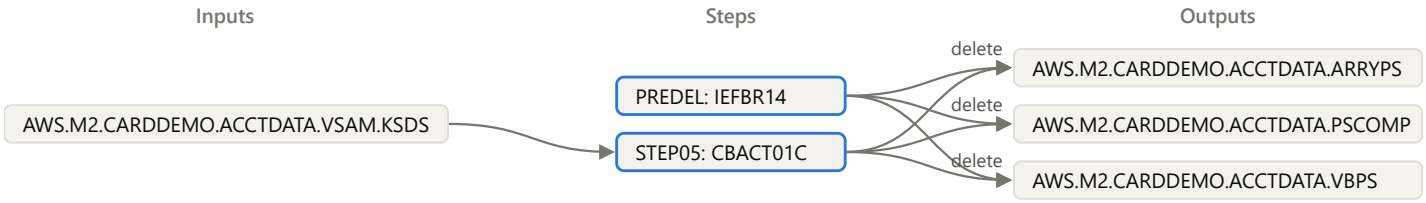
- Comment: Filter the TCATBALFions for a the parm date and sort by card num
- Sorts AWS.M2.CARDDEMO.TCATBALF.BKUP into AWS.M2.CARDDEMO.TCATBALF.REPT
- Reads: AWS.M2.CARDDEMO.TCATBALF.BKUP (DD SORTIN)
- Creates: AWS.M2.CARDDEMO.TCATBALF.REPT (DD SORTOUT)
- Sort keys: TRANCAT-ACCT-ID (ascending, positions 1-11), TRANCAT-TYPE-CD (ascending, positions 12-13), TRANCAT-CD (ascending, positions 14-17).
- Reformats the records (INREC/OUTREC) - see the control statements.
- Control statements: `SORT FIELDS=(TRANCAT-ACCT-ID,A,TRANCAT-TYPE-CD,A,TRANCAT-CD,A) OUTREC FIELDS=(TRANCAT-ACCT-ID,X, TRANCAT-TYPE-CD,X, TRANCAT-CD,X, TRAN-CAT-BAL,EDIT=(TTTTTTTTT.TT),9X)`

Inputs from outside the job: AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS **Outputs it leaves behind:** AWS.M2.CARDDEMO.TCATBALF.REPT

Job READACCT

Refactor suggested - 2 program(s), 497 lines: small enough to rewrite as a batch service, query, or report on the target platform.

1	2	2	2
jobs	steps	programs	system programs
2	**4**	**6**	**497**
copybooks	datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job READACCT

Step 1 - PREDEL: runs IEFBR14 (IBM utility)

- Comment: PRE DELETE STEP
- Deletes: AWS.M2.CARDDEMO.ACCTDATA.ARRYPS (DD DD02), AWS.M2.CARDDEMO.ACCTDATA.PSCOMP (DD DD01), AWS.M2.CARDDEMO.ACCTDATA.VBPS (DD DD03)

Step 2 - STEP05: runs CBACT01C

- Comment: RUN THE PROGRAM THAT READS THE ACCOUNT MASTER VSAM FILE
- CBACT01C's comments: PROGRAM : CBACT01C.CBL / Application : CardDemo / Type : BATCH COBOL Program / FUNCTION : READ THE ACCOUNT FILE AND WRITE INTO FILES.
- Reads: AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS (DD ACCTFILE)
- Creates: AWS.M2.CARDDEMO.ACCTDATA.ARRYPS (DD ARRYFILE), AWS.M2.CARDDEMO.ACCTDATA.PSCOMP (DD OUTFILE), AWS.M2.CARDDEMO.ACCTDATA.VBPS (DD VBRCFILE)

Inputs from outside the job: AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS **Outputs it leaves behind:** AWS.M2.CARDDEMO.ACCTDATA.ARRYPS, AWS.M2.CARDDEMO.ACCTDATA.PSCOMP, AWS.M2.CARDDEMO.ACCTDATA.VBPS

Job READCARD

Refactor suggested - 1 program(s), 146 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
1	**1**	**2**	**146**
copybooks	datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job READCARD

Step 1 - STEP05: runs CBACT02C

- Comment: RUN THE PROGRAM THAT READS THE CARD MASTER VSAM FILE
- CBACT02C's comments: Program : CBACT02C.CBL / Application : CardDemo / Type : BATCH COBOL Program / Function : Read and print card data file.
- Reads: AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS (DD CARDFILE)

Inputs from outside the job: AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS **Outputs it leaves behind:** none

Job READCUST

Refactor suggested - 1 program(s), 159 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
1	**1**	**2**	**159**
copybooks	datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job READCUST

Step 1 - STEP05: runs CBCUS01C

- Comment: RUN THE PROGRAM THAT READS THE CUSTOMER MASTER VSAM FILE
- CBCUS01C's comments: Program : CBCUS01C.CBL / Application : CardDemo / Type : BATCH COBOL Program / Function : Read and print customer data file.
- Reads: AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS (DD CUSTFILE)

Inputs from outside the job: AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS **Outputs it leaves behind:** none

Job READXREF

Refactor suggested - 1 program(s), 144 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
1	**1**	**2**	**144**
copybooks	datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job READXREF

Step 1 - STEP05: runs CBACT03C

- Comment: RUN THE PROGRAM THAT READS THE XREF MASTER VSAM FILE
- CBACT03C's comments: Program : CBACT03C.CBL / Application : CardDemo / Type : BATCH COBOL Program / Function : Read and print account cross reference data file.
- Reads: AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS (DD XREFFILE)

Inputs from outside the job: AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS **Outputs it leaves behind:** none

Job REPTFILE

Retire suggested - Only IBM utilities that define and delete data sets - housekeeping a database-backed platform doesn't need.

1	1	1	1
jobs	steps	system programs	datasets

10
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job REPTFILE

Step 1 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE TRANSACATION MASTER VSAM FILE IF ONE ALREADY EXISTS
- Defines: AWS.M2.CARDDEMO.TRANREPT
- Control statements: `DEFINE GENERATIONDATAGROUP (NAME(AWS.M2.CARDDEMO.TRANREPT) LIMIT(10) )`

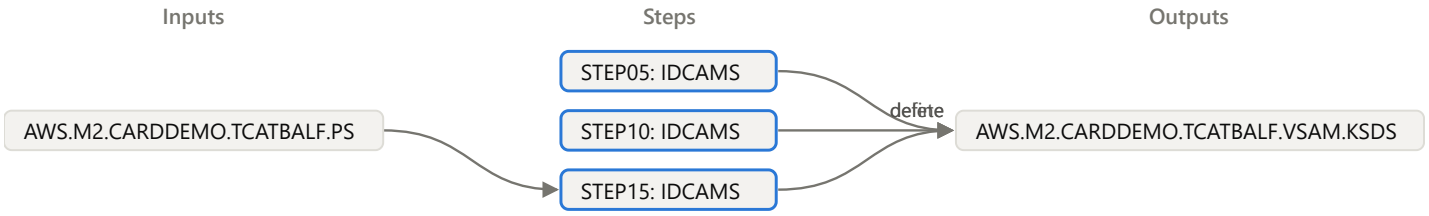
Inputs from outside the job: none **Outputs it leaves behind:** none

Job TCATBALF

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	3	1	2
jobs	steps	system programs	datasets

36
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job TCATBALF

Step 1 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE TRANSACTION CATEGORY BALANCE VSAM FILE IF ONE ALREADY EXISTS
- Deletes: AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS
- Control statements: `DELETE AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS CLUSTER SET MAXCC = 0`

Step 2 - STEP10: runs IDCAMS (IBM utility)

- Comment: DEFINE TRANSACTION CATEGORY BALANCE VSAM FILE
- Defines: AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS
- Control statements: `DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS) CYLINDERS(1 5) VOLUMES(AWSHJ1 ) KEYS(17 0) RECORDSIZE(50 50) SHAREOPTIONS(2 3) ERASE`

Step 3 - STEP15: runs IDCAMS (IBM utility)

- Comment: COPY DATA FROM FLAT FILE TO VSAM FILE
- Copies AWS.M2.CARDDEMO.TCATBALF.PS into AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS
- Reads: AWS.M2.CARDDEMO.TCATBALF.PS (DD TCATBAL)
- Writes: AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS (DD TCATBALV)
- Control statements: `REPRO INFILE(TCATBAL) OUTFILE(TCATBALV)`

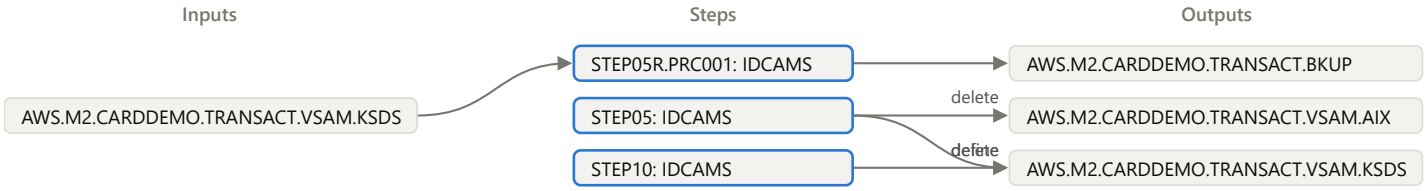
Inputs from outside the job: AWS.M2.CARDDEMO.TCATBALF.PS **Outputs it leaves behind:** AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS

Job TRANBKP

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	1	3	1
jobs	PROCs	steps	system programs

1	3	65
control cards	datasets	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job TRANBKP

Step 1 - STEP05R.PRC001 (in PROC REPROC): runs IDCAMS (IBM utility)

- Comment: Repro the processed transaction file / REPRO UTILITY TO LOAD OR UNLOAD VSAM FILE
- Copies AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS into AWS.M2.CARDDEMO.TRANSACT.BKUP
- Reads: AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS (DD FILEIN)
- Creates: AWS.M2.CARDDEMO.TRANSACT.BKUP (DD FILEOUT)
- Control statements: `REPRO INFILE(FILEIN) OUTFILE(FILEOUT)`
- Reads control card REPROCT (DD SYSIN)

Step 2 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE TRANSACATION MASTER VSAM FILE IF ONE ALREADY EXISTS
- Deletes: AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX, AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
- Control statements: `DELETE AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS CLUSTER IF MAXCC LE 08 THEN SET MAXCC = 0 DELETE AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX ALTERNATEINDEX IF MAXCC LE 08 THEN SET MAXCC = 0`

Step 3 - STEP10: runs IDCAMS (IBM utility)

- Comment: DEFINE TRANSACATION MASTER VSAM FILE
- Defines: AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
- Control statements: `DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS) CYLINDERS(1 5) VOLUMES(AWSHJ1 ) KEYS(16 0) RECORDSIZE(350 350) SHAREOPTIONS(2 3) ERASE`

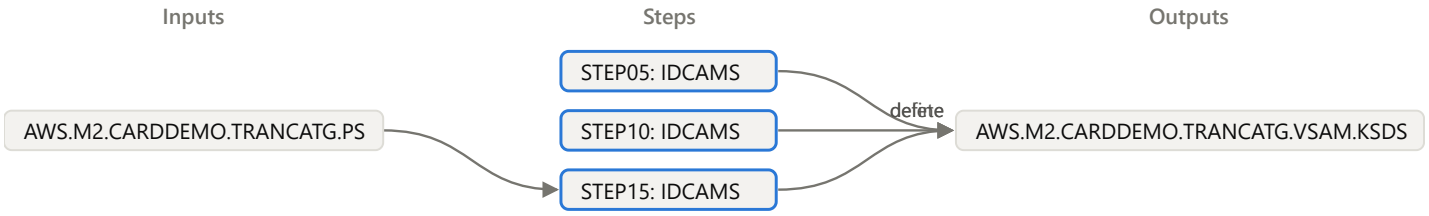
Inputs from outside the job: AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS **Outputs it leaves behind:** AWS.M2.CARDDEMO.TRANSACT.BKUP

Job TRANCATG

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	3	1	2
jobs	steps	system programs	datasets

36
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job TRANCATG

Step 1 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE TRANSACTION CATEGORY TYPE VSAM FILE IF ONE ALREADY EXISTS
- Deletes: AWS.M2.CARDDEMO.TRANCATG.VSAM.KSDS
- Control statements: `DELETE AWS.M2.CARDDEMO.TRANCATG.VSAM.KSDS CLUSTER SET MAXCC = 0`

Step 2 - STEP10: runs IDCAMS (IBM utility)

- Comment: DEFINE TRANSACTION CATEGORY TYPE VSAM FILE
- Defines: AWS.M2.CARDDEMO.TRANCATG.VSAM.KSDS
- Control statements: `DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.TRANCATG.VSAM.KSDS) CYLINDERS(1 5) VOLUMES(AWSHJ1 ) KEYS(6 0) RECORDSIZE(60 60) SHAREOPTIONS(2 3) ERASE`

Step 3 - STEP15: runs IDCAMS (IBM utility)

- Comment: COPY DATA FROM FLAT FILE TO VSAM FILE
- Copies AWS.M2.CARDDEMO.TRANCATG.PS into AWS.M2.CARDDEMO.TRANCATG.VSAM.KSDS
- Reads: AWS.M2.CARDDEMO.TRANCATG.PS (DD TRANCATG)
- Writes: AWS.M2.CARDDEMO.TRANCATG.VSAM.KSDS (DD TCATVSAM)
- Control statements: `REPRO INFILE(TRANCATG) OUTFILE(TCATVSAM)`

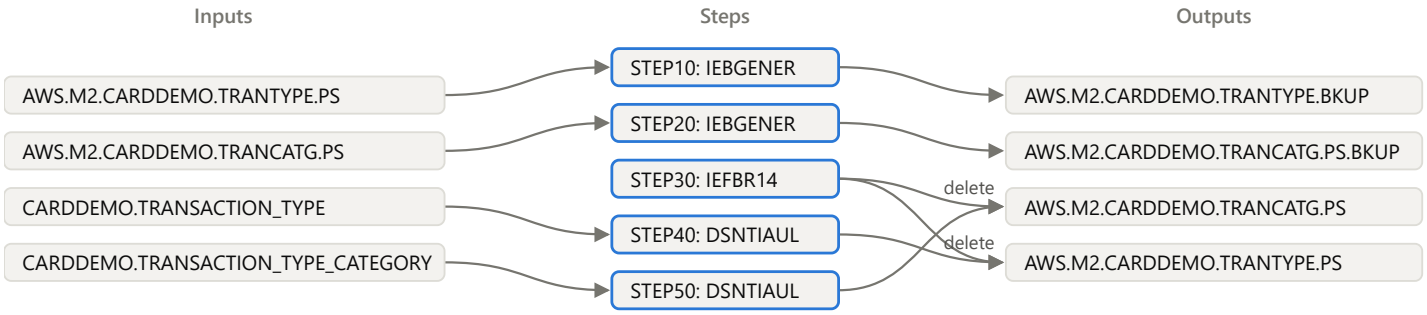
Inputs from outside the job: AWS.M2.CARDDEMO.TRANCATG.PS **Outputs it leaves behind:** AWS.M2.CARDDEMO.TRANCATG.VSAM.KSDS

Job TRANEXTR

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	5	3	4
jobs	steps	system programs	datasets

2	83
Db2 tables	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job TRANEXTR

Step 1 - STEP10: runs IEBGENER (IBM utility)

- Comment: This JCL will extract reference data for use in Transaction / Report generation. It runs once a day. So changes will reflect / in report only after the daily batch is run. / Set Parms for this compile: / Db2 Unload using DSNTIAUL utility / STEP 10 : BACKUP THE PREVIOUS VERSION OF TRANTYPE FILE TO GDG
- Copies AWS.M2.CARDDEMO.TRANTYPE.PS into AWS.M2.CARDDEMO.TRANTYPE.BKUP
- Reads: AWS.M2.CARDDEMO.TRANTYPE.PS (DD SYSUT1)
- Creates: AWS.M2.CARDDEMO.TRANTYPE.BKUP (DD SYSUT2)

Step 2 - STEP20: runs IEBGENER (IBM utility)

- Comment: STEP 20 : BACKUP THE PREVIOUS VERSION OF TRANCATG FILE TO GDG
- Copies AWS.M2.CARDDEMO.TRANCATG.PS into AWS.M2.CARDDEMO.TRANCATG.PS.BKUP
- Reads: AWS.M2.CARDDEMO.TRANCATG.PS (DD SYSUT1)
- Creates: AWS.M2.CARDDEMO.TRANCATG.PS.BKUP (DD SYSUT2)

Step 3 - STEP30: runs IEFBR14 (IBM utility)

- Comment: STEP 30 : DELETE FILES FROM PREVIOUS RUN
- Deletes: AWS.M2.CARDDEMO.TRANCATG.PS (DD DD02), AWS.M2.CARDDEMO.TRANTYPE.PS (DD DD01)

Step 4 - STEP40: runs DSNTIAUL (IBM utility), through IKJEFT01

- Comment: STEP 40 : EXTRACT DATA FROM TRANSACTION TYPE TABLE
- UNLOAD TABLE CARDDEMO.TRANSACTION_TYPE into AWS.M2.CARDDEMO.TRANTYPE.PS
- Reads: CARDDEMO.TRANSACTION_TYPE
- Creates: AWS.M2.CARDDEMO.TRANTYPE.PS (DD SYSREC00)

Step 5 - STEP50: runs DSNTIAUL (IBM utility), through IKJEFT01

- Comment: STEP 50 : EXTRACT DATA FROM TRANSACTION TYPE TABLE
- UNLOAD TABLE CARDDEMO.TRANSACTION_TYPE_CATEGORY into AWS.M2.CARDDEMO.TRANCATG.PS
- Reads: CARDDEMO.TRANSACTION_TYPE_CATEGORY
- Creates: AWS.M2.CARDDEMO.TRANCATG.PS (DD SYSREC00)

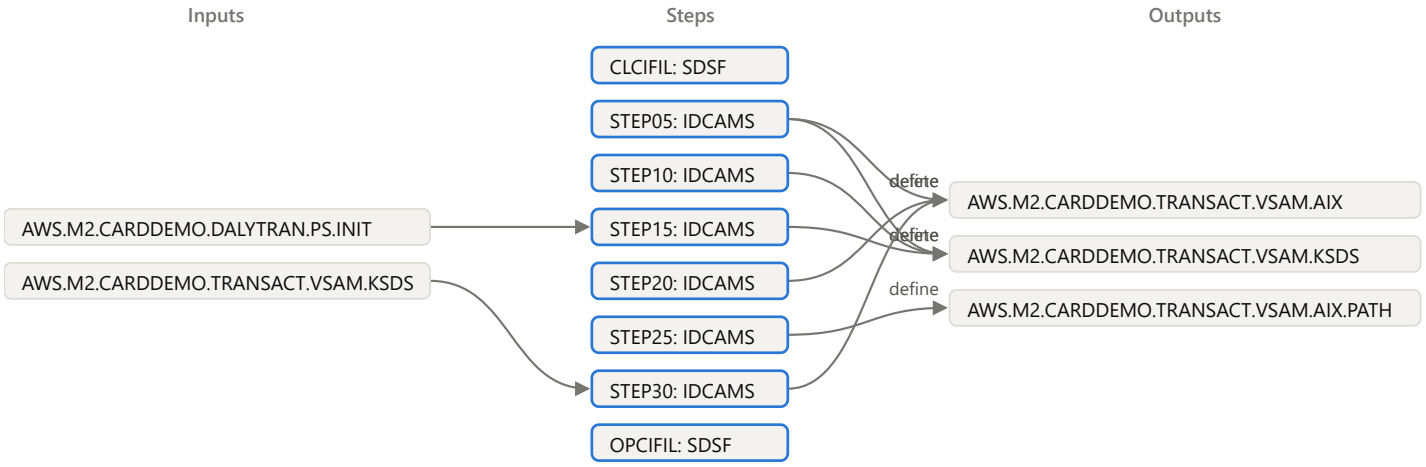
Inputs from outside the job: CARDDEMO.TRANSACTION_TYPE, CARDDEMO.TRANSACTION_TYPE_CATEGORY **Outputs it leaves behind:** AWS.M2.CARDDEMO.TRANCATG.PS, AWS.M2.CARDDEMO.TRANCATG.PS.BKUP, AWS.M2.CARDDEMO.TRANTYPE.BKUP, AWS.M2.CARDDEMO.TRANTYPE.PS

Job TRANFILE

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	8	2	4
jobs	steps	system programs	datasets

81
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job TRANFILE

Step 1 - CLCIFIL: runs SDSF (IBM utility)

- Comment: Close files in CICS region

Step 2 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE TRANSACATION MASTER VSAM FILE IF ONE ALREADY EXISTS
- Deletes: AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX, AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
- Control statements: `DELETE AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS CLUSTER IF MAXCC LE 08 THEN SET MAXCC = 0 DELETE AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX ALTERNATEINDEX IF MAXCC LE 08 THEN SET MAXCC = 0`

Step 3 - STEP10: runs IDCAMS (IBM utility)

- Comment: DEFINE TRANSACATION MASTER VSAM FILE
- Defines: AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
- Control statements: `DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS) CYLINDERS(1 5) VOLUMES(AWSHJ1 ) KEYS(16 0) RECORDSIZE(350 350) SHAREOPTIONS(2 3) ERASE`

Step 4 - STEP15: runs IDCAMS (IBM utility)

- Comment: COPY DATA FROM FLAT FILE TO VSAM FILE
- Copies AWS.M2.CARDDEMO.DALYTRAN.PS.INIT into AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
- Reads: AWS.M2.CARDDEMO.DALYTRAN.PS.INIT (DD TRANSACT)

- Writes: AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS (DD TRANVSAM)
- Control statements: `REPRO INFILE(TRANSACT) OUTFILE(TRANVSAM)`

Step 5 - STEP20: runs IDCAMS (IBM utility)

- Comment: CREATE ALTERNATE INDEX ON PROCESSED TIMESTAMP
- Defines: AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX
- Control statements: `DEFINE ALTERNATEINDEX (NAME(AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX)-
  RELATE(AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS) KEYS(26 304) NONUNIQUEKEY UPGRADE RECORDSIZE(350,350)
  VOLUMES(AWSHJ1) CYLINDERS(5,1))`

Step 6 - STEP25: runs IDCAMS (IBM utility)

- Comment: DEFINE PATH IS USED TO RELATE THE ALTERNATE INDEX TO BASE CLUSTER
- Defines: AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX.PATH
- Control statements: `DEFINE PATH (NAME(AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX.PATH)
  PATHENTRY(AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX))`

Step 7 - STEP30: runs IDCAMS (IBM utility)

- Comment: BUILD ALTERNATE INDEX CLUSTER
- BLDINDEX AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS into AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX
- Reads: AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
- Writes: AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX
- Control statements: `BLDINDEX INDATASET(AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS)
  OUTDATASET(AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX)`

Step 8 - OPCIFIL: runs SDSF (IBM utility)

- Comment: Opem files in CICS region

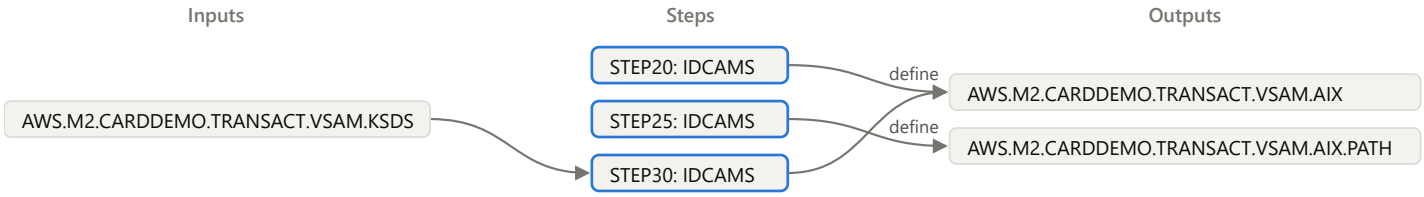
Inputs from outside the job: AWS.M2.CARDDEMO.DALYTRAN.PS.INIT **Outputs it leaves behind:** AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX

Job TRANIDX

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	3	1	3
jobs	steps	system programs	datasets

30
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job TRANIDX

Step 1 - STEP20: runs IDCAMS (IBM utility)

- Comment: CREATE ALTERNATE INDEX ON PROCESSED TIMESTAMP
- Defines: AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX
- Control statements: `DEFINE ALTERNATEINDEX (NAME(AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX)-RELATE(AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS) KEYS(26 304) NONUNIQUEKEY UPGRADE RECORDSIZE(350,350) VOLUMES(AWSHJ1) CYLINDERS(5,1))`

Step 2 - STEP25: runs IDCAMS (IBM utility)

- Comment: DEFINE PATH IS USED TO RELATE THE ALTERNATE INDEX TO BASE CLUSTER
- Defines: AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX.PATH
- Control statements: `DEFINE PATH (NAME(AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX.PATH) PATHENTRY(AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX))`

Step 3 - STEP30: runs IDCAMS (IBM utility)

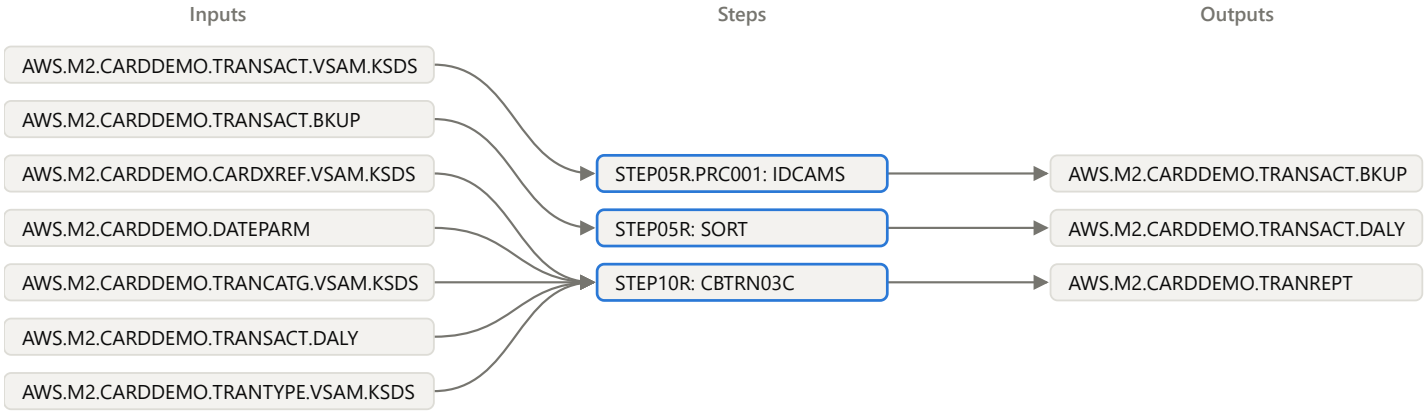
- Comment: BUILD ALTERNATE INDEX CLUSTER
- BLDINDEX AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS into AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX
- Reads: AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
- Writes: AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX
- Control statements: `BLDINDEX INDATASET(AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS) OUTDATASET(AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX)`

Inputs from outside the job: AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS **Outputs it leaves behind:** AWS.M2.CARDDEMO.TRANSACT.VSAM.AIX

Job TRANREPT

Refactor suggested - 1 program(s), 708 lines: small enough to rewrite as a batch service, query, or report on the target platform.

1 jobs	**1** PROCs	**3** steps	**1** programs
3 system programs	**5** copybooks	**1** control cards	**8** datasets
15 record layouts	**708** lines of source		



What it does

TRANREPT produces the Daily Transaction Report. It takes a backup copy of the transaction master file, keeps only the transactions processed within a date range, sorts them by card number, and prints a report listing each transaction with its account, type, category, source, and amount, with totals per card, per page, and overall. Each run creates new generations of the backup, the filtered extract, and the report, so earlier runs are kept.

Step by step

1. **Unload the transaction master (STEP05R, PROC REPROC, IDCAMS REPRO).** Copies the whole VSAM transaction master file (`TRANSACT.VSAM.KSDS`) to a new generation of a sequential backup (`TRANSACT.BKUP(+1)`).
2. **Select and sort (STEP05R, SORT).** From that backup, keeps only the transactions whose processing date (TRAN-PROC-DT, the date part of TRAN-PROC-TS) is between the start and end dates coded in the JCL - currently 2022-01-01 to 2022-07-06 - sorts them by card number (TRAN-CARD-NUM), and writes them to a new generation of the daily extract (`TRANSACT.DALY(+1)`).
3. **Print the report (STEP10R, CBTRN03C).** Reads the extract in card-number order, reads its date range from the date parameter file (`DATEPARM`), looks up each card's account ID and each transaction's type and category descriptions, and writes the report (`TRANREPT(+1)`): a "Daily Transaction Report" heading with the date range, one line per transaction, an "Account Total" when the card number changes, a "Page Total" every 20 lines, and a "Grand Total" at the end.

What it needs

- The transaction master file `AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS`, already updated by the posting job (POSTTRAN) - TRANREPT reports on processed transactions, so it should run after posting.
- Reference files: card cross-reference (`CARDXREF.VSAM.KSDS`), transaction types (`TRANTYPE.VSAM.KSDS`), and transaction categories (`TRANCATG.VSAM.KSDS`). A card, type, or category missing from them stops the job.
- The date range, in two places that must match: the SYMNAMES constants in the TRANREPT JCL (used by the SORT) and the first record of `AWS.M2.CARDDEMO.DATEPARM` (printed on the report and checked by the program).

What it produces

- `AWS.M2.CARDDEMO.TRANREPT(+1)` - the printed report (133-character lines), for operations or finance to review.
- `AWS.M2.CARDDEMO.TRANSACT.BKUP(+1)` - a full backup of the transaction master as of this run; other jobs (COMBTRAN) read this backup.
- `AWS.M2.CARDDEMO.TRANSACT.DALY(+1)` - the filtered, sorted extract the report was printed from.

Notes for modernization

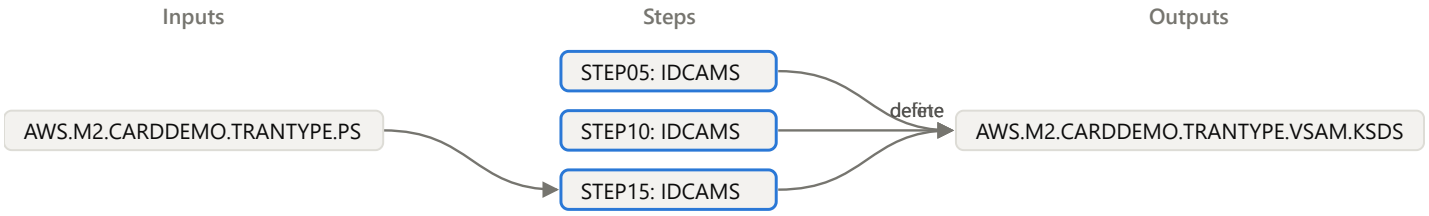
- **Two sources for the date range.** The SORT's dates are hard-coded in the JCL while the program reads DATEPARM; if they differ, the report heading can show one range while the lines come from another. The program's own date check doesn't work as written (an out-of-range record would end the report early), so today only the SORT's dates actually select the data.
- **Totals are slightly off at the end:** the last transaction is counted twice in the final page and grand totals, and the last card's total line is missing.
- The first two steps (unload, filter, sort) and the report program together amount to "select transactions by processing date, ordered by card, with lookups and subtotals" - a single query and a reporting tool on a modern platform.
- The full unload of the VSAM master on every run is mainly there to feed the SORT; a database-backed version wouldn't need it, but COMBTRAN also reads the backup generation, so that dependency has to move with it.

Job TRANTYPE

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	3	1	2
jobs	steps	system programs	datasets

36
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job TRANTYPE

Step 1 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE TRANSACATION TYPE VSAM FILE IF ONE ALREADY EXISTS
- Deletes: AWS.M2.CARDDEMO.TRANTYPE.VSAM.KSDS
- Control statements: `DELETE AWS.M2.CARDDEMO.TRANTYPE.VSAM.KSDS CLUSTER SET MAXCC = 0`

Step 2 - STEP10: runs IDCAMS (IBM utility)

- Comment: DEFINE TRANSACATION TYPE VSAM FILE
- Defines: AWS.M2.CARDDEMO.TRANTYPE.VSAM.KSDS
- Control statements: `DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.TRANTYPE.VSAM.KSDS) CYLINDERS(1 5) VOLUMES(AWSHJ1 ) KEYS(2 0) RECORDSIZE(60 60) SHAREOPTIONS(1 4) ERASE`

Step 3 - STEP15: runs IDCAMS (IBM utility)

- Comment: COPY DATA FROM FLAT FILE TO VSAM FILE
- Copies AWS.M2.CARDDEMO.TRANTYPE.PS into AWS.M2.CARDDEMO.TRANTYPE.VSAM.KSDS
- Reads: AWS.M2.CARDDEMO.TRANTYPE.PS (DD TRANTYPE)
- Writes: AWS.M2.CARDDEMO.TRANTYPE.VSAM.KSDS (DD TTYPVSAM)
- Control statements: `REPRO INFILE(TRANTYPE) OUTFILE(TTYPVSAM)`

Inputs from outside the job: AWS.M2.CARDDEMO.TRANTYPE.PS **Outputs it leaves behind:** AWS.M2.CARDDEMO.TRANTYPE.VSAM.KSDS

Job TXT2PDF1

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	1	1	2
jobs	steps	system programs	datasets

14
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job TXT2PDF1

Step 1 - TXT2PDF: runs IKJEFT1B (IBM utility)

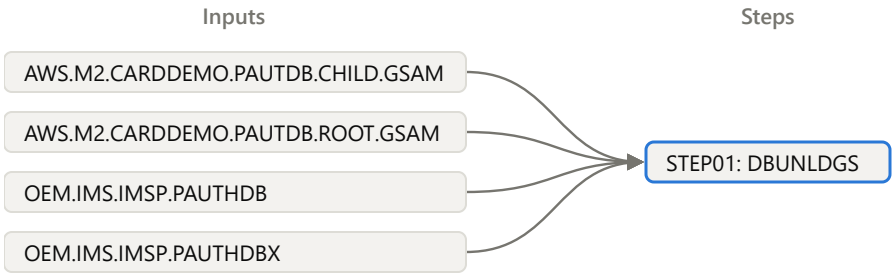
- Comment: CONVERT TEXT FILE TO A PDF FILE
- Uses: AWS.M2.CARDDEMO.STATEMNT.PS (DD INDD), AWS.M2.LBD.TXT2PDF.EXEC (DD SYSEXEC)

Inputs from outside the job: none **Outputs it leaves behind:** none

Job UNLDGSAM

Rehost suggested - Uses IMS databases; rehost first and plan the data migration separately.

1	1	1	1
jobs	steps	programs	system programs
6	**4**	**4**	**311**
copybooks	datasets	IMS segments	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job UNLDGSAM

Step 1 - STEP01: runs DBUNLDGS, through DFSRRC00

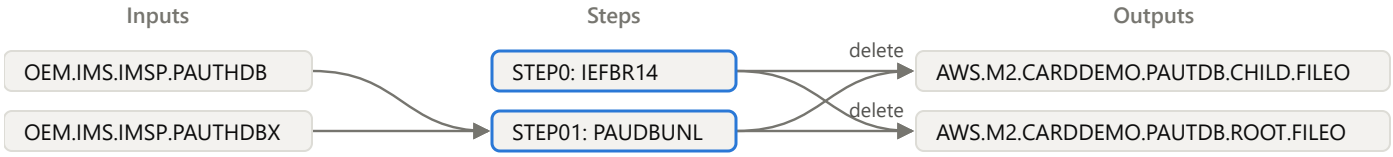
- Comment: EXECUTE IMS PROGRAM
- DBUNLDGS's comments: SELECT OPFILE1 ASSIGN TO OUTFIL1 / ORGANIZATION IS SEQUENTIAL / ACCESS MODE IS SEQUENTIAL / FILE STATUS IS WS-OUTFL1-STATUS.
- Uses: AWS.M2.CARDDEMO.PAUTDB.CHILD.GSAM (DD PADFILOP), AWS.M2.CARDDEMO.PAUTDB.ROOT.GSAM (DD PASFILOP), OEM.IMS.IMSP.PAUTHDB (DD DDPAUTP0), OEM.IMS.IMSP.PAUTHDBX (DD DDPAUTX0)
- Adds records to IMS segment (PCB PADFLPCB)
- Adds records to IMS segment (PCB PASFLPCB)
- Reads IMS segment PAUTDTL1
- Reads IMS segment PAUTSUM0

Inputs from outside the job: none **Outputs it leaves behind:** none

Job UNLDPADB

Rehost suggested - Uses IMS databases; rehost first and plan the data migration separately.

1 jobs	2 steps	1 programs	2 system programs
4 copybooks	4 datasets	2 IMS segments	2 record layouts
312 lines of source			



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job UNLDPADB

Step 1 - STEP0: runs IEFBR14 (IBM utility)

- Comment: EXECUTE IMS PROGRAM
- Deletes: AWS.M2.CARDDEMO.PAUTDB.CHILD.FILEO (DD DD2), AWS.M2.CARDDEMO.PAUTDB.ROOT.FILEO (DD DD1)

Step 2 - STEP01: runs PAUDBUNL, through DFSRRC00

- Creates: AWS.M2.CARDDEMO.PAUTDB.CHILD.FILEO (DD OUTFIL2), AWS.M2.CARDDEMO.PAUTDB.ROOT.FILEO (DD OUTFIL1)
- Uses: OEM.IMS.IMSP.PAUTHDB (DD DDPAUTP0), OEM.IMS.IMSP.PAUTHDBX (DD DDPAUTX0)
- Reads IMS segment PAUTDTL1
- Reads IMS segment PAUTSUM0

Inputs from outside the job: none **Outputs it leaves behind:** AWS.M2.CARDDEMO.PAUTDB.CHILD.FILEO, AWS.M2.CARDDEMO.PAUTDB.ROOT.FILEO

Job WAITSTEP

Refactor suggested - 2 program(s), 35 lines: small enough to rewrite as a batch service, query, or report on the target platform.

1	1	2	35
jobs	steps	programs	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job WAITSTEP

Step 1 - WAIT: runs COBSWAIT

- Comment: WAIT FOR CENTISECONDS IN THE PARM EG: 00003600 = 36 SECONDS
- COBSWAIT's comments: PROGRAM : COBSWAIT.CBL / Application : CardDemo / Type : BATCH COBOL Program / FUNCTION : UTILITY PROGRAM TO WAIT (PARM IN CENTISECONDS)

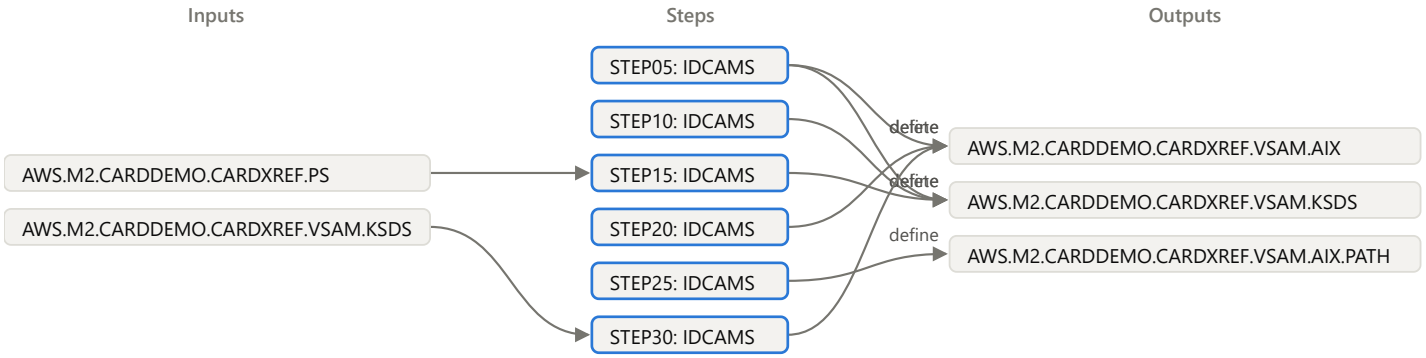
Inputs from outside the job: none **Outputs it leaves behind:** none

Job XREFFILE

Refactor suggested - Only IBM utilities (copy, sort, unload): replace with SQL or the target platform's ETL and scheduling tools.

1	6	1	4
jobs	steps	system programs	datasets

68
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Job XREFFILE

Step 1 - STEP05: runs IDCAMS (IBM utility)

- Comment: DELETE CARD XREF VSAM FILE IF ONE ALREADY EXISTS
- Deletes: AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX, AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS
- Control statements: `DELETE AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS CLUSTER IF MAXCC LE 08 THEN SET MAXCC = 0 DELETE AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX ALTERNATEINDEX IF MAXCC LE 08 THEN SET MAXCC = 0`

Step 2 - STEP10: runs IDCAMS (IBM utility)

- Comment: DEFINE CARD XREF VSAM FILE
- Defines: AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS
- Control statements: `DEFINE CLUSTER (NAME(AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS) CYLINDERS(1 5) VOLUMES(AWSHJ1 ) KEYS(16 0) RECORDSIZE(50 50) SHAREOPTIONS(2 3) ERASE`

Step 3 - STEP15: runs IDCAMS (IBM utility)

- Comment: COPY DATA FROM FLAT FILE TO VSAM FILE
- Copies AWS.M2.CARDDEMO.CARDXREF.PS into AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS
- Reads: AWS.M2.CARDDEMO.CARDXREF.PS (DD XREFDATA)
- Writes: AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS (DD XREFVSAM)
- Control statements: `REPRO INFILE(XREFDATA) OUTFILE(XREFVSAM)`

Step 4 - STEP20: runs IDCAMS (IBM utility)

- Comment: CREATE ALTERNATE INDEX ON ACCT ID
- Defines: AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX

- Control statements: `DEFINE ALTERNATEINDEX (NAME(AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX)-
RELATE(AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS) KEYS(11,25) NONUNIQUEKEY UPGRADE RECORDSIZE(50,50)
FREESPACE(10,20) VOLUMES(AWSHJ1)`

Step 5 - STEP25: runs IDCAMS (IBM utility)

- Comment: DEFINE PATH IS USED TO RELATE THE ALTERNATE INDEX TO BASE CLUSTER
- Defines: AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX.PATH
- Control statements: `DEFINE PATH (NAME(AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX.PATH)
PATHENTRY(AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX))`

Step 6 - STEP30: runs IDCAMS (IBM utility)

- Comment: BUILD ALTERNATE INDEX CLUSTER
- BLDINDEX AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS into AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX
- Reads: AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS
- Writes: AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX
- Control statements: `BLDINDEX INDATASET(AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS)
OUTDATASET(AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX)`

Inputs from outside the job: AWS.M2.CARDDEMO.CARDXREF.PS **Outputs it leaves behind:** AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX

Transaction CA00

Refactor suggested - Screen and navigation only (no data access): rebuild in the new user interface.

1	7	1	1
programs	copybooks	transactions	screens (mapsets)

695
lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CA00

Starting transaction CA00 runs program COADM01C (from the CSD).

Program COADM01C

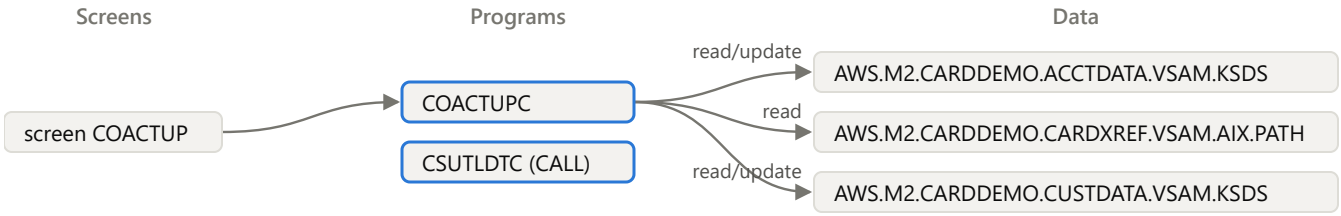
- Its comments: Program : COADM01C.CBL / Application : CardDemo / Type : CICS COBOL Program / Function : Admin Menu for Admin users
- Screen: sends and receives mapset COADM01
- Ends the turn and waits for the user; the next input runs transaction CA00 again
- Transfers control (XCTL) to COSGN00C (entry program of transaction CC00)
- Transfers control (XCTL) to COTRTLIC (entry program of transaction CTLI)
- Transfers control (XCTL) to COTRTUPC (entry program of transaction CTTU)
- Transfers control (XCTL) to COUSR00C (entry program of transaction CU00)
- Transfers control (XCTL) to COUSR01C (entry program of transaction CU01)
- Transfers control (XCTL) to COUSR02C (entry program of transaction CU02)
- Transfers control (XCTL) to COUSR03C (entry program of transaction CU03)

Where it connects to other work: XCTL PROGRAM COSGN00C (entry program of transaction CC00); XCTL PROGRAM COTRTLIC (entry program of transaction CTLI); XCTL PROGRAM COTRTUPC (entry program of transaction CTTU); XCTL PROGRAM COUSR00C (entry program of transaction CU00); XCTL PROGRAM COUSR01C (entry program of transaction CU01); XCTL PROGRAM COUSR02C (entry program of transaction CU02); XCTL PROGRAM COUSR03C (entry program of transaction CU03)

Transaction CAUP

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

2 programs	**1** system programs	**16** copybooks	**1** transactions
1 screens (mapsets)	**3** datasets	**5** record layouts	**6,518** lines of source



What it does

CAUP lets a user look up an account and change it: the account's status, dates, credit limits, and balances, and the account holder's name, address, phone numbers, SSN, date of birth, FICO score, and EFT account. Changes are validated field by field, shown back for confirmation, and saved to the account and customer records together, with protection against two people updating the same account at once.

Step by step

1. From the main menu (option 2), COACTUPC shows an empty screen asking for an account number.
2. The user enters an account number; the program finds the account's customer through the card cross-reference, reads the account and customer records, and shows them.
3. The user changes fields and presses Enter. If anything changed, every field is validated - required fields, dates, amounts, US state and ZIP consistency, phone format, SSN rules, FICO score 300-850 - and errors are shown on the screen.
4. When the changes pass, the screen asks the user to press PF5 to save. PF12 cancels the changes and shows the stored data again.
5. On PF5 the program locks both records, checks nobody changed them since they were shown, and rewrites the account and then the customer; if the customer can't be saved, both changes are rolled back. It reports success, a lock or update failure, or "changed by someone else" (showing the current data).
6. PF3 returns to the main menu.

What it needs

- The account master (`ACCTDATA.VSAM.KSDS`), the customer master (`CUSTDATA.VSAM.KSDS`), and the card cross-reference by account (`CARDXREF.VSAM.AIX.PATH`).
- The date-checking routine CSUTLDTC (IBM CEEDAYS).

What it produces

- Updated account and customer records - used by posting (POSTTRAN, credit limit and expiry checks), interest (INTCALC, balance and cycle totals), statements (CREASTMT, name, address, balance, FICO score), and the other account and card screens.
- These files are also read by batch jobs outside this screen: the account file by ACCTFILE, CBEXPORT, CREASTMT, INTCALC, POSTTRAN, and READACCT, and the customer file by CBEXPORT, CREASTMT, CUSTFILE, and READCUST.

Notes for modernization

- **Balances and limits can be typed in:** any user can change the current balance, cycle totals, and credit limits directly, bypassing posting - a control gap to raise with the client, and a rule to decide before building the new screen.
- **Strong update pattern to keep:** confirmation, locking, the "changed by someone else" check, and the joint save map directly onto a database transaction with optimistic concurrency.
- The business rules (state/ZIP check, SSN rules, FICO range, phone format) are worth extracting into a shared validation service; most of the program's 3,400 lines are screen handling that a web form replaces.
- **Expose as API:** "get account with holder" and "update account with holder" are natural service operations behind a new web screen, with the batch jobs above reading the same data.
- Leaving with PF3 resets the session's user type to regular user (see COACTUPC).

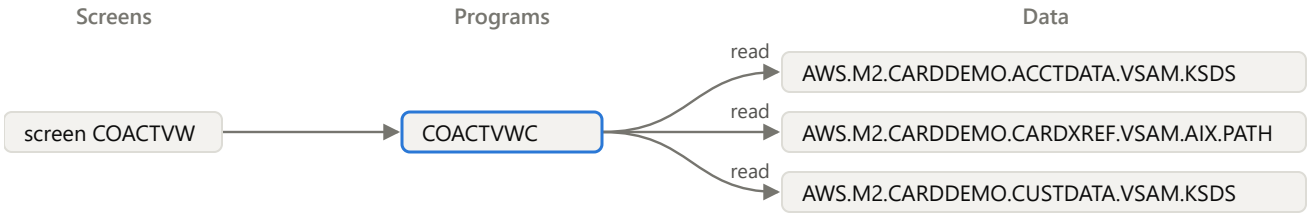
Where it connects to other work: XCTL PROGRAM COMEN01C (entry program of transaction CM00); XCTL PROGRAM COMEN01C (back to the caller)

Transaction CAVW

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	13	1	1
programs	copybooks	transactions	screens (mapsets)

3	3	1,745
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CAVW

Starting transaction CAVW runs program COACTVWC (from the CSD).

Program COACTVWC

- Its comments: Program: COACTVWC.CBL / Layer: Business logic / Function: Accept and process Account View request
- Screen: sends and receives mapset COACTVW
- Reads file AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS
- Reads file AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX.PATH
- Reads file AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS
- Ends the turn and waits for the user; the next input runs transaction CAVW again
- Transfers control (XCTL) to COMEN01C (entry program of transaction CM00)
- Transfers control (XCTL) to COMEN01C (back to the caller)

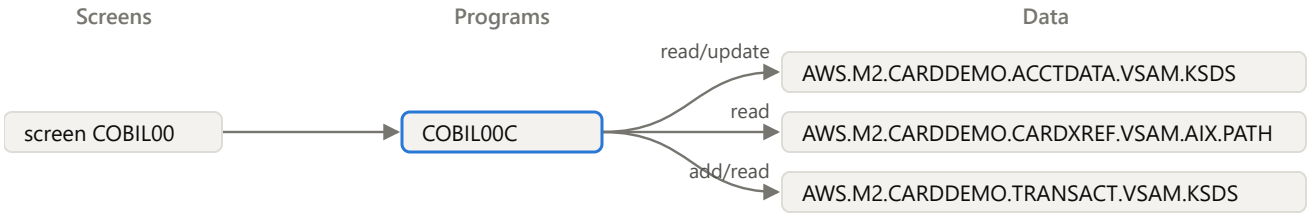
Where it connects to other work: XCTL PROGRAM COMEN01C (entry program of transaction CM00); XCTL PROGRAM COMEN01C (back to the caller)

Transaction CB00

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	8	1	1
programs	copybooks	transactions	screens (mapsets)

3	3	774
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CB00

Starting transaction CB00 runs program COBIL00C (from the CSD).

Program COBIL00C

- Its comments: Program : COBIL00C.CBL / Application : CardDemo / Type : CICS COBOL Program / Function : Bill Payment - Pay account balance in full and a
- Screen: sends and receives mapset COBIL00
- Reads and updates file AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS
- Reads file AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX.PATH
- Adds records to and reads file AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
- Ends the turn and waits for the user; the next input runs transaction CB00 again
- Transfers control (XCTL) to COMEN01C (entry program of transaction CM00)
- Transfers control (XCTL) to COMEN01C (back to the caller)
- Transfers control (XCTL) to COSGN00C (entry program of transaction CC00)

Where it connects to other work: XCTL PROGRAM COMEN01C (entry program of transaction CM00); XCTL PROGRAM COMEN01C (back to the caller); XCTL PROGRAM COSGN00C (entry program of transaction CC00)

Transaction CC00

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	6	1	1
programs	copybooks	transactions	screens (mapsets)

1	1	581
datasets	record layouts	lines of source



What it does

CC00 is how every CardDemo user starts: the sign-on screen. The user types a user ID and password; the program checks them against the user security file and sends administrators to the admin menu and everyone else to the main menu, carrying their identity and role to every screen after it. PF3 ends the session.

Step by step

1. The user starts transaction CC00 (or returns to it from any screen with PF3); COSGN00C shows the sign-on screen with the date, time, and system name.
2. The user enters a user ID and password and presses Enter. Both are required and are converted to upper case.
3. The program reads the user's record from the user security file and compares the password.
4. On a match it records the user ID and type in the communication area and transfers to the admin menu (transaction CA00, program COADM01C) for an administrator, or the main menu (CM00, COMEN01C) for a regular user.
5. Otherwise it redisplays the screen with "Wrong Password", "User not found", or "Unable to verify the User", and waits for the next attempt.

What it needs

- The user security file `AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS` (user ID, names, password, user type), maintained by the admin user screens and loaded by batch.

What it produces

- No data changes - only the user's identity and role in the communication area, which the menus and every function screen rely on.

Notes for modernization

- **Passwords are kept and compared in clear text,** are case-insensitive, and are at most 8 characters; there's no lockout, and the messages reveal whether a user ID exists. Treat these as security findings for the client, separate from modernization.

- On a modern platform sign-on is replaced by the organization's identity provider (single sign-on), and the user security file becomes user and role data there - this transaction is retired rather than converted.
- Every other screen trusts the role set here; the target design should check authorization in the services, not only at sign-on.

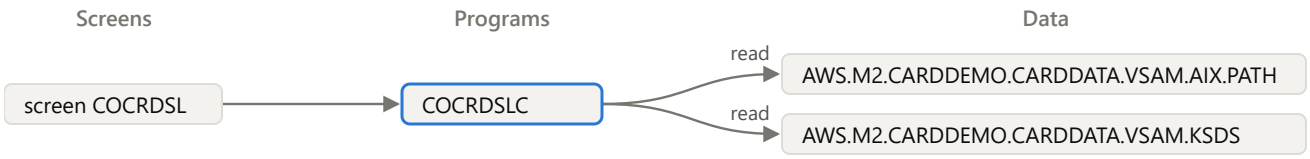
Where it connects to other work: XCTL PROGRAM COADM01C (entry program of transaction CA00); XCTL PROGRAM COMEN01C (entry program of transaction CM00)

Transaction CCDL

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	11	1	1
programs	copybooks	transactions	screens (mapsets)

2	1	1,180
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CCDL

Starting transaction CCDL runs program COCRDSLC (from the CSD).

Program COCRDSLC

- Its comments: Program: COCRDSLC.CBL / Layer: Business logic / Function: Accept and process credit card detail request
- Screen: sends and receives mapset COCRDSL
- Reads file AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX.PATH
- Reads file AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS
- Ends the turn and waits for the user; the next input runs transaction CCDL again
- Transfers control (XCTL) to COCRDLIC (back to the caller)
- Transfers control (XCTL) to COMEN01C (entry program of transaction CM00)
- Transfers control (XCTL) to COMEN01C (back to the caller)

Where it connects to other work: XCTL PROGRAM COCRDLIC (back to the caller); XCTL PROGRAM COMEN01C (entry program of transaction CM00); XCTL PROGRAM COMEN01C (back to the caller)

Transaction CCLI

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	9	1	1
programs	copybooks	transactions	screens (mapsets)

1	1	2,149
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CCLI

Starting transaction CCLI runs program COCRDLIC (from the CSD).

Program COCRDLIC

- Its comments: Program: COCRDLIC.CBL / Layer: Business logic / Function: List Credit Cards / a) All cards if no context passed and admin user
- Screen: sends and receives mapset COCRDLI
- Reads file AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS
- Ends the turn and waits for the user; the next input runs transaction CCLI again
- Transfers control (XCTL) to COCRDLIC (VALUE of LIT-THISPGM via CCARD-NEXT-PROG)
- Transfers control (XCTL) to COCRDSLC (entry program of transaction CCDL)
- Transfers control (XCTL) to COCRDUPC (entry program of transaction CCUP)
- Transfers control (XCTL) to COMEN01C (entry program of transaction CM00)

Where it connects to other work: XCTL PROGRAM COCRDSLC (entry program of transaction CCDL); XCTL PROGRAM COCRDUPC (entry program of transaction CCUP); XCTL PROGRAM COMEN01C (entry program of transaction CM00)

Transaction CCUP

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	11	1	1
programs	copybooks	transactions	screens (mapsets)

1	2	1,771
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CCUP

Starting transaction CCUP runs program COCRDUPC (from the CSD).

Program COCRDUPC

- Its comments: Program: COCRDUPC.CBL / Layer: Business logic / Function: Accept and process credit card detail request
- Screen: sends and receives mapset COCRDUP
- Reads and updates file AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS
- Ends the turn and waits for the user; the next input runs transaction CCUP again
- Transfers control (XCTL) to COCRDLIC (back to the caller)
- Transfers control (XCTL) to COMEN01C (entry program of transaction CM00)
- Transfers control (XCTL) to COMEN01C (back to the caller)

Where it connects to other work: XCTL PROGRAM COCRDLIC (back to the caller); XCTL PROGRAM COMEN01C (entry program of transaction CM00); XCTL PROGRAM COMEN01C (back to the caller)

Transaction CDRA

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	4	1	1
programs	system programs	copybooks	transactions

1	1	514
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CDRA

Starting transaction CDRA runs program COACCT01 (from the CSD).

Program COACCT01

- Reads file AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS
- Calls MQCLOSE (IBM routine)
- Calls MQGET (IBM routine)
- Calls MQOPEN (IBM routine)
- Calls MQPUT (IBM routine)

Transaction CDRD

Refactor suggested - Screen and navigation only (no data access): rebuild in the new user interface.

1	4	1	409
programs	system programs	transactions	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CDRD

Starting transaction CDRD runs program CODATE01 (from the CSD).

Program CODATE01

- Calls MQCLOSE (IBM routine)
- Calls MQGET (IBM routine)
- Calls MQOPEN (IBM routine)
- Calls MQPUT (IBM routine)

Transaction CDV1

Retire suggested - PROGRAM COCRDSEC isn't in the source: confirm with the client whether it's still used, and retire it if not.

1	0
transactions	lines of source

A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CDV1

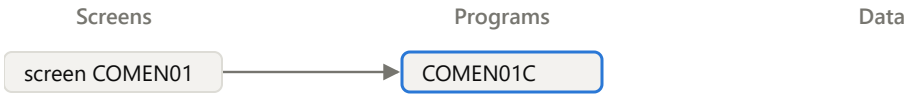
Starting transaction CDV1 runs program COCRDSEC (from the CSD).

Transaction CM00

Refactor suggested - Screen and navigation only (no data access): rebuild in the new user interface.

1	7	1	1
programs	copybooks	transactions	screens (mapsets)

751
lines of source



What it does

CM00 is the main menu for regular users. It lists the eleven things a user can do - view or update accounts, list, view, or update credit cards, list, view, or add transactions, run transaction reports, pay bills, and view pending authorizations - and starts the chosen function. Each function returns here when the user presses PF3, and PF3 here returns to sign-on.

Step by step

1. After sign-on (or on returning from a function), COMEN01C shows the numbered menu.
2. The user types an option number and presses Enter.
3. The program checks the number is between 1 and 11 and that the user's role allows the option, then transfers to that function's program - for example option 2 starts the account update (COACTUPC) and option 8 the transaction add (COTRN02C).
4. For Pending Authorization View, it first checks the optional authorization module is installed, and says so if it isn't.
5. PF3 returns to the sign-on screen.

What it needs

- The user's identity and role from sign-on (communication area).
- The menu options defined in copybook COMEN02Y - names, programs, and which user type may use each.

What it produces

- No data changes - it only navigates to the eleven function screens.

Notes for modernization

- **Every option is open to every user:** the menu supports admin-only options, but none is marked admin-only (Transaction Add's "Admin Only" label is commented out), so regular users can add transactions and update accounts, balances, and limits. Agree the intended permissions with the business and enforce them in the services.
- The menu itself is navigation; in a web front end it becomes the application's navigation bar, and this transaction is retired.

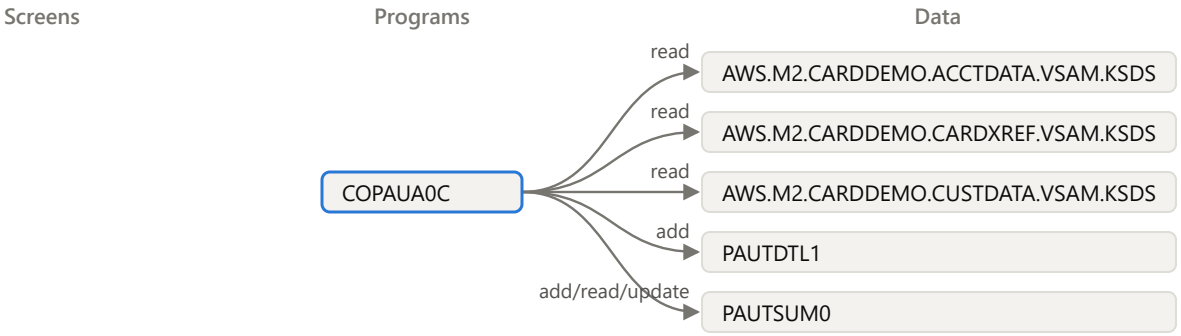
- The options table in a copybook is a simple, readable source for the target application's menu and permissions.

Where it connects to other work: XCTL PROGRAM COACTUPC (entry program of transaction CAUP); XCTL PROGRAM COACTVWC (entry program of transaction CAVW); XCTL PROGRAM COBIL00C (entry program of transaction CB00); XCTL PROGRAM COCRDLIC (entry program of transaction CCLI); XCTL PROGRAM COCRDSLC (entry program of transaction CCDL); XCTL PROGRAM COCRDUPC (entry program of transaction CCUP); XCTL PROGRAM COPAUS0C (entry program of transaction CPVS); XCTL PROGRAM CORPT00C (entry program of transaction CR00); XCTL PROGRAM COSGN00C (entry program of transaction CC00); XCTL PROGRAM COTRN00C (entry program of transaction CT00); XCTL PROGRAM COTRN01C (entry program of transaction CT01); XCTL PROGRAM COTRN02C (entry program of transaction CT02)

Transaction CP00

Rehost suggested - 905 lines with IMS or heavy logic: rehost, then expose services.

1 programs	4 system programs	8 copybooks	1 transactions
3 datasets	2 IMS segments	3 record layouts	905 lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CP00

Starting transaction CP00 runs program COPAUA0C (from the CSD).

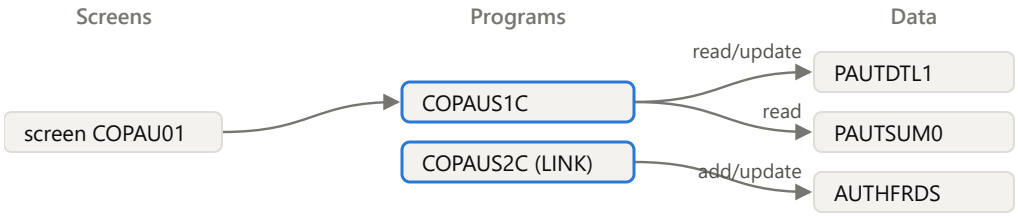
Program COPAUA0C

- Its comments: Program : COPAUA0C.CBL / Application : CardDemo - Authorization Module / Type : CICS COBOL IMS MQ Program / Function : Card Authorization Decision Program
- Reads file AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS
- Reads file AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS
- Reads file AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS
- Adds records to IMS segment PAUTDTL1
- Adds records to and reads and updates IMS segment PAUTSUM0
- Calls MQCLOSE (IBM routine)
- Calls MQGET (IBM routine)
- Calls MQOPEN (IBM routine)
- Calls MQPUT1 (IBM routine)

Transaction CPVD

Rehost suggested - 1,461 lines with IMS or heavy logic: rehost, then expose services.

2 programs	8 copybooks	1 DCLGENs	1 transactions
1 screens (mapsets)	1 Db2 tables	2 IMS segments	1,461 lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CPVD

Starting transaction CPVD runs program COPAUS1C (from the CSD).

Program COPAUS1C

- Its comments: Program : COPAUS1C.CBL / Application : CardDemo - Authorization Module / Type : CICS COBOL IMS BMS Program / Function : Detail View of Authorization Message
- Screen: sends and receives mapset COPAU01
- Reads and updates IMS segment PAUTDTL1
- Reads IMS segment PAUTSUM0
- Links COPAUS2C
- Ends the turn and waits for the user; the next input runs transaction CPVD again
- Transfers control (XCTL) to COPAUS0C (entry program of transaction CPVS)

Program COPAUS2C (LINK from its caller)

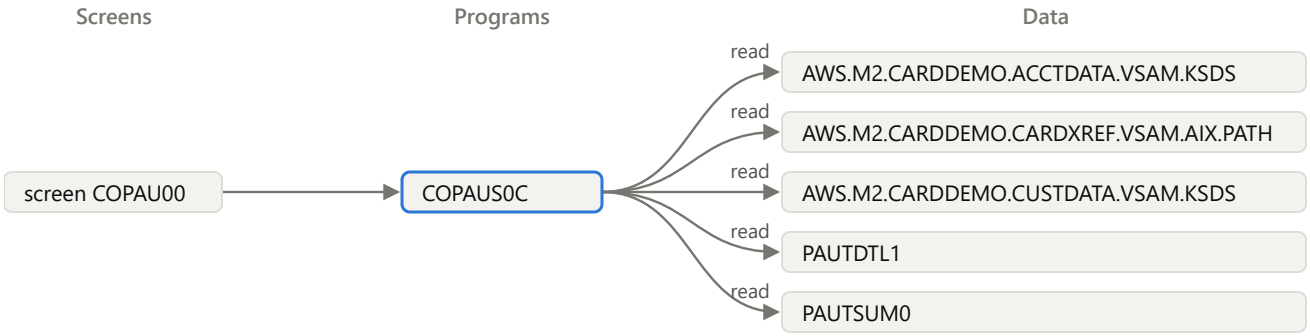
- Its comments: Program : COPAUS2C.CBL / Application : CardDemo - Authorization Module / Type : CICS COBOL IMS DB2 Program / Function : Mark Authorization Message Fraud
- Adds records to and updates Db2 table AUTHFRDS

Where it connects to other work: XCTL PROGRAM COPAUS0C (entry program of transaction CPVS)

Transaction CPVS

Rehost suggested - 2,218 lines with IMS or heavy logic: rehost, then expose services.

1 programs	12 copybooks	1 transactions	1 screens (mapsets)
3 datasets	2 IMS segments	3 record layouts	2,218 lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CPVS

Starting transaction CPVS runs program COPAUS0C (from the CSD).

Program COPAUS0C

- Its comments: Program : COPAUS0C.CBL / Application : CardDemo - Authorization Module / Type : CICS COBOL IMS BMS Program / Function : Summary View of Authoriation Messages
- Screen: sends and receives mapset COPAU00
- Reads file AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS
- Reads file AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX.PATH
- Reads file AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS
- Reads IMS segment PAUTDTL1
- Reads IMS segment PAUTSUM0
- Ends the turn and waits for the user; the next input runs transaction CPVS again
- Transfers control (XCTL) to COMEN01C (entry program of transaction CM00)
- Transfers control (XCTL) to COPAUS0C (VALUE of WS-PGM-AUTH-SMRY via CDEMO-TO-PROGRAM)
- Transfers control (XCTL) to COPAUS1C (entry program of transaction CPVD)
- Transfers control (XCTL) to COSGN00C (entry program of transaction CC00)

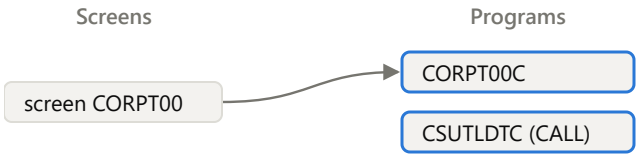
Where it connects to other work: XCTL PROGRAM COMEN01C (entry program of transaction CM00); XCTL PROGRAM COPAUS1C (entry program of transaction CPVD); XCTL PROGRAM COSGN00C (entry program of transaction CC00)

Transaction CR00

Refactor suggested - Screen and navigation only (no data access): rebuild in the new user interface.

2	1	6	1
programs	system programs	copybooks	transactions

1	1,122
screens (mapsets)	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CR00

Starting transaction CR00 runs program CORPT00C (from the CSD).

Program CORPT00C

- Its comments: Program : CORPT00C.CBL / Application : CardDemo / Type : CICS COBOL Program / Function : Print Transaction reports by submitting batch
- Screen: sends and receives mapset CORPT00
- Calls CSUTLDTC
- Ends the turn and waits for the user; the next input runs transaction CR00 again
- Transfers control (XCTL) to COMEN01C (entry program of transaction CM00)
- Transfers control (XCTL) to COSGN00C (entry program of transaction CC00)

Program CSUTLDTC (CALL from its caller)

- What it does: CSUTLDTC is a small shared routine that checks whether a date is valid for a given format (for example YYYY-MM-DD). It asks the IBM Language Environment date service CEEDAYS to convert the date, and returns a readable result message and a severity as the return code. The account update, transaction add, and transaction report screens use it.
- Calls CEEDAYS (IBM routine)

Where it connects to other work: XCTL PROGRAM COMEN01C (entry program of transaction CM00); XCTL PROGRAM COSGN00C (entry program of transaction CC00)

Transaction CT00

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	6	1	1
programs	copybooks	transactions	screens (mapsets)

1	1	1,775
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CT00

Starting transaction CT00 runs program COTRN00C (from the CSD).

Program COTRN00C

- Its comments: Program : COTRN00C.CBL / Application : CardDemo / Type : CICS COBOL Program / Function : List Transactions from TRANSACT file
- Screen: sends and receives mapset COTRN00
- Reads file AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
- Ends the turn and waits for the user; the next input runs transaction CT00 again
- Transfers control (XCTL) to COMEN01C (entry program of transaction CM00)
- Transfers control (XCTL) to COSGN00C (entry program of transaction CC00)
- Transfers control (XCTL) to COTRN01C (entry program of transaction CT01)

Where it connects to other work: XCTL PROGRAM COMEN01C (entry program of transaction CM00); XCTL PROGRAM COSGN00C (entry program of transaction CC00); XCTL PROGRAM COTRN01C (entry program of transaction CT01)

Transaction CT01

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	6	1	1
programs	copybooks	transactions	screens (mapsets)

1	1	831
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CT01

Starting transaction CT01 runs program COTRN01C (from the CSD).

Program COTRN01C

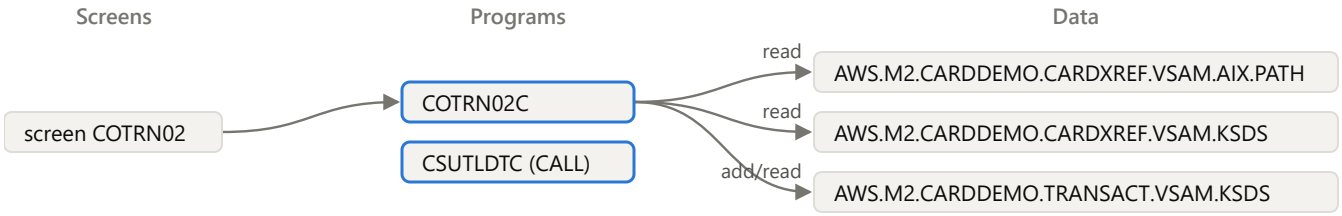
- Its comments: Program : COTRN01C.CBL / Application : CardDemo / Type : CICS COBOL Program / Function : View a Transaction from TRANSACT file
- Screen: sends and receives mapset COTRN01
- Reads file AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS
- Ends the turn and waits for the user; the next input runs transaction CT01 again
- Transfers control (XCTL) to COMEN01C (entry program of transaction CM00)
- Transfers control (XCTL) to COMEN01C (back to the caller)
- Transfers control (XCTL) to COSGN00C (entry program of transaction CC00)
- Transfers control (XCTL) to COTRN00C (entry program of transaction CT00)
- Transfers control (XCTL) to COTRN00C (back to the caller)

Where it connects to other work: XCTL PROGRAM COMEN01C (entry program of transaction CM00); XCTL PROGRAM COMEN01C (back to the caller); XCTL PROGRAM COSGN00C (entry program of transaction CC00); XCTL PROGRAM COTRN00C (entry program of transaction CT00); XCTL PROGRAM COTRN00C (back to the caller)

Transaction CT02

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

2 programs	**1** system programs	**8** copybooks	**1** transactions
1 screens (mapsets)	**3** datasets	**2** record layouts	**1,381** lines of source



What it does

CT02 lets a user enter a card transaction by hand. The user picks the card (by account or card number), types the transaction details - type, category, source, description, amount, dates, merchant - confirms with Y, and the transaction is written to the transaction master with the next transaction number.

Step by step

1. From the main menu (option 8), or from a transaction list with a card already selected, COTRN02C shows the add-transaction form.
2. The user enters an account ID or a card number; the program looks up the other in the card cross-reference.
3. The user fills in the transaction fields and presses Enter. Each field is checked in turn (required, numeric codes, amount format, real dates, numeric merchant ID) and the first problem is shown.
4. The user confirms with Y. The program reads the highest transaction ID on the master, adds 1, and writes the new transaction, then shows the new transaction ID.
5. PF5 copies the details of the most recent transaction into the form, PF4 clears it, and PF3 returns to the main menu.

What it needs

- The card cross-reference (`CARDXREF.VSAM.KSDS` and its account index) and the transaction master (`TRANSACT.VSAM.KSDS`).
- The date-checking routine CSUTLDTC.

What it produces

- A new record on the transaction master - which the transaction report (TRANREPT), statements (CREASTMT), and the backup and combine jobs (TRANBKP, COMBTRAN, CBEXPORT) all read.

Notes for modernization

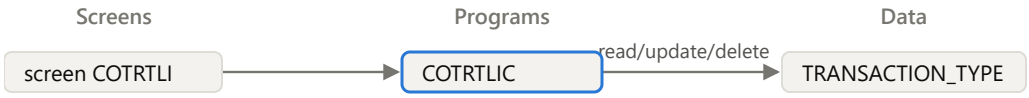
- **Manually added transactions skip posting:** they aren't checked against the account (status, credit limit, expiration) and don't update the account balance or category balances, so balances, interest, and statements won't reflect them. Confirm with the business what this screen is for (adjustments? testing?) and route it through the same posting logic as the daily batch.
- **Next-ID-plus-one numbering** can collide when two users add at once; use a database sequence.
- Type and category codes aren't checked against the reference files, and the processing date is typed in rather than set by the system.
- Available to all users (the menu's "Admin Only" restriction is commented out).
- As an API this becomes "post a transaction", shared with batch posting so online and batch apply the same rules.

Where it connects to other work: XCTL PROGRAM COMEN01C (entry program of transaction CM00); XCTL PROGRAM COMEN01C (back to the caller); XCTL PROGRAM COSGN00C (entry program of transaction CC00)

Transaction CTLI

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1 programs	1 system programs	11 copybooks	1 DCLGENs
1 transactions	1 screens (mapsets)	1 Db2 tables	2,674 lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CTLI

Starting transaction CTLI runs program COTRTLIC (from the CSD).

Program COTRTLIC

- Its comments: Program: COTRTLIC.CBL / Layer: Business logic / Function: List Transaction Type for updates and deletes / Demonstrates paging with cursors in Db2
- Screen: sends and receives mapset COTRTLI
- Reads and updates and deletes from Db2 table TRANSACTION_TYPE
- Calls DSNTIAC (IBM routine)
- Ends the turn and waits for the user; the next input runs transaction CTLI again
- Transfers control (XCTL) to COADM01C (entry program of transaction CA00)
- Transfers control (XCTL) to COADM01C (back to the caller)
- Transfers control (XCTL) to COTRTUPC (entry program of transaction CTTU)
- Transfers control (XCTL) to COTRTUPC (entry program of transaction CTTU)

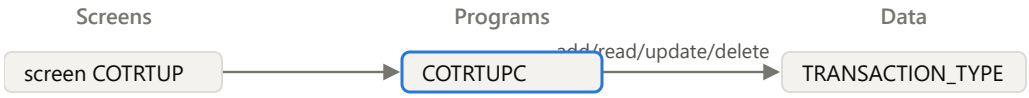
Where it connects to other work: XCTL PROGRAM COADM01C (entry program of transaction CA00); XCTL PROGRAM COADM01C (back to the caller); XCTL PROGRAM COTRTUPC (entry program of transaction CTTU)

Transaction CTTU

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	11	2	1
programs	copybooks	DCLGENs	transactions

1	1	1,851
screens (mapsets)	Db2 tables	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CTTU

Starting transaction CTTU runs program COTRTUPC (from the CSD).

Program COTRTUPC

- Its comments: Program: COTRTUPC.CBL / Layer: Business logic / Function: Accept and process TRANSACTION TYPE UPDATE
- Screen: sends and receives mapset COTRTUP
- Adds records to and reads and updates and deletes from Db2 table TRANSACTION_TYPE
- Ends the turn and waits for the user; the next input runs transaction CTTU again
- Transfers control (XCTL) to COADM01C (entry program of transaction CA00)
- Transfers control (XCTL) to COADM01C (back to the caller)
- Transfers control (XCTL) to COTRTLIC (back to the caller)

Where it connects to other work: XCTL PROGRAM COADM01C (entry program of transaction CA00); XCTL PROGRAM COADM01C (back to the caller); XCTL PROGRAM COTRTLIC (back to the caller)

Transaction CU00

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	6	1	1
programs	copybooks	transactions	screens (mapsets)

1	1	1,769
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CU00

Starting transaction CU00 runs program COUSR00C (from the CSD).

Program COUSR00C

- Its comments: Program : COUSR00C.CBL / Application : CardDemo / Type : CICS COBOL Program / Function : List all users from USRSEC file
- Screen: sends and receives mapset COUSR00
- Reads file AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS
- Ends the turn and waits for the user; the next input runs transaction CU00 again
- Transfers control (XCTL) to COADM01C (entry program of transaction CA00)
- Transfers control (XCTL) to COSGN00C (entry program of transaction CC00)
- Transfers control (XCTL) to COUSR02C (entry program of transaction CU02)
- Transfers control (XCTL) to COUSR03C (entry program of transaction CU03)

Where it connects to other work: XCTL PROGRAM COADM01C (entry program of transaction CA00); XCTL PROGRAM COSGN00C (entry program of transaction CC00); XCTL PROGRAM COUSR02C (entry program of transaction CU02); XCTL PROGRAM COUSR03C (entry program of transaction CU03)

Transaction CU01

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	6	1	1
programs	copybooks	transactions	screens (mapsets)

1	1	573
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CU01

Starting transaction CU01 runs program COUSR01C (from the CSD).

Program COUSR01C

- Its comments: Program : COUSR01C.CBL / Application : CardDemo / Type : CICS COBOL Program / Function : Add a new Regular/Admin user to USRSEC file
- Screen: sends and receives mapset COUSR01
- Adds records to file AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS
- Ends the turn and waits for the user; the next input runs transaction CU01 again
- Transfers control (XCTL) to COADM01C (entry program of transaction CA00)
- Transfers control (XCTL) to COSGN00C (entry program of transaction CC00)

Where it connects to other work: XCTL PROGRAM COADM01C (entry program of transaction CA00); XCTL PROGRAM COSGN00C (entry program of transaction CC00)

Transaction CU02

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	6	1	1
programs	copybooks	transactions	screens (mapsets)

1	1	683
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CU02

Starting transaction CU02 runs program COUSR02C (from the CSD).

Program COUSR02C

- Its comments: Program : COUSR02C.CBL / Application : CardDemo / Type : CICS COBOL Program / Function : Update a user in USRSEC file
- Screen: sends and receives mapset COUSR02
- Reads and updates file AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS
- Ends the turn and waits for the user; the next input runs transaction CU02 again
- Transfers control (XCTL) to COADM01C (entry program of transaction CA00)
- Transfers control (XCTL) to COADM01C (back to the caller)
- Transfers control (XCTL) to COSGN00C (entry program of transaction CC00)
- Transfers control (XCTL) to COUSR00C (back to the caller)

Where it connects to other work: XCTL PROGRAM COADM01C (entry program of transaction CA00); XCTL PROGRAM COADM01C (back to the caller); XCTL PROGRAM COSGN00C (entry program of transaction CC00); XCTL PROGRAM COUSR00C (back to the caller)

Transaction CU03

Expose as API suggested - Reads or updates business data behind a 3270 screen: keep the logic as a service (API) and replace the screen with a web interface.

1	6	1	1
programs	copybooks	transactions	screens (mapsets)

1	1	603
datasets	record layouts	lines of source



A reviewed plain-English summary isn't available yet; the step-by-step walkthrough below is built from the source.

Transaction CU03

Starting transaction CU03 runs program COUSR03C (from the CSD).

Program COUSR03C

- Its comments: Program : COUSR03C.CBL / Application : CardDemo / Type : CICS COBOL Program / Function : Delete a user from USRSEC file
- Screen: sends and receives mapset COUSR03
- Reads and deletes from file AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS
- Ends the turn and waits for the user; the next input runs transaction CU03 again
- Transfers control (XCTL) to COADM01C (entry program of transaction CA00)
- Transfers control (XCTL) to COADM01C (back to the caller)
- Transfers control (XCTL) to COSGN00C (entry program of transaction CC00)
- Transfers control (XCTL) to COUSR00C (back to the caller)

Where it connects to other work: XCTL PROGRAM COADM01C (entry program of transaction CA00); XCTL PROGRAM COADM01C (back to the caller); XCTL PROGRAM COSGN00C (entry program of transaction CC00); XCTL PROGRAM COUSR00C (back to the caller)

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	CBACT01C	CBACT02C	CBACT03C	CBACT04C	CBCUS01C	CBEXPORT	CBIMPORT	CBPAUP0C	CBSTM03A	CBSTM03B	CBTRN02C	CBTRN03C	COACCT01	COACTUPC	COACTVWC	COBIL00C	COBTUPDT	COCRDLIC	COCRDSLC	COCRDUPC	COPAUA0C	COPAUS0C	COPAUS1C	COPAUS2C	COSGN00C	COTRN00C	COTRN01C	COTRN02C	COTRTLIC	COTRTUPC	COUSR00C	COUSR01C	COUSR02C	COUSR03C	DBUNLDGS	PAUDBLOD	PAUDBUNL
AWS.M2.CARDDEMO.ACCTDATA.ARRYPS file	C																																			
AWS.M2.CARDDEMO.ACCTDATA.IMPORT file							C																													
AWS.M2.CARDDEMO.ACCTDATA.PSCOMP file	C																																			
AWS.M2.CARDDEMO.ACCTDATA.VBPS file	C																																			
AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS file	R			RU		R				R	RU		R	RU	R	RU					R	R														
AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX.PATH file																			R																	
AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS file		R				R												R	R	RU																
AWS.M2.CARDDEMO.CARDXREF.IMPORT file							C																													
AWS.M2.CARDDEMO.CARDXREF.VSAM.AIX.PATH file														R	R	R						R					R									
AWS.M2.CARDDEMO.CARDXREF.VSAM.KSDS file			R	R		R				R	R	R									R						R									
AWS.M2.CARDDEMO.CUSTDATA.IMPORT file							C																													
AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS file					R	R				R				RU	R						R	R														
AWS.M2.CARDDEMO.DALYREJS file											C																										
AWS.M2.CARDDEMO.DALYTRAN.PS file											R																										
AWS.M2.CARDDEMO.DATEPARM file												R																									
AWS.M2.CARDDEMO.DISCGRP.VSAM.KSDS file				R																																
AWS.M2.CARDDEMO.EXPORT.DATA file						C	R																													
AWS.M2.CARDDEMO.IMPORT.ERRORS file							C																													
AWS.M2.CARDDEMO.PAUTDB.CHILD.FILEO file																																			R	C
AWS.M2.CARDDEMO.PAUTDB.ROOT.FILEO file																																			R	C
AWS.M2.CARDDEMO.STATEMNT.HTML file									C																											
AWS.M2.CARDDEMO.STATEMNT.PS file									C																											
AWS.M2.CARDDEMO.SYSTRAN file				C																																
AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS file				R							CRU																									
AWS.M2.CARDDEMO.TRANCATG.VSAM.KSDS file												R																								
AWS.M2.CARDDEMO.TRANREPT file												C																								
AWS.M2.CARDDEMO.TRANSACT.DALY file												R																								
AWS.M2.CARDDEMO.TRANSACT.IMPORT file							C																													
AWS.M2.CARDDEMO.TRANSACT.VSAM.KSDS file						R					C					CR									R	R	CR									
AWS.M2.CARDDEMO.TRANTYPE.VSAM.KSDS file												R																								
AWS.M2.CARDDEMO.TRXFL.VSAM.KSDS file										R																										
AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS file																								R						R	C	RU	RD			
INPFILE file																	R																			
(PCB PADFLPCB) segment																																		C		
(PCB PASFLPCB) segment																																		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".

AUTHFRDS: DCLAUTHFRDS

- Stored in table CARDDEMO.AUTHFRDS: inserted by COPAUS2C; updated by COPAUS2C

Field	PIC	Bytes	Values	Set by
DCLAUTHFRDS		1-204		
CARD-NUM	X(16)	1-16		COPAUS2C
AUTH-TS	X(26)	17-42		COPAUS2C
AUTH-TYPE	X(4)	43-46		COPAUS2C
CARD-EXPIRY-DATE	X(4)	47-50		COPAUS2C
MESSAGE-TYPE	X(6)	51-56		COPAUS2C
MESSAGE-SOURCE	X(6)	57-62		COPAUS2C
AUTH-ID-CODE	X(6)	63-68		COPAUS2C
AUTH-RESP-CODE	X(2)	69-70		COPAUS2C
AUTH-RESP-REASON	X(4)	71-74		COPAUS2C
PROCESSING-CODE	X(6)	75-80		COPAUS2C
TRANSACTION-AMT	S9(10)V9(2)	81-87		COPAUS2C
APPROVED-AMT	S9(10)V9(2)	88-94		COPAUS2C
MERCHANT-CATAGORY-CODE	X(4)	95-98		COPAUS2C
ACQR-COUNTRY-CODE	X(3)	99-101		COPAUS2C
POS-ENTRY-MODE	S9(4)	102-103		COPAUS2C
MERCHANT-ID	X(15)	104-118		COPAUS2C
MERCHANT-NAME		119-142		COPAUS2C
MERCHANT-NAME-LEN	S9(4)	119-120		
MERCHANT-NAME-TEXT	X(22)	121-142		COPAUS2C
MERCHANT-CITY	X(13)	143-155		COPAUS2C
MERCHANT-STATE	X(2)	156-157		COPAUS2C
MERCHANT-ZIP	X(9)	158-166		COPAUS2C
TRANSACTION-ID	X(15)	167-181		COPAUS2C
MATCH-STATUS	X(1)	182-182		COPAUS2C
AUTH-FRAUD	X(1)	183-183		COPAUS2C
FRAUD-RPT-DATE	X(10)	184-193		
ACCT-ID	S9(11)V	194-199		COPAUS2C
CUST-ID	S9(9)V	200-204		COPAUS2C

CIPAUSMY: PENDING-AUTH-SUMMARY

- Stored in AWS.M2.CARDDEMO.PAUTDB.ROOT.FILEO: read by PAUDBLOD; written by PAUDBUNL

Field	PIC	Bytes	Values	Set by
PENDING-AUTH-SUMMARY		1-100		

PA-ACCT-ID	S9(11)	1-6	COPAUA0C (transaction CP00), COPAUS0C (transaction CPVS), COPAUS1C (transaction CPVD)
PA-CUST-ID	9(09)	7-15	COPAUA0C (transaction CP00)
PA-AUTH-STATUS	X(01)	16-16	
PA-ACCOUNT-STATUS	X(02)	17-26	
PA-CREDIT-LIMIT	S9(09)V99	27-32	COPAUA0C (transaction CP00)
PA-CASH-LIMIT	S9(09)V99	33-38	COPAUA0C (transaction CP00)
PA-CREDIT-BALANCE	S9(09)V99	39-44	COPAUA0C (transaction CP00)
PA-CASH-BALANCE	S9(09)V99	45-50	COPAUA0C (transaction CP00)
PA-APPROVED-AUTH-CNT	S9(04)	51-52	CBPAUP0C (CBPAUP0J), COPAUA0C (transaction CP00)
PA-DECLINED-AUTH-CNT	S9(04)	53-54	CBPAUP0C (CBPAUP0J), COPAUA0C (transaction CP00)
PA-APPROVED-AUTH-AMT	S9(09)V99	55-60	CBPAUP0C (CBPAUP0J), COPAUA0C (transaction CP00)
PA-DECLINED-AUTH-AMT	S9(09)V99	61-66	CBPAUP0C (CBPAUP0J), COPAUA0C (transaction CP00)

CSUSR01Y: SEC-USER-DATA

- Stored in AWS.M2.CARDDEMO.USRSEC.VSAM.KSDS: looked up by COSGN00C; read by COUSR00C; updated by COUSR02C, COUSR03C; written by COUSR01C

Field	PIC	Bytes	Values	Set by
SEC-USER-DATA		1-80		
SEC-USR-ID	X(08)	1-8		COUSR00C (transaction CU00)
SEC-USR-FNAME	X(20)	9-28		
SEC-USR-LNAME	X(20)	29-48		
SEC-USR-PWD	X(08)	49-56		
SEC-USR-TYPE	X(01)	57-57		
SEC-USR-FILLER	X(23)	58-80		

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)

CVACT02Y: CARD-RECORD

- Stored in AWS.M2.CARDDEMO.CARDDATA.VSAM.AIX.PATH: looked up by COCRDSLC
- Stored in AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS: looked up by COCRDSLC; read by CBACT02C, CBEXPORT, COCRDLIC; updated by COCRDUPC
- Stored in CARDOUT: written by CBIMPORT

One record per credit card: its number, the account it belongs to, the name embossed on it, its expiry date, and whether it's active. Kept in the card master file (AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS, keyed by card number), 150 bytes long, with an alternate index path (CARDDATA.VSAM.AIX.PATH) that the card screens use to find cards by account.

Fields

Field	Meaning	Notes
CARD-RECORD	The whole card record	Read by the card list and card view screens (CCLI, CCDL) and the batch card report and export; rewritten by the card update screen (CCUP)
CARD-NUM	Card number, 16 characters	The file's key. Kept as text, not a number
CARD-ACCT-ID	The account the card belongs to, 11 digits	Points to the account record (ACCT-ID). Probably the key of the alternate index the card screens use to list an account's cards
CARD-CVV-CD	Card verification value (CVV), 3 digits	Stored in clear - see the notes. Read by the card update screen
CARD-EMBOSSED-NAME	Cardholder name printed on the card, 50 characters	Changed on the card update screen (CCUP)
CARD-EXPIRAION-DATE	Card expiry date, 10 characters	Probably YYYY-MM-DD, kept as text. The name is misspelled ("EXPIRAION") in the source
CARD-ACTIVE-STATUS	Whether the card is active, one character	Probably Y or N; the copybook defines no values. Changed on the card update screen (CCUP)

Notes for modernization

- **Security:** the CVV is stored in clear in the card file. Card industry rules (PCI DSS) don't allow storing it after authorization at all; plan to drop it, not convert it.
- Only the import job and the card update screen write cards. Nothing in the source issues new cards, so card creation must happen outside this application.
- The card update screen also writes the record back from its own copy (CARD-UPDATE-RECORD), which has the same fields, so the update path is consistent with this layout.
- 59 bytes of the 150-byte record are unused FILLER.

Field	PIC	Bytes	Values	Set by
CARD-RECORD		1-150		
CARD-NUM	X(16)	1-16		CBIMPORT (CBIMPORT)
CARD-ACCT-ID	9(11)	17-27		CBIMPORT (CBIMPORT)
CARD-CVV-CD	9(03)	28-30		CBIMPORT (CBIMPORT)
CARD-EMBOSSED-NAME	X(50)	31-80		CBIMPORT (CBIMPORT), COCRDUPC (transaction CCUP)
CARD-EXPIRAION-DATE	X(10)	81-90		CBIMPORT (CBIMPORT)
CARD-ACTIVE-STATUS	X(01)	91-91		CBIMPORT (CBIMPORT), COCRDUPC (transaction CCUP)

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)

CVCUS01Y: CUSTOMER-RECORD

- Stored in AWS.M2.CARDDEMO.CUSTDATA.IMPORT: written by CBIMPORT
- Stored in AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS: looked up by COACTVWC, COPAUA0C, COPAUS0C; read by CBCUS01C, CBEXPORT; updated by COACTUPC

One record per customer: name, address, phone numbers, identity details, and credit score. Kept in the customer master file (AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS, keyed by customer number), 500 bytes long.

Fields

Field	Meaning	Notes
CUSTOMER-RECORD	The whole customer record	Read by the account view screen, authorization, the customer report, and the export; rewritten by the account update screen (CAUP)
CUST-ID	Customer number, 9 digits	The file's key. The card cross-reference (XREF-CUST-ID) points to it
CUST-FIRST-NAME	First name, 25 characters	
CUST-MIDDLE-NAME	Middle name, 25 characters	
CUST-LAST-NAME	Last name, 25 characters	

CUST-ADDR-LINE-1	Address line 1, 50 characters	
CUST-ADDR-LINE-2	Address line 2, 50 characters	
CUST-ADDR-LINE-3	Address line 3, 50 characters	Probably the city, since there's no separate city field
CUST-ADDR-STATE-CD	State code, 2 characters	Probably the US postal abbreviation
CUST-ADDR-COUNTRY-CD	Country code, 3 characters	Probably an ISO 3166 alpha-3 code such as USA
CUST-ADDR-ZIP	ZIP or postal code, 10 characters	
CUST-PHONE-NUM-1	Primary phone number, 15 characters	Kept as formatted text
CUST-PHONE-NUM-2	Second phone number, 15 characters	Kept as formatted text
CUST-SSN	Social Security number, 9 digits	Stored in clear - see the notes
CUST-GOVT-ISSUED-ID	Government-issued ID number (driver's license, passport, ...), 20 characters	Stored in clear
CUST-DOB-YYYY-MM-DD	Date of birth, YYYY-MM-DD as text	The format is in the field's name
CUST-EFT-ACCOUNT-ID	Bank account for electronic funds transfer, 10 characters	Probably the account payments are drawn from; nothing in the source uses it for a payment
CUST-PRI-CARD-HOLDER-IND	Whether the customer is the primary cardholder, one character	Probably Y or N; the copybook defines no values
CUST-FICO-CREDIT-SCORE	FICO credit score, 3 digits	Probably 300-850. Displayed and editable on the account update screen; nothing in the source uses it to decide anything

Notes for modernization

- **Sensitive data in clear:** SSN, government ID, date of birth, and the EFT bank account are stored unencrypted. Plan encryption or tokenization in the target, and mask them in test data.
- The account update screen (CAUP) maintains every field through its own copy of the record (CUST-UPDATE-RECORD); the fields line up with this layout, so the field-by-field table shows only the import job as setting them.
- The same layout is also defined in copybook CUSTREC; keep one definition in the target.
- 168 bytes of the 500-byte record are unused FILLER.

Field	PIC	Bytes	Values	Set by
CUSTOMER-RECORD		1-500		
CUST-ID	9(09)	1-9		CBIMPORT (CBIMPORT)
CUST-FIRST-NAME	X(25)	10-34		CBIMPORT (CBIMPORT)
CUST-MIDDLE-NAME	X(25)	35-59		CBIMPORT (CBIMPORT)
CUST-LAST-NAME	X(25)	60-84		CBIMPORT (CBIMPORT)
CUST-ADDR-LINE-1	X(50)	85-134		CBIMPORT (CBIMPORT)
CUST-ADDR-LINE-2	X(50)	135-184		CBIMPORT (CBIMPORT)
CUST-ADDR-LINE-3	X(50)	185-234		CBIMPORT (CBIMPORT)
CUST-ADDR-STATE-CD	X(02)	235-236		CBIMPORT (CBIMPORT)
CUST-ADDR-COUNTRY-CD	X(03)	237-239		CBIMPORT (CBIMPORT)
CUST-ADDR-ZIP	X(10)	240-249		CBIMPORT (CBIMPORT)
CUST-PHONE-NUM-1	X(15)	250-264		CBIMPORT (CBIMPORT)
CUST-PHONE-NUM-2	X(15)	265-279		CBIMPORT (CBIMPORT)
CUST-SSN	9(09)	280-288		CBIMPORT (CBIMPORT)
CUST-GOVT-ISSUED-ID	X(20)	289-308		CBIMPORT (CBIMPORT)
CUST-DOB-YYYY-MM-DD	X(10)	309-318		CBIMPORT (CBIMPORT)
CUST-EFT-ACCOUNT-ID	X(10)	319-328		CBIMPORT (CBIMPORT)
CUST-PRI-CARD-HOLDER-IND	X(01)	329-329		CBIMPORT (CBIMPORT)

CUST-FICO-CREDIT-SCORE	9(03)	330-332		CBIMPORT (CBIMPORT)

CVEXPORT: EXPORT-RECORD

- Stored in AWS.M2.CARDDEMO.EXPORT.DATA: read by CBIMPORT; written by CBEXPORT

Field	PIC	Bytes	Values	Set by
EXPORT-RECORD		1-500		
EXPORT-REC-TYPE	X(1)	1-1		CBEXPORT (CBEXPORT)
EXPORT-TIMESTAMP	X(26)	2-27		CBEXPORT (CBEXPORT)
EXPORT-TIMESTAMP-R		2-27		
EXPORT-DATE	X(10)	2-11		
EXPORT-DATE-TIME-SEP	X(1)	12-12		
EXPORT-TIME	X(15)	13-27		
EXPORT-SEQUENCE-NUM	9(9)	28-31		CBEXPORT (CBEXPORT)
EXPORT-BRANCH-ID	X(4)	32-35		CBEXPORT (CBEXPORT)
EXPORT-REGION-CODE	X(5)	36-40		CBEXPORT (CBEXPORT)
EXPORT-RECORD-DATA	X(460)	41-500		
EXPORT-CUSTOMER-DATA		41-500		
EXP-CUST-ID	9(09)	41-44		CBEXPORT (CBEXPORT)
EXP-CUST-FIRST-NAME	X(25)	45-69		CBEXPORT (CBEXPORT)
EXP-CUST-MIDDLE-NAME	X(25)	70-94		CBEXPORT (CBEXPORT)
EXP-CUST-LAST-NAME	X(25)	95-119		CBEXPORT (CBEXPORT)
EXP-CUST-ADDR-LINES		120-269		
EXP-CUST-ADDR-LINE	X(50)	120-169		CBEXPORT (CBEXPORT)
EXP-CUST-ADDR-STATE-CD	X(02)	270-271		CBEXPORT (CBEXPORT)
EXP-CUST-ADDR-COUNTRY-CD	X(03)	272-274		CBEXPORT (CBEXPORT)
EXP-CUST-ADDR-ZIP	X(10)	275-284		CBEXPORT (CBEXPORT)
EXP-CUST-PHONE-NUMS		285-314		
EXP-CUST-PHONE-NUM	X(15)	285-299		CBEXPORT (CBEXPORT)
EXP-CUST-SSN	9(09)	315-323		CBEXPORT (CBEXPORT)
EXP-CUST-GOVT-ISSUED-ID	X(20)	324-343		CBEXPORT (CBEXPORT)
EXP-CUST-DOB-YYYY-MM-DD	X(10)	344-353		CBEXPORT (CBEXPORT)
EXP-CUST-EFT-ACCOUNT-ID	X(10)	354-363		CBEXPORT (CBEXPORT)
EXP-CUST-PRI-CARD-HOLDER-IND	X(01)	364-364		CBEXPORT (CBEXPORT)
EXP-CUST-FICO-CREDIT-SCORE	9(03)	365-366		CBEXPORT (CBEXPORT)
EXPORT-ACCOUNT-DATA		41-500		
EXP-ACCT-ID	9(11)	41-51		CBEXPORT (CBEXPORT)
EXP-ACCT-ACTIVE-STATUS	X(01)	52-52		CBEXPORT (CBEXPORT)
EXP-ACCT-CURR-BAL	S9(10)V99	53-59		CBEXPORT (CBEXPORT)
EXP-ACCT-CREDIT-LIMIT	S9(10)V99	60-71		CBEXPORT (CBEXPORT)
EXP-ACCT-CASH-CREDIT-LIMIT	S9(10)V99	72-78		CBEXPORT (CBEXPORT)
EXP-ACCT-OPEN-DATE	X(10)	79-88		CBEXPORT (CBEXPORT)
EXP-ACCT-EXPIRAION-DATE	X(10)	89-98		CBEXPORT (CBEXPORT)
EXP-ACCT-REISSUE-DATE	X(10)	99-108		CBEXPORT (CBEXPORT)
EXP-ACCT-CURR-CYC-CREDIT	S9(10)V99	109-120		CBEXPORT (CBEXPORT)
EXP-ACCT-CURR-CYC-DEBIT	S9(10)V99	121-128		CBEXPORT (CBEXPORT)

EXP-ACCT-ADDR-ZIP	X(10)	129-138	CBEXPORT (CBEXPORT)
EXP-ACCT-GROUP-ID	X(10)	139-148	CBEXPORT (CBEXPORT)
EXPORT-TRANSACTION-DATA		41-500	
EXP-TRAN-ID	X(16)	41-56	CBEXPORT (CBEXPORT)
EXP-TRAN-TYPE-CD	X(02)	57-58	CBEXPORT (CBEXPORT)
EXP-TRAN-CAT-CD	9(04)	59-62	CBEXPORT (CBEXPORT)
EXP-TRAN-SOURCE	X(10)	63-72	CBEXPORT (CBEXPORT)
EXP-TRAN-DESC	X(100)	73-172	CBEXPORT (CBEXPORT)
EXP-TRAN-AMT	S9(09)V99	173-178	CBEXPORT (CBEXPORT)
EXP-TRAN-MERCHANT-ID	9(09)	179-182	CBEXPORT (CBEXPORT)
EXP-TRAN-MERCHANT-NAME	X(50)	183-232	CBEXPORT (CBEXPORT)
EXP-TRAN-MERCHANT-CITY	X(50)	233-282	CBEXPORT (CBEXPORT)
EXP-TRAN-MERCHANT-ZIP	X(10)	283-292	CBEXPORT (CBEXPORT)
EXP-TRAN-CARD-NUM	X(16)	293-308	CBEXPORT (CBEXPORT)
EXP-TRAN-ORIG-TS	X(26)	309-334	CBEXPORT (CBEXPORT)
EXP-TRAN-PROC-TS	X(26)	335-360	CBEXPORT (CBEXPORT)
EXPORT-CARD-XREF-DATA		41-500	
EXP-XREF-CARD-NUM	X(16)	41-56	CBEXPORT (CBEXPORT)
EXP-XREF-CUST-ID	9(09)	57-65	CBEXPORT (CBEXPORT)
EXP-XREF-ACCT-ID	9(11)	66-73	CBEXPORT (CBEXPORT)
EXPORT-CARD-DATA		41-500	
EXP-CARD-NUM	X(16)	41-56	CBEXPORT (CBEXPORT)
EXP-CARD-ACCT-ID	9(11)	57-64	CBEXPORT (CBEXPORT)
EXP-CARD-CVV-CD	9(03)	65-66	CBEXPORT (CBEXPORT)
EXP-CARD-EMBOSSED-NAME	X(50)	67-116	CBEXPORT (CBEXPORT)
EXP-CARD-EXPIRAION-DATE	X(10)	117-126	CBEXPORT (CBEXPORT)
EXP-CARD-ACTIVE-STATUS	X(01)	127-127	CBEXPORT (CBEXPORT)

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)

CVTRA02Y: DIS-GROUP-RECORD

- Stored in AWS.M2.CARDDEMO.DISCGRP.VSAM.KSDS: looked up by CBACT04C

Field	PIC	Bytes	Values	Set by
DIS-GROUP-RECORD		1-50		
DIS-GROUP-KEY		1-16		
DIS-ACCT-GROUP-ID	X(10)	1-10		

DIS-TRAN-TYPE-CD	X(02)	11-12		
DIS-TRAN-CAT-CD	9(04)	13-16		
DIS-INT-RATE	S9(04)V99	17-22		

CVTRA03Y: TRAN-TYPE-RECORD

- Stored in AWS.M2.CARDDEMO.TRANTYPE.VSAM.KSDS: looked up by CBTRN03C

Field	PIC	Bytes	Values	Set by
TRAN-TYPE-RECORD		1-60		
TRAN-TYPE	X(02)	1-2		
TRAN-TYPE-DESC	X(50)	3-52		

CVTRA04Y: TRAN-CAT-RECORD

- Stored in AWS.M2.CARDDEMO.TRANCATG.VSAM.KSDS: looked up by CBTRN03C

Field	PIC	Bytes	Values	Set by
TRAN-CAT-RECORD		1-60		
TRAN-CAT-KEY		1-6		
TRAN-TYPE-CD	X(02)	1-2		
TRAN-CAT-CD	9(04)	3-6		
TRAN-CAT-TYPE-DESC	X(50)	7-56		

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		

CVTRA07Y: REPORT-ACCOUNT-TOTALS, REPORT-GRAND-TOTALS, REPORT-NAME-HEADER, REPORT-PAGE-TOTALS, TRANSACTION-DETAIL-REPORT, TRANSACTION-HEADER-1

- Stored in AWS.M2.CARDDEMO.TRANREPT: written by CBTRN03C

Field	PIC	Bytes	Values	Set by
REPORT-ACCOUNT-TOTALS		1-112		
REPT-ACCOUNT-TOTAL	+ZZZ,ZZZ,ZZZ.ZZ	98-112		CBTRN03C (TRANREPT)
REPORT-GRAND-TOTALS		1-112		
REPT-GRAND-TOTAL	+ZZZ,ZZZ,ZZZ.ZZ	98-112		CBTRN03C (TRANREPT)
REPORT-NAME-HEADER		1-115		
REPT-SHORT-NAME	X(38)	1-38		
REPT-LONG-NAME	X(41)	39-79		
REPT-DATE-HEADER	X(12)	80-91		
REPT-START-DATE	X(10)	92-101		CBTRN03C (TRANREPT)
REPT-END-DATE	X(10)	106-115		CBTRN03C (TRANREPT)
REPORT-PAGE-TOTALS		1-112		
REPT-PAGE-TOTAL	+ZZZ,ZZZ,ZZZ.ZZ	98-112		CBTRN03C (TRANREPT)

TRANSACTION-DETAIL-REPORT		1-114	
TRAN-REPORT-TRANS-ID	X(16)	1-16	CBTRN03C (TRANREPT)
TRAN-REPORT-ACCOUNT-ID	X(11)	18-28	CBTRN03C (TRANREPT)
TRAN-REPORT-TYPE-CD	X(02)	30-31	
TRAN-REPORT-TYPE-DESC	X(15)	33-47	CBTRN03C (TRANREPT)
TRAN-REPORT-CAT-CD	9(04)	49-52	
TRAN-REPORT-CAT-DESC	X(29)	54-82	CBTRN03C (TRANREPT)
TRAN-REPORT-SOURCE	X(10)	84-93	CBTRN03C (TRANREPT)
TRAN-REPORT-AMT	-ZZZ,ZZZ,ZZZ.ZZ	98-112	CBTRN03C (TRANREPT)
TRANSACTION-HEADER-1		1-114	

DCLTRTYP: DCLTRANSACTION-TYPE

- Stored in table CARDDEMO.TRANSACTION_TYPE: deleted by COBTUPDT, COTRTLIC, COTRTUPC; inserted by COBTUPDT, COTRTUPC; read by COTRTLIC, COTRTUPC; updated by COBTUPDT, COTRTLIC, COTRTUPC

Field	PIC	Bytes	Values	Set by
DCLTRANSACTION-TYPE		1-54		
DCL-TR-TYPE	X(2)	1-2		COTRTLIC (transaction CTLI), COTRTUPC (transaction CTTU)
DCL-TR-DESCRIPTION		3-54		COTRTLIC (transaction CTLI), COTRTUPC (transaction CTTU)
DCL-TR-DESCRIPTION-LEN	S9(4)	3-4		COTRTLIC (transaction CTLI), COTRTUPC (transaction CTTU)
DCL-TR-DESCRIPTION-TEXT	X(50)	5-54		

Findings

▲ Serious **1 member(s) referenced but not in the source**

These are called, run, or copied but weren't in any library provided. Ask the client for them before estimating the work that depends on them.

- PROGRAM COCRDSEC - needed by transaction CDV1

▲ Serious **1 program(s) use ALTER**

ALTER changes where a GO TO goes while the program runs, so the code doesn't show the real flow. Untangle it before converting.

- CBSTM03A: 4 ALTER

▲ Serious **1 file(s) a program uses that its job doesn't define**

The program opens a DD name that no step running it defines; the job would fail or the file comes from somewhere not in the source.

- CBIMPORT DD CARDOUT

● Check **COACTUPC writes AWS.M2.CARDDEMO.ACCTDATA.VSAM.KSDS back from a different structure**

COACTUPC reads the record into one structure and writes it back from another; they don't line up field for field, so the record is written back with these bytes holding something else, or blank:

- read into ACCOUNT-RECORD (CVACT01Y, 300 bytes)
- written from ACCT-UPDATE-RECORD (COACTUPC, 300 bytes)
- bytes 103-112: read as ACCT-ADDR-ZIP, written from ACCT-UPDATE-GROUP-ID
- bytes 113-122: read as ACCT-GROUP-ID, written from FILLER

● Check **1 program(s) with NEXT SENTENCE inside an END-IF**

NEXT SENTENCE jumps past the END-IF to the next period, skipping statements after the IF that the indentation suggests will run. Check each one against the intended logic.

- CBTRN03C: 1

● 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)

---

i Note **Record CARD-RECORD coded inline in COCRDUPC**

1 inline layout(s) have the same fields as CARD-RECORD in copybook CVACT02Y: COPY CVACT02Y instead, so a change to the record is made once.

- CARD-UPDATE-RECORD (COCRDUPC, 150 bytes)

Note **Record CUSTOMER-RECORD coded inline in COACTUPC**

1 inline layout(s) have the same fields as CUSTOMER-RECORD in copybook CVCUS01Y (also in CUSTREC): COPY CVCUS01Y instead, so a change to the record is made once.

- CUST-UPDATE-RECORD (COACTUPC, 500 bytes)

Note **Record INFIL2-REC coded inline in PAUDBLOD, PAUDBUNL**

The same 206-byte record is coded inline in 2 programs: extract it into one copybook, so a change to the record is made once.

- INFIL2-REC (PAUDBLOD, 206 bytes)
- OPFIL2-REC (PAUDBUNL, 206 bytes)

Note **COACTUPC writes AWS.M2.CARDDEMO.CUSTDATA.VSAM.KSDS back from a different structure**

COACTUPC reads the record into one structure and writes it back from another; the fields line up, so it's only a naming difference to tidy up.

- read into CUSTOMER-RECORD (CVCUS01Y, 500 bytes)
- written from CUST-UPDATE-RECORD (COACTUPC, 500 bytes)

Note **COCRDUPC writes AWS.M2.CARDDEMO.CARDDATA.VSAM.KSDS back from a different structure**

COCRDUPC reads the record into one structure and writes it back from another; the fields line up, so it's only a naming difference to tidy up.

- read into CARD-RECORD (CVACT02Y, 150 bytes)
- written from CARD-UPDATE-RECORD (COCRDUPC, 150 bytes)

Note **10 program(s) use GO TO**

Jumps make the flow harder to follow and to convert to structured code.

- COACTUPC: 66
- COTRTLIC: 28
- COTRTUPC: 23
- COCRDUPC: 21
- COCRDLIC: 16
- CBSTM03A: 14
- CBSTM03B: 13
- COACTVWC: 9
- COCRDSLC: 9
- CORPT00C: 1

Note **5 program(s) with paragraphs nothing reaches**

No PERFORM, GO TO, or fall-through reaches these paragraphs: candidates to remove rather than convert (check for a reference the source doesn't show first).

- COACTUPC: 1230-EDIT-ALPHANUM-REQD, 1240-EDIT-ALPHANUM-OPT
- COACTVWC: SEND-LONG-TEXT
- COCRDLIC: SEND-PLAIN-TEXT, SEND-LONG-TEXT
- COCRDSLC: 9150-GETCARD-BYACCT, SEND-LONG-TEXT
- COTRTLIC: SEND-PLAIN-TEXT

Note 1 program(s) nothing in the source runs or calls

No JCL step, CSD transaction, or CALL / XCTL / LINK reaches these. They may be retired, or run from somewhere not provided (another application, a scheduler, an IMS or CICS definition); 11 dynamic call(s) couldn't be resolved and might reach one.

- CBTRN01C

Note 4 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)
- PRTCATBL reads what TCATBALF writes, but they're in different flows (schedule ID 032, schedule ID 031): the order depends on when each flow runs - confirm it with the client. (AWS.M2.CARDDEMO.TCATBALF.VSAM.KSDS)
- INTCALC reads what DISCGRP writes, but they're in different flows (WEEKLY-DisclosureGroupsRefresh, MONTHLY-InterestCalculation): the order depends on when each flow runs (WEEKLY-DisclosureGroupsRefresh runs until 23:00; MONTHLY-InterestCalculation runs until 23:00). (AWS.M2.CARDDEMO.DISCGRP.VSAM.KSDS)

Note 3 scheduled order(s) with no shared data behind them

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

- In schedule ID 030, READCARD runs after READACCT, but they share no dataset or Db2 table: the order may be for something the source doesn't show (a resource, a business rule), or they could run in parallel.
- In schedule ID 030, READCUST runs after READCARD, but they share no dataset or Db2 table: the order may be for something the source doesn't show (a resource, a business rule), or they could run in parallel.
- In schedule ID 030, READXREF runs after READCUST, but they share no dataset or Db2 table: the order may be for something the source doesn't show (a resource, a business rule), or they could run in parallel.

Note 1 copybook(s) nothing uses

Candidates to retire rather than convert.

- UNUSED1Y

Note From the review of job CREASTMT

- **Only works for demo-sized data:** the statement program loads every transaction into a table of 51 cards x 10 transactions per card with no overflow check, so a realistic number of cards or transactions corrupts memory rather than failing cleanly.
- **No statement period:** every transaction ever posted to the master appears on the statement; there's no cycle start and end date. Real statements need one - confirm the intended cycle with the business.
- **One statement per card,** not per account: a customer with two cards on one account gets two statements, each showing the full account balance.
- The re-keying steps (delete, define, sort, load) exist only to read transactions in card order; with a database, a query ordered by card replaces steps 1-3.
- The statement program uses z/OS-specific control-block access and ALTER/GO TO (deliberately, to exercise modernization tools); both are removed in a rewrite. Statement layout and HTML generation map naturally to a templating or document-generation tool.

Note **From the review of job INTCALC**

- **The last account in the file is never updated:** its interest transactions are written but the interest isn't added to its balance and its cycle isn't reset (a loop branch the program can't reach) - the transactions and balances disagree for that account every month.
- **Interest is charged on running totals that nothing resets:** the category balances are only ever added to by posting; neither this job nor the posting job clears them after interest is charged, so each month's interest is calculated on all activity to date, not the month's balance. Confirm with the business how the balance subject to interest should be defined - this is a likely defect, not just a modernization detail.
- **Hard-coded run date** in the JCL; on a modern scheduler this is the business date passed in automatically.
- Fees are a placeholder in the program (not implemented).
- Interest posting is split across three jobs (INTCALC writes, COMBTRAN merges and reloads the master); a modern design would post the interest directly, in the same database transaction as the balance update.

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 job TRANREPT**

- **Two sources for the date range.** The SORT's dates are hard-coded in the JCL while the program reads DATEPARM; if they differ, the report heading can show one range while the lines come from another. The program's own date check doesn't work as written (an out-of-range record would end the report early), so today only the SORT's dates actually select the data.
- **Totals are slightly off at the end:** the last transaction is counted twice in the final page and grand totals, and the last card's total line is missing.
- The first two steps (unload, filter, sort) and the report program together amount to "select transactions by processing date, ordered by card, with lookups and subtotals" - a single query and a reporting tool on a modern platform.
- The full unload of the VSAM master on every run is mainly there to feed the SORT; a database-backed version wouldn't need it, but COMBTRAN also reads the backup generation, so that dependency has to move with it.

Note **From the review of program CBACT04C**

- **The last account is never updated.** The update for the final account sits in a branch the read loop can't reach (the loop ends as soon as end of file is set), so the last account in the category balance file gets neither its interest added to the balance nor its cycle totals reset - its interest transactions are written, though, so the transactions and the balance disagree.
- **Interest is charged on the category balance as it stands,** including negative (credit) balances, which produce negative interest; confirm with the business whether credits should earn or be excluded.
- **Cycle close is a side effect** of the interest run (resetting the cycle totals); a modern design would make the cycle close an explicit step.
- Fees are not implemented (1400-COMPUTE-FEES is empty).
- The account's card number comes from the first cross-reference record for the account; an account with several cards gets its interest transactions under one of them.
- The interest transactions go to a separate file (SYSTRAN) rather than the transaction master; another job has to post or merge them - see the job flow.
- The open-error message for the disclosure group file says "DALY REJECTS FILE" (copy-and-paste slip).

Note **From the review of program CBSTM03A**

- **Fixed in-memory limits with no checks:** the transaction table holds 51 cards and 10 transactions per card. More cards or transactions than that overwrite other storage instead of failing cleanly - the program only works for demo-sized data. A modern version would read transactions per card on demand (a query by card) instead of loading them all.
- **z/OS-only code:** reading the PSA, TCB, and TIOT control blocks to list DD names has no equivalent off the mainframe; it's diagnostic only and can be dropped.
- **ALTER and GO TO:** the file-opening sequence is a self-modifying jump chain; rewrite it as straight-line opens.
- **No statement period:** statements list all transactions on file. Real statements need a cycle (from-to dates), which the job and program don't have - a gap to raise with the business.
- The HTML is assembled line by line from fixed text; a templating tool replaces most of this program.
- Depends on the cross-reference and the transaction file both being in card-number order; the matching stops looking once the table passes the current card.

Note **From the review of program CBSTM03B**

- A hand-written data-access layer: on a modern platform this becomes ordinary database queries or a repository class, and the subroutine goes away.
- Unsupported combinations (for example a keyed read on the transaction file, or any write) silently do nothing and return the previous status - callers must not rely on them.
- Uses GO TO between paragraphs for control flow; straightforward to restructure.

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.

Note **From the review of program CBTRN03C**

- **The date filter doesn't work as written.** An out-of-range transaction executes `NEXT SENTENCE`, which jumps past the end of the whole read loop, so the first out-of-range record would end the report early (the remaining transactions and the final totals are skipped). It works in practice only because the job's SORT step already keeps just the dates in range - so the range lives in two places that must agree: SYMNAMES constants in the TRANREPT JCL (hard-coded 2022-01-01 to 2022-07-06) and the DATEPARM file.
- **Totals at end of file are off:** the last transaction's amount is added to the page and card totals a second time, and the last card's "Account Total" line is never written.
- "Account Total" is really a card total (the break is on card number, not account).
- Every transaction record is DISPLAYed, which can make SYSOUT very large.
- Depends on the input being sorted by card number; on a modern platform this becomes an ORDER BY or a sort in the reporting tool.
- Simple, self-contained report logic: a good candidate to replace with a query plus a reporting tool, once the two date-range sources are reconciled.

Note **From the review of program COACTUPC**

- **Any user can change balances and limits from a screen:** current balance, cycle credit and debit totals, and credit limits are editable with no extra authorization (and the main menu offers this screen to all users). Financial fields like the balance normally change only through posted transactions; decide with the business which fields should be editable and by whom.
- **Leaving the screen with PF3 sets the user type to "regular user"** in the communication area, so an administrator passing through loses admin status for the rest of the session.
- **Well-designed update logic worth keeping:** locking, the "changed by someone else" check, and the rollback map directly onto a database transaction with optimistic concurrency.
- **Most of the 3,400 lines of code are field validation and screen attribute handling,** written out per field. A modern form with a validation library (and a shared US address/phone/SSN validator) replaces most of it; the business rules above are the part to carry over.
- One screen updates two entities (account and customer); as an API this naturally becomes two resources, or one "account with holder" operation that keeps the single-transaction save.
- Uses the CSUTLDTC routine (IBM CEEDAYS) for date checks - replace with the target language's date handling.

Note **From the review of program COMEN01C**

- **Authorization is effectively off:** the admin-only mechanism exists but no option uses it. Decide with the business which functions (adding transactions, account and card updates) should be restricted, and enforce it in the services rather than in the menu.
- **An invalid option number is still used to index the options table** before the error takes effect (zero or above 11 reads outside the table); harmless in practice because the error stops the transfer, but sloppy.
- The copybook holding these options is headed "Admin Menu Options" although it's the regular user menu.
- A menu is navigation only: in a web front end it becomes the application's navigation, and this program is retired rather than converted.

Note **From the review of program COSGN00C**

- **Passwords are stored in clear text** in the user security file (8 characters) and compared as plain text - a security finding in its own right; replace with the target platform's identity provider or at least salted hashes.
- **No lockout or attempt limit,** and the messages tell a caller whether a user ID exists ("User not found" vs. "Wrong Password"), which helps guessing.
- Passwords are effectively case-insensitive and limited to 8 characters.
- The whole application relies on this program filling in the communication area (user ID and type); every other screen trusts it. A modern front end would use a session or token from a standard sign-in instead.
- A good candidate to retire rather than convert: sign-on becomes the target platform's single sign-on, and the user file becomes identity data.

Note **From the review of program COTRN02C**

- **Added transactions don't update balances:** the transaction goes straight onto the master file without the posting job's checks (card and account status, credit limit, expiration) and without updating the account balance or the category balances. Interest and statements will disagree with the balance for any account with manually added transactions - confirm with the business whether this screen is meant for adjustments that are posted some other way.
- **Transaction IDs can collide:** "read the highest ID, add 1, write" isn't safe with two users adding at the same time - one of them gets "Tran ID already exist". Use a database sequence or identity column.
- The user types in the processing date as well as the original date; normally the system sets the processing date.
- Open to every user from the main menu (its "Admin Only" label is commented out).
- Field-by-field validation with one error at a time maps to a standard form with validation; the "add a transaction" logic itself belongs in the same service as batch posting, so both apply the same rules.

Note **From the review of program CSUTLDTC**

- **Depends on an IBM Language Environment service (CEEDAYS)** that doesn't exist off the mainframe; replace it with the target language's date parsing (one line in Java, C#, or Python) - this routine then disappears.
- Callers check the return code or the message text; when replacing it, make sure the callers' checks still mean the same thing (severity 0 = valid).

Note **From the review of transaction CAUP**

- **Balances and limits can be typed in:** any user can change the current balance, cycle totals, and credit limits directly, bypassing posting - a control gap to raise with the client, and a rule to decide before building the new screen.
- **Strong update pattern to keep:** confirmation, locking, the "changed by someone else" check, and the joint save map directly onto a database transaction with optimistic concurrency.
- The business rules (state/ZIP check, SSN rules, FICO range, phone format) are worth extracting into a shared validation service; most of the program's 3,400 lines are screen handling that a web form replaces.
- **Expose as API:** "get account with holder" and "update account with holder" are natural service operations behind a new web screen, with the batch jobs above reading the same data.
- Leaving with PF3 resets the session's user type to regular user (see COACTUPC).

Note **From the review of transaction CC00**

- **Passwords are kept and compared in clear text,** are case-insensitive, and are at most 8 characters; there's no lockout, and the messages reveal whether a user ID exists. Treat these as security findings for the client, separate from modernization.
- On a modern platform sign-on is replaced by the organization's identity provider (single sign-on), and the user security file becomes user and role data there - this transaction is retired rather than converted.
- Every other screen trusts the role set here; the target design should check authorization in the services, not only at sign-on.

Note **From the review of transaction CM00**

- **Every option is open to every user:** the menu supports admin-only options, but none is marked admin-only (Transaction Add's "Admin Only" label is commented out), so regular users can add transactions and update accounts, balances, and limits. Agree the intended permissions with the business and enforce them in the services.
- The menu itself is navigation; in a web front end it becomes the application's navigation bar, and this transaction is retired.
- The options table in a copybook is a simple, readable source for the target application's menu and permissions.

Note **From the review of transaction CT02**

- **Manually added transactions skip posting:** they aren't checked against the account (status, credit limit, expiration) and don't update the account balance or category balances, so balances, interest, and statements won't reflect them. Confirm with the business what this screen is for (adjustments? testing?) and route it through the same posting logic as the daily batch.
- **Next-ID-plus-one numbering** can collide when two users add at once; use a database sequence.
- Type and category codes aren't checked against the reference files, and the processing date is typed in rather than set by the system.
- Available to all users (the menu's "Admin Only" restriction is commented out).
- As an API this becomes "post a transaction", shared with batch posting so online and batch apply the same rules.

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.