DATA CONTROL TOWER · GUIDE

FTC Safeguards Rule two-year disposal evidence for Snowflake, Databricks and Redshift

A free script that lists the customers past two years with rows left, the exceptions, the deleted data still recoverable and the retention policy reviews, from Snowflake, Databricks and Redshift records.

Published 2026-11-01 · Online version and downloads: www.datameaning.com/guides/ftc-safeguards-rule-two-year-disposal-evidence-for-snowflake-databricks-redshift/

Count the rows each customer still has and compare that with the date you last served them. We count the Safeguards Rule's two years per customer, not per table. Our free script lists customers past two years with rows left, exceptions, deleted data still recoverable, and overdue policy reviews. It gives evidence, not a verdict. Demo records are generated.

The rule, in its own words

The Safeguards Rule is 16 CFR part 314. The text below is public: we read it on 2026-11-01 through the eCFR's own API, current as of 2026-10-29, and link the eCFR page for part 314 (read 2026-11-01). We could not read the eCFR website itself by script, so the page is linked for you to check against. This is not legal advice. Whether your company is a financial institution the FTC has jurisdiction over is a question for the scope in 314.1(b) and the definitions in 314.2, and this page does not answer it.

314.4(c)(6)(i) says a financial institution must:

“Develop, implement, and maintain procedures for the secure disposal of customer information in any format no later than two years after the last date the information is used in connection with the provision of a product or service to the customer to which it relates, unless such information is necessary for business operations or for other legitimate business purposes, is otherwise required to be retained by law or regulation, or where targeted disposal is not reasonably feasible due to the manner in which the information is maintained” (eCFR: 16 CFR 314.4 (read 2026-11-01)).

314.4(c)(6)(ii) adds: “Periodically review your data retention policy to minimize the unnecessary retention of data”. **314.2(d)** defines customer information as “any record containing nonpublic personal information about a customer of a financial institution, whether in paper, electronic, or other form, that is handled or maintained by or on behalf of you or your affiliates”.

Three more facts from the same part. 314.6 lists what an institution with customer information on fewer than five thousand consumers is exempt from: 314.4(b)(1), (d)(2), (h) and (i). Paragraph (c)(6) is not on that list. And 314.5, as it read on 1 January 2023, gave June 9, 2023 as the effective date of 314.4(c)(1) through (8). Today's 314.5 gives only the effective date of 314.4(j), May 13, 2024.

What the text fixes, what we chose, and what it leaves to you

- **We count the clock per customer.** The rule starts it at the last date the information is used to serve the customer it relates to. Your system of record holds that date for each customer, so the script takes one last-use date per customer. If you track use per record or per table, whether an old row counts as still in use is a question for your lawyer.
- **The rule sets two years but not how to count them.** We count to the same calendar date two years on, so 29 February 2024 ends on 28 February 2026, the earlier reading; the deadline day itself is not past. That counting is ours, not the rule's.
- **The rule gives three reasons for keeping information.** Business operations or other legitimate business purposes; required to be retained by law or regulation; targeted disposal not reasonably feasible. What counts as a business need is yours to write down. The script never judges it.
- **"Periodically" has no interval.** So the script has no default for how often the retention policy is reviewed: you set `RETENTION_REVIEW_DAYS`, or that list does not run.

Why a platform alone cannot answer it

No platform knows when you last served a customer. That date sits in your own system of record. Deleting the rows is also not the end of the story. Here is what each platform keeps after a delete, from its own documentation:

- **Snowflake.** Time Travel keeps historical data for the table's retention period: for permanent tables on Enterprise Edition and higher, “any value from 0 up to 90 days” (Snowflake: Time Travel (read 2026-11-01)). If your account sets `MIN_DATA_RETENTION_TIME_IN_DAYS`, “the effective minimum data retention period for an object is determined by MAX(DATA_RETENTION_TIME_IN_DAYS, MIN_DATA_RETENTION_TIME_IN_DAYS)” (Snowflake: Time Travel (read 2026-11-01)); the TABLES page does not say whether RETENTION_TIME reflects it, so check `SHOW PARAMETERS LIKE 'MIN_DATA_RETENTION_TIME_IN_DAYS' IN ACCOUNT` and set `SNOWFLAKE_MIN_DATA_RETENTION_DAYS`. Then Fail-safe, “a (non-configurable) 7-day period during which historical data may be recoverable by Snowflake” (Snowflake: Fail-safe (read

2026-11-01)). “Transient and temporary tables have no Fail-safe period” (Snowflake: temporary and transient tables (read 2026-11-01)).

- **Databricks.** “The default retention threshold for vacuum is 7 days” (Databricks: Remove unused data files with VACUUM (read 2026-11-01)). Old files from a delete stay in storage until a VACUUM runs after that threshold, and the threshold is the table property `delta.deletedFileRetentionDuration`, 7 days by default (Databricks: VACUUM (SQL) (read 2026-11-01)). On a table with deletion vectors, a DELETE only marks rows: “Some table features, such as deletion vectors, use metadata files to mark data as deleted rather than rewriting data files”, so their data stays in the table's current files until `REORG TABLE ... APPLY (PURGE)` rewrites them, and a later VACUUM removes the old files (Databricks: Remove unused data files with VACUUM (read 2026-11-01)). “Table history retention is determined by the table setting logRetentionDuration, which is 30 days by default”, so export `DESCRIBE HISTORY` more often than that (Databricks: Work with table history (read 2026-11-01)).
- **Redshift.** “By default, manual snapshots are retained indefinitely, even after you delete your cluster”; automated snapshots are deleted at the end of a retention period whose default is one day (Amazon Redshift: snapshots and backups (read 2026-11-01)). A snapshot “contains a copy of the cluster data as of this exact time” (AWS CLI: describe-cluster-snapshots (read 2026-11-01)), so one taken before a delete still holds the rows.

Our sibling guide on ISO 27001 deletion and masking evidence goes into those recovery windows in detail. This page uses them for one question: how long after the two-year date can a disposed record still come back.

What you will have at the end

A Python script, safeguards_disposal_evidence.py, that reads five files you keep or export from your own systems, plus each platform's own recovery settings, all as CSV. It runs 27 input checks first (fewer if you leave a platform out), then prints five lists (1, 1b, 2, 3 and 4), each with its filters, its row count and a SHA-256 of its rows, and writes each as a CSV. The details are also in a PDF: Download the full guide (PDF).

1. **Customers past two years.** Every customer whose last date of use is more than two years before the as-of day and who still has rows in any table you registered, with the rows per table. It is PAST TWO YEARS unless an exception you recorded is in force. An open relationship with an old last-use date, or no last-use date at all, is LAST USE UNCLEAR and is never cleared. A customer in the platform rows but not in your register is NOT IN REGISTER. A second list shows every registered table and says NOT MEASURED where the script could not read it by customer.

2. **Exceptions.** Each exception, its reason, and whether it is in force: NO BASIS, EXPIRED and NOT REVIEWED (past its own review date) are flags. An exception with a flag does not count as in force.
3. **Disposed but still recoverable.** Per table, how many days a row deleted on the as-of day can still be brought back on its platform. A count to look at, not a finding, except INDEFINITE (a Redshift manual snapshot kept until someone deletes it), DELETION VECTORS, NO PURGE IN HISTORY (a Databricks table whose deleted rows stay until a purge nobody has run) and NOT MEASURED.
4. **Retention policy reviews.** The review dates on record, the days between them, and OVERDUE against the gap you set.

The records we ran it on are **generated**: a seeded stand-in, not a company's data and not a measurement of anything. We have not run the script on a real Snowflake, Databricks or Redshift account. It produces evidence; it does not say the rule is followed, and your lawyer or assessor decides what is enough.

What you will need

- Your system of record: a list of customers with the last date each one's information was used to serve them, and whether the relationship is still open. The script cannot work this out for you.
- A table register: every table on each platform that holds customer information, and the column that holds the customer id. A table with no such column cannot be disposed of customer by customer: it needs an exception for "targeted disposal is not reasonably feasible", or it is listed NOT MEASURED. Table names are written `database.schema.table` on Snowflake and Databricks, and `schema.table` on Redshift.
- Snowflake: `ACCOUNT_USAGE.TABLES` (RETENTION_TIME, IS_TRANSIENT, DELETED), which can lag by up to 90 minutes (Snowflake: ACCOUNT_USAGE.TABLES (read 2026-11-01)); the script reads only the live rows of tables in your register, so views and dropped tables in the export do no harm. Databricks: each table's `deletedFileRetentionDuration` property (written as a number of days: `interval 30 days` is 30), whether it has deletion vectors on (`delta.enableDeletionVectors`, true or false), and `DESCRIBE HISTORY` for the last VACUUM and the last REORG (Databricks: Table history schema (read 2026-11-01)). Redshift: `aws redshift describe-cluster-snapshots` (AWS CLI: describe-cluster-snapshots (read 2026-11-01)).
- Python 3 and the `duckdb` module (we ran it with duckdb 1.5.6 and 2.0.0).

Step 1. Write your registers and export the platforms' records

The script reads these nine files from `EXPORT_DIR`. Column names are the platforms' own where a platform writes them; the script lower-cases the header, so `RETENTION_TIME` and `retention_time` both work, and a missing column fails an input check that names it. Table names are compared whole and without case or double quotes; customer ids are compared exactly, after trimming spaces. Dates are strict `YYYY-MM-DD`; timestamps are read as `YYYY-MM-DD HH:MM:SS` in UTC. A value that cannot be read is counted by an input check, never skipped.

The nine files

File	Columns the script reads	Where it comes from
`customer_register.csv`	customer_id, last_use_date, relationship_open (true or false)	Your system of record
`table_register.csv`	platform, table_name, customer_key_column (blank if none)	You write it
`customer_rows.csv`	platform, table_name, customer_id, row_count, measured_on	The query that `--queries` prints
`exceptions.csv`	kind (customer or table), id, exception, basis, ends_on, review_by	You write it. A table's id is `platform:table_name`
`review_log.csv`	review_date, reviewed_by	You write it
`snowflake_tables.csv`	TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, IS_TRANSIENT, RETENTION_TIME, DELETED	Snowflake: ACCOUNT_USAGE.TABLES (read 2026-11-01)
`databricks_table_properties.csv`	table_name, deleted_file_retention_days (blank if the table does not set it), deletion_vectors (true or false)	Databricks: VACUUM (SQL) (read 2026-11-01)
`databricks_history.csv`	table_name, timestamp, operation (DESCRIBE HISTORY, one table after another; it keeps 30 days by default)	Databricks: Table history schema (read 2026-11-01); Databricks: Work with table history (read 2026-11-01)
`redshift_snapshots.csv`	SnapshotIdentifier, SnapshotCreateTime, SnapshotType, ManualSnapshotRetentionPeriod	AWS CLI: describe-cluster-snapshots (read 2026-11-01)

The exception column holds one of the rule's three reasons, in these shortened forms of its wording: `business operations or other legitimate business purposes`, `required by law or regulation`, `targeted disposal is not reasonably feasible`. Any other reason fails an input check.

Step 2. Count each customer's rows in every register table

No platform view holds how many rows each customer has in a table, so the script prints the query. Set `EXPORT_DIR`, put your table register there as `table_register.csv`, then run `python3 safeguards_disposal_evidence.py --queries`. It prints one query for each table that has a customer key column, with each name in the platform's own quoting (a double-quoted part stays quoted, in backticks on Databricks) and the customer id trimmed of spaces. Each reads the whole table, so run it when the load is light. Here are the first two that `python3 safeguards_disposal_evidence.py --demo --queries` prints:

```
SELECT 'snowflake' AS platform, 'sf_prod.crm.customers' AS table_name, TRIM(CAST(customer_id AS
VARCHAR)) AS customer_id,
       COUNT(*) AS row_count, CURRENT_DATE AS measured_on
FROM sf_prod.crm.customers
GROUP BY 1, 2, 3;

SELECT 'snowflake' AS platform, 'sf_prod.loans.applications' AS table_name, TRIM(CAST(cust_id AS
VARCHAR)) AS customer_id,
       COUNT(*) AS row_count, CURRENT_DATE AS measured_on
FROM sf_prod.loans.applications
GROUP BY 1, 2, 3;
```

Run each query on the platform it names and save every answer as lines of `customer_rows.csv`. A customer id that is NULL in the table comes out as a blank id, and the script reports those rows as rows with no customer key. We printed these queries but did not run them on a platform: they are plain SQL, so read one before you run it. A registered table with no lines at all is NOT MEASURED, because an empty table cannot be told from one nobody counted. The demo's exception list looks like this:

```
kind,id,exception,basis,ends_on,review_by
customer,C-100057,required by law or regulation,Retention schedule item 4 (document LR-17),,2027-
03-31
table,snowflake:sf_prod.ops.app_logs,targeted disposal is not reasonably feasible,"Log lines hold
customer ids inside free text and cannot be found one customer at a time",,2027-02-28
```

Step 3. How each list is decided

The two years are counted in the list 1 query with DuckDB's own date arithmetic, our counting and not the rule's, so 29 February moves to 28 February:

```
           c.last_use + INTERVAL 2 YEAR AS two_years,
```

Every customer with rows is then joined to your register in SQL, and the status is chosen in this order. A customer inside two years is counted and not listed. Notice that an open relationship is never cleared, whatever its exceptions say:

```
        if not inreg:
            status = "NOT IN REGISTER"
        elif last is None:
            status = "LAST USE UNCLEAR"
        elif asof <= two:
            within += 1
            continue
        elif op is not False:
            status = "LAST USE UNCLEAR"      # an open relationship with an old last use is never
cleared
        elif cexc_ or notex == 0:
            status = "EXCEPTION IN FORCE"
        else:
            status = "PAST TWO YEARS"
```

An exception is in force only when its reason is one of the three, a basis is written, its review date has not passed, and, if it has an end date, that date has not passed. A blank end date is allowed. A table with no customer key is covered only by a table exception. For the retention policy review, the script lists the dates you recorded and the days between them, and marks a gap, or the days since the last review, as over the number you set in `RETENTION_REVIEW_DAYS`.

Step 4. Run it on the clean generated records

Run `python3 safeguards_disposal_evidence.py --demo`. The generated records give 59 customers, a table register of seven tables, one exception for a log table with no customer key and two exceptions for customers held under a stated reason. The run printed this (the 25 input checks all passed, shown here by their heading; the PDF has them in full):

```
== Input checks (27 run, 0 failed)
```

```
Population: 59 customers in the register; 53 customers have rows in a register table; 51 of them
are inside two years as of 2026-11-01; 2 are listed in 1.

== Customers past two years: ftc_1_customers_past_two_years
   as of 2026-11-01; customers with at least one row in a register table, more than two years after
the last date of use
   rows: 2   sha256: 1930484dbc301a3b987908f28bdb0e110dd208b9328360442182717bb5c3d10c
   customer_id | status | last_use_date | two_years_after | rows | rows_not_excepted | tables |
exception
   C-100057 | EXCEPTION IN FORCE | 2024-05-15 | 2026-05-15 | 14 | 14 |
databricks:main.crm.customers=5; databricks:main.loans.payments=9 | required by law or regulation
   C-100058 | EXCEPTION IN FORCE | 2023-10-28 | 2025-10-28 | 46 | 46 |
databricks:main.loans.payments=37; redshift:loans.applications=9 | business operations or other
legitimate business purposes

== Tables read by customer: ftc_1b_tables_by_customer
   as of 2026-11-01; every table in table_register
   rows: 7   sha256: 3f397ab75ae02bdbcbf6f2f29168fe04d10abf6b3af8aa67e9f48a0118381543
   platform | table_name | customer_key_column | status | customers_with_rows | rows
   databricks | main.crm.customers | customer_id | MEASURED | 29 | 498
```

```
   databricks | main.loans.payments | borrower_id | MEASURED | 26 | 563
   redshift | crm.customers | customer_id | MEASURED | 26 | 489
   redshift | loans.applications | applicant_id | MEASURED | 23 | 448
   snowflake | sf_prod.crm.customers | customer_id | MEASURED | 28 | 649
   snowflake | sf_prod.loans.applications | cust_id | MEASURED | 22 | 400
   snowflake | sf_prod.ops.app_logs |  | NO CUSTOMER KEY, EXCEPTION IN FORCE (targeted disposal is
not reasonably feasible) | 0 | 0

== Exceptions: ftc_2_exceptions
   as of 2026-11-01; every exception in exceptions.csv with one of the rule's three reasons; the
script never judges the stated need
   rows: 3   sha256: cd3461b6ad9bf5b8bfb7d74fa4d62f4f0f66ff6e85f43f501b584bce552fd4b5
   kind | id | exception | basis_written | ends_on | review_by | status | rows_held
   customer | C-100057 | required by law or regulation | yes |  | 2027-03-31 | IN FORCE | 14
   customer | C-100058 | business operations or other legitimate business purposes | yes |  | 2027-
01-15 | IN FORCE | 46
   table | snowflake:sf_prod.ops.app_logs | targeted disposal is not reasonably feasible | yes |  |
2027-02-28 | IN FORCE | not counted

== Disposed but still recoverable: ftc_3_recoverable_after_disposal
   as of 2026-11-01; days a row deleted on AS_OF can still be brought back; a count to look at, not
a finding, except INDEFINITE, DELETION VECTORS WITH NO PURGE and NOT MEASURED
   rows: 7   sha256: 2e1a1036fe6156583eb8449ac5e50b43c1c7d7ccace55639ac2df6cc61aed8bf
   platform | table_name | recoverable_days | status | basis
   databricks | main.crm.customers | 7 | AT LEAST 7 DAYS | until the first VACUUM after that; last
VACUUM 2026-10-24 (8 days before AS_OF)
   databricks | main.loans.payments | 30 | AT LEAST 30 DAYS | until the first VACUUM after that;
last VACUUM 2026-10-30 (2 days before AS_OF)
   redshift | crm.customers | 28 | UP TO 28 DAYS | a manual snapshot has 28 days left; automated
snapshots up to 1 day
   redshift | loans.applications | 28 | UP TO 28 DAYS | a manual snapshot has 28 days left;
automated snapshots up to 1 day
   snowflake | sf_prod.crm.customers | 21 | 21 DAYS | Time Travel 14 days + 7 days Fail-safe
   snowflake | sf_prod.loans.applications | 37 | 37 DAYS | Time Travel 30 days + 7 days Fail-safe
   snowflake | sf_prod.ops.app_logs | 1 | 1 DAY | transient: Time Travel 1 day, no Fail-safe


== Retention policy reviews: ftc_4_retention_policy_reviews
   as of 2026-11-01; reviews on record; the allowed gap is your setting, 365 days; the last row is
AS_OF itself
   rows: 4   sha256: 3e7839c943a0b0c77505b26df2969b22ffdc38a0af27a71ef22c0a7fd3948825
   date | days_since_previous | reviewer_named | status
   2024-05-15 |  | yes | OK
   2025-04-02 | 322 | yes | OK
   2026-03-09 | 341 | yes | OK
   2026-11-01 | 237 |  | OK: 237 DAYS SINCE THE LAST REVIEW

Wrote the lists to evidence-out/run-20261101-211206/ (CSV, one file per list).
Result: nothing needs follow-up (0 items)
```

The script found no item to follow up in these records. Two customers are past two years and still have rows, and both have an exception in force. A third customer's two years end on the as-of day, so that customer is not listed. The recoverable list is a count to read, not a finding.

Step 5. Run it again with findings planted in the records

Run `python3 safeguards_disposal_evidence.py --demo --plant`. This adds customers past two years, one open relationship with an old last-use date, one customer found only in the platform rows, exceptions with no basis, expired and not reviewed, a table never counted, 12 rows with no customer key, a Databricks table with deletion vectors and no purge in its history, a Redshift manual snapshot kept indefinitely and a review log that has gone quiet. One customer was last served on 2024-10-31, so their two years ended one day before the as-of day; another on 2024-02-29, which counts to 2026-02-28.

```
Population: 67 customers in the register; 62 customers have rows in a register table; 51 of them
are inside two years as of 2026-11-01; 11 are listed in 1.

== Customers past two years: ftc_1_customers_past_two_years
   as of 2026-11-01; customers with at least one row in a register table, more than two years after
the last date of use
   rows: 11   sha256: d791670b7bd77a1514c739b01605f58b73d7f4668f83fe6dafcb09d9afc5c5b4
   customer_id | status | last_use_date | two_years_after | rows | rows_not_excepted | tables |
exception
   C-100057 | EXCEPTION IN FORCE | 2024-05-15 | 2026-05-15 | 5 | 5 |
databricks:main.crm.customers=5 | required by law or regulation
   C-100058 | EXCEPTION IN FORCE | 2023-10-28 | 2025-10-28 | 9 | 9 | redshift:loans.applications=9
| business operations or other legitimate business purposes
   C-100060 | PAST TWO YEARS | 2023-05-04 | 2025-05-04 | 67 | 67 |
databricks:main.crm.customers=37; redshift:loans.applications=30 |
   C-100061 | PAST TWO YEARS | 2022-12-19 | 2024-12-19 | 21 | 21 | redshift:loans.applications=6;
snowflake:sf_prod.crm.customers=15 |
   C-100062 | PAST TWO YEARS | 2024-10-31 | 2026-10-31 | 16 | 16 | redshift:loans.applications=16 |
   C-100063 | PAST TWO YEARS | 2024-02-29 | 2026-02-28 | 7 | 7 | redshift:loans.applications=7 |
   C-100064 | LAST USE UNCLEAR | 2023-01-15 | 2025-01-15 | 23 | 23 |
snowflake:sf_prod.loans.applications=23 |
   C-100065 | PAST TWO YEARS | 2023-08-20 | 2025-08-20 | 52 | 52 |
databricks:main.crm.customers=16; redshift:crm.customers=36 |
   C-100066 | PAST TWO YEARS | 2023-09-02 | 2025-09-02 | 1 | 1 | databricks:main.crm.customers=1 |
   C-100067 | PAST TWO YEARS | 2023-10-10 | 2025-10-10 | 42 | 42 |
databricks:main.crm.customers=12; redshift:loans.applications=30 |
   C-100099 | NOT IN REGISTER |  |  | 4 | 4 | redshift:loans.applications=4 |

== Tables read by customer: ftc_1b_tables_by_customer
   as of 2026-11-01; every table in table_register
   rows: 7   sha256: 7c17ad7b03475f12786cf1ba5e8c9fccb0bf8172a7c6b5f60a93000dc7e04b6b
   platform | table_name | customer_key_column | status | customers_with_rows | rows
   databricks | main.crm.customers | customer_id | MEASURED | 33 | 564
   databricks | main.loans.payments | borrower_id | NOT MEASURED: no row counts for this table | 0
| 0
   redshift | crm.customers | customer_id | MEASURED | 27 | 525
   redshift | loans.applications | applicant_id | MEASURED | 29 | 541
   snowflake | sf_prod.crm.customers | customer_id | MEASURED; 12 ROWS WITH NO CUSTOMER KEY | 29 |
664
   snowflake | sf_prod.loans.applications | cust_id | MEASURED | 23 | 423
   snowflake | sf_prod.ops.app_logs |  | NOT MEASURED: no customer key and no exception in force |
0 | 0

== Exceptions: ftc_2_exceptions
```

```
   as of 2026-11-01; every exception in exceptions.csv with one of the rule's three reasons; the
script never judges the stated need
   rows: 5   sha256: 77cae8f1670d90af98191d0e196573112dc7f90a1a714e2b0f513c62cc4acddc
   kind | id | exception | basis_written | ends_on | review_by | status | rows_held
   customer | C-100057 | required by law or regulation | yes |  | 2027-03-31 | IN FORCE | 5
   customer | C-100058 | business operations or other legitimate business purposes | yes |  | 2027-
01-15 | IN FORCE | 9
   customer | C-100065 | business operations or other legitimate business purposes | yes | 2026-06-
30 | 2026-12-01 | EXPIRED | 52
   customer | C-100066 | required by law or regulation | no |  | 2027-01-01 | NO BASIS | 1
   customer | C-100067 | required by law or regulation | yes |  | 2026-09-30 | NOT REVIEWED | 42

== Disposed but still recoverable: ftc_3_recoverable_after_disposal
   as of 2026-11-01; days a row deleted on AS_OF can still be brought back; a count to look at, not
a finding, except INDEFINITE, DELETION VECTORS WITH NO PURGE and NOT MEASURED
   rows: 7   sha256: 94064534853d96518c946be3beb28dac44e1e9396285d357315a2d25255ae724
   platform | table_name | recoverable_days | status | basis
   databricks | main.crm.customers |  | DELETION VECTORS, NO PURGE IN HISTORY | a DELETE only marks
rows: they stay in the table's files until REORG TABLE ... APPLY (PURGE), then at least 7 days
until a VACUUM; history is kept 30 days by default
   databricks | main.loans.payments | 30 | NO VACUUM IN HISTORY, AT LEAST 30 DAYS | old files stay
until a VACUUM removes them; history is kept 30 days by default, so an older VACUUM may not show
   redshift | crm.customers |  | INDEFINITE | a manual snapshot has 28 days left; a manual snapshot
with retention -1 is kept indefinitely; automated snapshots up to 1 day
   redshift | loans.applications |  | INDEFINITE | a manual snapshot has 28 days left; a manual
snapshot with retention -1 is kept indefinitely; automated snapshots up to 1 day
   snowflake | sf_prod.crm.customers | 21 | 21 DAYS | Time Travel 14 days + 7 days Fail-safe
   snowflake | sf_prod.loans.applications | 37 | 37 DAYS | Time Travel 30 days + 7 days Fail-safe
   snowflake | sf_prod.ops.app_logs | 1 | 1 DAY | transient: Time Travel 1 day, no Fail-safe


== Retention policy reviews: ftc_4_retention_policy_reviews
   as of 2026-11-01; reviews on record; the allowed gap is your setting, 365 days; the last row is
AS_OF itself
   rows: 3   sha256: e3a51e56ac1fd59bbcaf337b5f79b4b68142f50a0aaa722acb79ee55bea101c8
   date | days_since_previous | reviewer_named | status
   2024-05-15 |  | yes | OK
   2025-08-14 | 456 | yes | GAP OVER 365 DAYS
   2026-11-01 | 444 |  | OVERDUE: 444 DAYS SINCE THE LAST REVIEW

Wrote the lists to evidence-out/run-20261101-211207/ (CSV, one file per list).
Result: NEEDS FOLLOW-UP (20 items)
```

Every planted finding is on a list: the customers past two years, the customer with an open relationship, the customer not in the register, the three exceptions that cannot count, the two tables that cannot be read by customer, the rows with no customer key, the deletion-vector table, the snapshot with no end and the overdue review.

Step 6. Run it with faults planted in the inputs

A list built from a broken export is worse than no list, so the script checks its inputs first and marks every list INCOMPLETE if one fails. Run `python3 safeguards_disposal_evidence.py --demo --plant-input` to plant one fault for each check, the settings' own checks included. A missing

column and a missing file would hide the checks behind them, so they have their own run, `--plant-files`.

```
== Input checks (27 run, 26 failed)
[ok]   input files are present, readable and have every column
[FAIL] customer_register: no customer_id twice or blank: data rows 60
[FAIL] customer_register: last_use_date is YYYY-MM-DD or blank: data rows 61
[FAIL] customer_register: no last_use_date after AS_OF: data rows 62
[FAIL] customer_register: relationship_open is true or false: data rows 63
[FAIL] table_register: no table twice: data rows 8
[FAIL] table_register: platform is snowflake, databricks or redshift: data rows 9
[FAIL] table_register: customer_key_column is a plain column name or blank: data rows 10
[FAIL] customer_rows: every table is in table_register: data rows 155
[FAIL] customer_rows: row_count is a whole number, 0 or more: data rows 156
[FAIL] customer_rows: no customer counted twice for one table: data rows 157
[FAIL] customer_rows: measured_on is YYYY-MM-DD, not after AS_OF, within 3 days of it: data rows 2
[FAIL] customer_rows: no counts for a table the register says has no customer key: data rows 158
[FAIL] exceptions: kind is customer or table: data rows 4
[FAIL] exceptions: exception is one of the rule's three: data rows 5
[FAIL] exceptions: every id is a customer in the register or the rows, or a table in
table_register: data rows 4, 6
[FAIL] exceptions: ends_on and review_by are YYYY-MM-DD or blank: data rows 7
[FAIL] review_log: review_date is YYYY-MM-DD and not after AS_OF: data rows 4
[FAIL] snowflake_tables: for each register table's live row, RETENTION_TIME is a whole number and
IS_TRANSIENT is YES or NO: data rows 2
[FAIL] SNOWFLAKE_MIN_DATA_RETENTION_DAYS is a whole number, 0 or more: detail setting
[FAIL] databricks_table_properties: deleted_file_retention_days is a whole number or blank: data
rows 2
[FAIL] databricks_table_properties: deletion_vectors is true or false: data rows 1
[FAIL] databricks_history: every VACUUM operation is VACUUM END or VACUUM START: data rows 5
[FAIL] databricks_history: timestamp is YYYY-MM-DD HH:MM:SS: data rows 6
[FAIL] redshift_snapshots: SnapshotCreateTime is YYYY-MM-DD HH:MM:SS and the type is manual or
automated: data rows 3
[FAIL] redshift_snapshots: a manual snapshot's ManualSnapshotRetentionPeriod is a whole number or
-1: data rows 4
[FAIL] redshift_snapshots: REDSHIFT_AUTOMATED_RETENTION_DAYS is set when automated snapshots exist:
detail setting is None
```

```
== Input checks (27 run, 1 failed)
[FAIL] input files are present, readable and have every column: detail review_log.csv: missing
column reviewed_by, snowflake_tables.csv: not found
```

Each check and the fault that trips it

Input check	Planted fault
Files are present, readable and have every column	the --plant-files run: no reviewed_by column, and snowflake_tables.csv missing
customer_register: no customer_id twice or blank	the first customer listed twice
customer_register: last_use_date is YYYY-MM-DD or blank	8/5/26

Input check	Planted fault
customer_register: no last_use_date after AS_OF	2027-01-10
customer_register: relationship_open is true or false	maybe
table_register: no table twice	the first table listed twice
table_register: platform is one of the three	snowflak
table_register: customer_key_column is a plain column name or blank	cust id
customer_rows: every table is in table_register	counts for crm.old_staging
customer_rows: row_count is a whole number	12x
customer_rows: no customer counted twice for one table	the first line repeated
customer_rows: measured_on is readable, not after AS_OF, recent	one line measured on 2026-09-01
customer_rows: no counts for a table with no customer key	counts for the log table
exceptions: kind is customer or table	kind group
exceptions: exception is one of the rule's three	we just like it
exceptions: every id exists	a customer id that is nowhere in the register or the rows
exceptions: ends_on and review_by are dates or blank	31/12/2026
review_log: review_date is a date, not after AS_OF	2026-12-25
snowflake_tables: a register table's RETENTION_TIME is a whole number	one week
SNOWFLAKE_MIN_DATA_RETENTION_DAYS is a whole number	the setting at -1
databricks_table_properties: whole number or blank	7d
databricks_table_properties: deletion_vectors is true or false	maybe
databricks_history: every VACUUM operation is known	VACUUM FULL
databricks_history: timestamp is readable	28 Oct 2026 3am

Input check	Planted fault
redshift_snapshots: time and type are readable	last Tuesday
redshift_snapshots: manual retention is a whole number or -1	a month
redshift_snapshots: automated retention setting is set	REDSHIFT_AUTOMATED_RETENTION_DAYS left unset

The code

The complete download, `safeguards_disposal_evidence.py`, also at www.datameaning.com/guides/ftc-safeguards-rule-two-year-disposal-evidence-for-snowflake-databricks-redshift/safeguards_disposal_evidence.py.

```
#!/usr/bin/env python3
"""Two-year disposal evidence (FTC Safeguards Rule, 16 CFR 314.4(c)(6)) from Snowflake, Databricks
and
Redshift exports, in DuckDB.

    python3 safeguards_disposal_evidence.py --demo                # clean generated records
    python3 safeguards_disposal_evidence.py --demo --plant        # generated records with planted
findings
    python3 safeguards_disposal_evidence.py --demo --plant-input  # generated records with planted
input faults
    python3 safeguards_disposal_evidence.py --demo --plant-files  # a missing column and a missing
file
    python3 safeguards_disposal_evidence.py --queries             # set EXPORT_DIR first: print the
row-count query for
                                                                  # every keyed table in YOUR
table_register.csv there
    python3 safeguards_disposal_evidence.py                       # your exports: set the settings
below first
    python3 safeguards_disposal_evidence.py --strict              # exit 1 unless nothing needs
follow-up
    (--work-dir DIR puts the generated/ and evidence-out/ folders somewhere other than the current
folder)

The rule (314.4(c)(6)(i)): dispose of customer information "no later than two years after the last
date the
information is used in connection with the provision of a product or service to the customer to
which it
relates", unless it is necessary for business operations or other legitimate business purposes, is
otherwise
required to be retained by law or regulation, or targeted disposal is not reasonably feasible. (c)
(6)(ii):
"Periodically review your data retention policy to minimize the unnecessary retention of data".

No platform knows when a customer was last served. You keep that date in your own system of record.
You export:
  customer_register.csv  customer_id, last_use_date (YYYY-MM-DD), relationship_open (true or false)
  table_register.csv     platform, table_name, customer_key_column (blank if the table has no
customer key)
```

```
  customer_rows.csv      platform, table_name, customer_id, row_count, measured_on  (--queries
prints the query)
  exceptions.csv         kind (customer or table), id, exception, basis, ends_on, review_by
  review_log.csv         review_date, reviewed_by
plus each platform's own recovery settings (Snowflake ACCOUNT_USAGE.TABLES, Databricks table
properties and
DESCRIBE HISTORY, Redshift describe-cluster-snapshots). It prints five lists, each with its
filters, row count
and a SHA-256 of its rows, and writes each as a CSV:
  1.  customers past two years that still have rows (PAST TWO YEARS, LAST USE UNCLEAR, NOT IN
REGISTER,
      EXCEPTION IN FORCE), and 1b the tables read, with NOT MEASURED where a table could not be
read by customer
  2.  exceptions, with NO BASIS, EXPIRED and NOT REVIEWED flags (the script never judges a stated
business need)
  3.  disposed but still recoverable: days a row deleted on AS_OF can still be brought back, per
table
  4.  retention policy reviews on record and the days between them (the rule gives no interval: set
one)
The script reads files; it never connects to a platform and never writes to one.

The demo records are GENERATED (seed 234): not a company's data. Column names come from:
  Snowflake  https://docs.snowflake.com/en/sql-reference/account-usage/tables
  Databricks https://docs.databricks.com/aws/en/tables/history-schema
             https://docs.databricks.com/aws/en/sql/language-manual/delta-vacuum
  Redshift   https://docs.aws.amazon.com/cli/latest/reference/redshift/describe-cluster-
snapshots.html
Our assumptions, to check against one of your own rows: Databricks DESCRIBE HISTORY names a
finished VACUUM
"VACUUM END" and a purge REORG (the history schema page does not list the operation names), so both
are settings.
"""
import argparse
import csv
import datetime as dt
import hashlib
import pathlib
import random
import re
import sys

import duckdb

# ---- CHANGE THESE FOR YOUR OWN DATA -------------------------------------------------------
AS_OF = "2026-11-01"                # the day you took the exports (UTC)
EXPORT_DIR = None                   # folder holding the export files named in FILES below
PLATFORMS_EXPORTED = ["snowflake", "databricks", "redshift"]   # a platform not listed is NOT
CHECKED in list 3
RETENTION_REVIEW_DAYS = None        # no default: the rule says "periodically". Your longest
allowed gap, in days
REDSHIFT_AUTOMATED_RETENTION_DAYS = None     # no default: your cluster's automated snapshot
retention in days
DATABRICKS_DEFAULT_RETENTION_DAYS = 7        # delta.deletedFileRetentionDuration when the table
does not set it
DATABRICKS_VACUUM_DONE = "VACUUM END"        # the DESCRIBE HISTORY operation that is a finished
VACUUM
DATABRICKS_PURGE_OPS = ["REORG"]             # the operation that rewrites files with deletion
vectors (REORG ... APPLY (PURGE))
```

```
SNOWFLAKE_MIN_DATA_RETENTION_DAYS = 0        # your account's MIN_DATA_RETENTION_TIME_IN_DAYS, 0 if
not set
ROW_COUNTS_MAX_STALE_DAYS = 3        # row counts must have been measured within this many days of
AS_OF
# ----------------------------------------------------------------------------------------------

FAILSAFE_DAYS = 7   # Snowflake: "a (non-configurable) 7-day period" (docs.snowflake.com/en/user-
guide/data-failsafe)
FILES = {
    "customer_register": "customer_register.csv",
    "table_register": "table_register.csv",
    "customer_rows": "customer_rows.csv",
    "exceptions": "exceptions.csv",
    "review_log": "review_log.csv",
    "snowflake_tables": "snowflake_tables.csv",
    "databricks_table_properties": "databricks_table_properties.csv",
    "databricks_history": "databricks_history.csv",
    "redshift_snapshots": "redshift_snapshots.csv",
}
PLATFORM_FILES = {
    "snowflake": ["snowflake_tables"],
    "databricks": ["databricks_table_properties", "databricks_history"],
    "redshift": ["redshift_snapshots"],
}
COLUMNS = {   # the platforms' published column names, lower-cased here; the file's own case does
not matter
    "customer_register": ["customer_id", "last_use_date", "relationship_open"],
    "table_register": ["platform", "table_name", "customer_key_column"],
    "customer_rows": ["platform", "table_name", "customer_id", "row_count", "measured_on"],
    "exceptions": ["kind", "id", "exception", "basis", "ends_on", "review_by"],
    "review_log": ["review_date", "reviewed_by"],
    "snowflake_tables": ["table_catalog", "table_schema", "table_name", "is_transient",
"retention_time", "deleted"],
    "databricks_table_properties": ["table_name", "deleted_file_retention_days",
"deletion_vectors"],
    "databricks_history": ["table_name", "timestamp", "operation"],
    "redshift_snapshots": ["snapshotidentifier", "snapshotcreatetime", "snapshottype",
"manualsnapshotretentionperiod"],
}
PLATFORMS = ["snowflake", "databricks", "redshift"]
# The rule's three exceptions, in short forms taken from 314.4(c)(6)(i); exceptions.csv must use
these exact forms.
EXCEPTIONS = ["business operations or other legitimate business purposes",
              "required by law or regulation",
              "targeted disposal is not reasonably feasible"]
OUT_DIR = pathlib.Path("evidence-out")
GEN_DIR = pathlib.Path("generated")
DATE_RE = re.compile(r"\d{4}-\d{2}-\d{2}")
TS_RE = re.compile(r"(\d{4}-\d{2}-\d{2})([ T]\d{2}:\d{2}:\d{2}(\.\d+)?( ?(Z|\+00:00|\+0000))?)?")
IDENT = re.compile(r"^[A-Za-z_][A-Za-z0-9_$]*$")
PART = r'(?:"(?:[^"]|"")+"|[A-Za-z0-9_$]+)'
NAME = re.compile(rf"^{PART}(\.{PART})*$")   # dotted parts, each plain or double-quoted


def pdate(s):
    """A date read strictly: YYYY-MM-DD only. 8/5/26 and 2026/09/24 are None, and counted by a
check."""
    s = (s or "").strip()
```

```
    if not DATE_RE.fullmatch(s):
        return None
    try:
        return dt.date.fromisoformat(s)
    except ValueError:
        return None


def pts(s):
    """The UTC date of a timestamp written YYYY-MM-DD HH:MM:SS (a T, fraction, Z or +00:00 is
accepted)."""
    m = TS_RE.fullmatch((s or "").strip())
    return pdate(m.group(1)) if m else None


def dn(n):
    return f"{n} DAY" if int(n) == 1 else f"{int(n)} DAYS"


def norm_table(s):
    return (s or "").replace('"', "").strip().lower()


def quote_name(platform, table):
    """The table name in the platform's own quoting: a double-quoted part stays quoted (backticks
on Databricks)."""
    parts = re.findall(PART, table)
    if platform == "databricks":
        parts = ["`" + p[1:-1].replace('""', '"').replace("`", "``") + "`" if p.startswith('"')
else p for p in parts]
    return ".".join(parts)


def query_for(platform, table, key):
    cast = "STRING" if platform == "databricks" else "VARCHAR"
    lit = norm_table(table).replace("'", "''")
    return (f"SELECT '{platform}' AS platform, '{lit}' AS table_name, TRIM(CAST({key} AS {cast}))
AS customer_id,\n"
            f"       COUNT(*) AS row_count, CURRENT_DATE AS measured_on\nFROM {quote_name(platform,
table)}\nGROUP BY 1, 2, 3;")


# ---------------------------------------------------------------------------------------------
# generated demo records (seed 234)
# ---------------------------------------------------------------------------------------------
def write_csv(path, header, rows):
    with open(path, "w", newline="", encoding="utf-8") as f:
        w = csv.writer(f)
        w.writerow(header)
        w.writerows(rows)


def generate(gen_dir, plant, plant_input, plant_files=False):
    gen_dir.mkdir(parents=True, exist_ok=True)
    rnd = random.Random(234)
    asof = dt.date.fromisoformat(AS_OF)
    tables = [("snowflake", "sf_prod.crm.customers", "customer_id"), ("snowflake",
"sf_prod.loans.applications", "cust_id"),
```

```
              ("snowflake", "sf_prod.ops.app_logs", ""), ("databricks", "main.crm.customers",
"customer_id"),
              ("databricks", "main.loans.payments", "borrower_id"), ("redshift", "crm.customers",
"customer_id"),
              ("redshift", "loans.applications", "applicant_id")]
    keyed = [t for t in tables if t[2]]
    cust, rows, exc = [], [], []

    def add_rows(cid, n_tables):
        picks = keyed[:] if n_tables is None else rnd.sample(keyed, n_tables)
        for p, t, _ in sorted(picks):
            rows.append([p, t, cid, str(rnd.randint(1, 40))])

    for i in range(1, 51):                       # served in the last two years
        cid = f"C-{100000 + i}"
        cust.append([cid, (asof - dt.timedelta(days=rnd.randint(10, 700))).isoformat(), "true" if
rnd.random() < .8 else "false"])
        add_rows(cid, rnd.randint(2, 4))
    for i in range(51, 57):                      # served more than two years ago and already
disposed: no rows
        cust.append([f"C-{100000 + i}", (asof - dt.timedelta(days=rnd.randint(760,
1500))).isoformat(), "false"])
    held = [("C-100057", 900, EXCEPTIONS[1], "Retention schedule item 4 (document LR-17)", "",
"2027-03-31"),
            ("C-100058", 1100, EXCEPTIONS[0], "Open dispute file DF-22, owner: legal", "", "2027-
01-15")]
    for cid, ago, typ, basis, ends, review in held:
        cust.append([cid, (asof - dt.timedelta(days=ago)).isoformat(), "false"])
        add_rows(cid, 2)
        exc.append(["customer", cid, typ, basis, ends, review])
    cust.append(["C-100059", "2024-11-01", "false"])   # two years end today: due, not yet past
    add_rows("C-100059", 2)
    exc.append(["table", "snowflake:sf_prod.ops.app_logs", EXCEPTIONS[2],
                "Log lines hold customer ids inside free text and cannot be found one customer at a
time", "", "2027-02-28"])
    sf = [["SF_PROD", "CRM", "CUSTOMERS", "NO", "14", ""], ["SF_PROD", "LOANS", "APPLICATIONS",
"NO", "30", ""],
          ["SF_PROD", "OPS", "APP_LOGS", "YES", "1", ""]]
    dbp = [["main.crm.customers", "", "false"], ["main.loans.payments", "30", "false"]]
    dbh = [["main.crm.customers", "2026-10-17 03:00:12", "VACUUM END"], ["main.crm.customers",
"2026-10-24 03:00:09", "VACUUM END"],
           ["main.crm.customers", "2026-10-26 11:14:40", "DELETE"], ["main.loans.payments", "2026-
10-30 03:01:55", "VACUUM END"]]
    snaps = [["rs-auto-0001", "2026-10-31 02:00:00", "automated", ""], ["rs-manual-q3", "2026-10-15
00:00:00", "manual", "45"]]
    reviews = [["2024-05-15", "Head of Data Governance"], ["2025-04-02", "Head of Data
Governance"],
               ["2026-03-09", "Head of Data Governance"]]
    cust_rows = rows
    if plant:
        planted = [("C-100060", "2023-05-04", "false", 3), ("C-100061", "2022-12-19", "false", 2),
                   ("C-100062", "2024-10-31", "false", 2), ("C-100063", "2024-02-29", "false", 1),
                   ("C-100064", "2023-01-15", "true", 2), ("C-100065", "2023-08-20", "false", 2),
                   ("C-100066", "2023-09-02", "false", 1), ("C-100067", "2023-10-10", "false", 2)]
        for cid, last, op, n in planted:
            cust.append([cid, last, op])
            add_rows(cid, n)
        exc.append(["customer", "C-100065", EXCEPTIONS[0], "Contract dispute CD-31", "2026-06-30",
```

```
"2026-12-01"])
        exc.append(["customer", "C-100066", EXCEPTIONS[1], "", "", "2027-01-01"])
        exc.append(["customer", "C-100067", EXCEPTIONS[1], "Retention schedule item 9", "", "2026-
09-30"])
        exc = [e for e in exc if e[0] != "table"]                      # no exception for the log
table
        cust_rows = [r for r in rows if r[1] != "main.loans.payments"]  # that table was never
counted
        cust_rows += [["redshift", "loans.applications", "C-100099", "4"], ["snowflake",
"sf_prod.crm.customers", "", "12"]]
        dbh = [r for r in dbh if not (r[0] == "main.loans.payments" and r[2] == "VACUUM END")] + \
              [["main.loans.payments", "2026-10-28 09:00:00", "DELETE"]]
        snaps.append(["rs-manual-2024-close", "2024-12-31 00:00:00", "manual", "-1"])
        dbp[0][2] = "true"                                              # deletion vectors on,
never purged
        reviews = [["2024-05-15", "Head of Data Governance"], ["2025-08-14", "Head of Data
Governance"]]
    cr = [r + [AS_OF] for r in cust_rows]
    files = {"customer_register": (COLUMNS["customer_register"], cust),
             "table_register": (COLUMNS["table_register"], [list(t) for t in tables]),
             "customer_rows": (COLUMNS["customer_rows"], cr), "exceptions": (COLUMNS["exceptions"],
exc),
             "review_log": (COLUMNS["review_log"], reviews),
             "snowflake_tables": (["TABLE_CATALOG", "TABLE_SCHEMA", "TABLE_NAME", "IS_TRANSIENT",
"RETENTION_TIME", "DELETED"], sf),
             "databricks_table_properties": (COLUMNS["databricks_table_properties"], dbp),
             "databricks_history": (COLUMNS["databricks_history"], dbh),
             "redshift_snapshots": (["SnapshotIdentifier", "SnapshotCreateTime", "SnapshotType",
"ManualSnapshotRetentionPeriod"], snaps)}
    if plant_input:
        files["review_log"][1].append(["2026-12-25", "Head of Data Governance"])
# a review dated after AS_OF
        files["customer_register"][1].extend([list(cust[0]), ["C-100070", "8/5/26", "false"], ["C-
100071", "2027-01-10", "false"],
                                              ["C-100072", "2025-01-10", "maybe"]])
# duplicate id, bad date, future date, bad flag
        files["table_register"][1].extend([list(tables[0]), ["snowflak", "sf_prod.x.y", "id"],
                                           ["redshift", "crm.keyed_oddly", "cust id"]])
# duplicate, unknown platform, unusable key name
        cr2 = files["customer_rows"][1]
        cr2.extend([["redshift", "crm.old_staging", "C-100001", "5", AS_OF], ["redshift",
"crm.customers", "C-100002", "12x", AS_OF],
                    list(cr2[0]), ["snowflake", "sf_prod.ops.app_logs", "C-100003", "2", AS_OF]])
# not in register, bad count, duplicate, keyless table
        cr2[1][4] = "2026-09-01"
# counted too long ago
        files["exceptions"][1].extend([["group", "C-100001", EXCEPTIONS[1], "x", "", "2027-01-01"],
                                       ["customer", "C-100002", "we just like it", "x", "", "2027-
01-01"],
                                       ["customer", "C-999999", EXCEPTIONS[1], "x", "", "2027-01-
01"],
                                       ["customer", "C-100003", EXCEPTIONS[1], "x", "31/12/2026",
"2027-01-01"]])
        files["snowflake_tables"][1][1][4] = "one week"
        files["databricks_table_properties"][1][1][1] = "7d"
        files["databricks_table_properties"][1][0][2] = "maybe"
        files["databricks_history"][1].extend([["main.crm.customers", "2026-10-28 03:00:00",
"VACUUM FULL"],
```

```
                                                ["main.crm.customers", "28 Oct 2026 3am",
"DELETE"]])
        files["redshift_snapshots"][1].extend([["rs-manual-bad", "last Tuesday", "manual", "30"],
                                                ["rs-manual-month", "2026-10-20 00:00:00", "manual",
"a month"]])
    if plant_files:
        files["review_log"] = (["review_date"], [[r[0]] for r in reviews])
# no reviewed_by column
        del files["snowflake_tables"]
# a file that is not there
    for name, (header, data) in files.items():
        write_csv(gen_dir / FILES[name], header, data)


# ------------------------------------------------------------------------------------------
def main():
    global EXPORT_DIR, RETENTION_REVIEW_DAYS, REDSHIFT_AUTOMATED_RETENTION_DAYS,
SNOWFLAKE_MIN_DATA_RETENTION_DAYS
    ap = argparse.ArgumentParser()
    ap.add_argument("--demo", action="store_true")
    ap.add_argument("--plant", action="store_true")
    ap.add_argument("--plant-input", action="store_true")
    ap.add_argument("--plant-files", action="store_true")
    ap.add_argument("--queries", action="store_true")
    ap.add_argument("--strict", action="store_true")
    ap.add_argument("--work-dir", default=".")
    a = ap.parse_args()
    work = pathlib.Path(a.work_dir)
    asof = pdate(AS_OF)
    if asof is None:
        sys.exit("AS_OF must be written YYYY-MM-DD")
    con = duckdb.connect()
    if a.demo:
        GEN_DIR_ = work / GEN_DIR
        generate(GEN_DIR_, a.plant, a.plant_input, a.plant_files)
        EXPORT_DIR = GEN_DIR_
        # Demo settings: ours, to let the lists run. The rule sets neither interval.
        RETENTION_REVIEW_DAYS = 365
        REDSHIFT_AUTOMATED_RETENTION_DAYS = None if a.plant_input else 1
        if a.plant_input:
            SNOWFLAKE_MIN_DATA_RETENTION_DAYS = -1   # a setting that cannot be right
        print("Demo records are GENERATED (seed 234); not a company's data.")
        print(f"Demo settings: AS_OF {AS_OF}, RETENTION_REVIEW_DAYS {RETENTION_REVIEW_DAYS}, "
              f"REDSHIFT_AUTOMATED_RETENTION_DAYS {REDSHIFT_AUTOMATED_RETENTION_DAYS}")
    if EXPORT_DIR is None:
        if a.queries:
            sys.exit("Set EXPORT_DIR to the folder holding table_register.csv, then run --
queries.")
        sys.exit("Set EXPORT_DIR at the top of the script, or run with --demo.")
    exp = pathlib.Path(EXPORT_DIR)

    file_problems = []

    def load(name):
        path = exp / FILES[name]
        if not path.is_file():
            file_problems.append(f"{FILES[name]}: not found")
            return []
```

```
        try:
            cur = con.execute("SELECT * FROM read_csv(?, header=true, all_varchar=true,
null_padding=true)", [str(path)])
            head = [d[0].strip().lower() for d in cur.description]
            data = cur.fetchall()
        except duckdb.Error:
            file_problems.append(f"{FILES[name]}: cannot be read as a CSV file")
            return []
        missing = [c for c in COLUMNS[name] if c not in head]
        if missing:
            file_problems.append(f"{FILES[name]}: missing column {', '.join(missing)}")
            return []
        out = []
        for i, r in enumerate(data, 1):
            d = {h: ("" if v is None else str(v).strip()) for h, v in zip(head, r)}
            d["_n"] = i
            out.append(d)
        return out

    if a.queries:
        reg = load("table_register")
        if file_problems:
            sys.exit("; ".join(file_problems))
        for r in reg:
            if not r["customer_key_column"]:
                continue
            if r["platform"].lower() not in PLATFORMS or not NAME.match(r["table_name"]) or not
IDENT.match(r["customer_key_column"]):
                sys.exit(f"table_register.csv data row {r['_n']}: platform, table or key column
cannot be used in a query")
            print(query_for(r["platform"].lower(), r["table_name"], r["customer_key_column"]) +
"\n")
        return 0

    # ---- load ------------------------------------------------------------------------
    custs = load("customer_register")
    treg = load("table_register")
    crow = load("customer_rows")
    excs = load("exceptions")
    revs = load("review_log")
    pdata = {n: load(n) for p in PLATFORMS if p in PLATFORMS_EXPORTED for n in PLATFORM_FILES[p]}

    checks = []

    def chk(label, bad):
        checks.append((label, bad))

    chk("input files are present, readable and have every column", file_problems)

    # customer register
    seen, dup = set(), []
    for r in custs:
        if r["customer_id"] in seen:
            dup.append(r["_n"])
        seen.add(r["customer_id"])
    chk("customer_register: no customer_id twice or blank",
        dup + [r["_n"] for r in custs if not r["customer_id"]])
    chk("customer_register: last_use_date is YYYY-MM-DD or blank",
```

```
        [r["_n"] for r in custs if r["last_use_date"] and pdate(r["last_use_date"]) is None])
    chk("customer_register: no last_use_date after AS_OF",
        [r["_n"] for r in custs if pdate(r["last_use_date"]) and pdate(r["last_use_date"]) > asof])
    chk("customer_register: relationship_open is true or false",
        [r["_n"] for r in custs if r["relationship_open"].lower() not in ("true", "false")])

    # table register
    keys, dupt = set(), []
    for r in treg:
        k = (r["platform"].lower(), norm_table(r["table_name"]))
        if k in keys:
            dupt.append(r["_n"])
        keys.add(k)
    chk("table_register: no table twice", dupt)
    chk("table_register: platform is snowflake, databricks or redshift",
        [r["_n"] for r in treg if r["platform"].lower() not in PLATFORMS])
    chk("table_register: customer_key_column is a plain column name or blank",
        [r["_n"] for r in treg if r["customer_key_column"] and not
IDENT.match(r["customer_key_column"])])
    key_of = {(r["platform"].lower(), norm_table(r["table_name"])): r["customer_key_column"] for r
in treg}

    # customer rows
    def rk(r):
        return (r["platform"].lower(), norm_table(r["table_name"]))

    chk("customer_rows: every table is in table_register", [r["_n"] for r in crow if rk(r) not in
key_of])
    chk("customer_rows: row_count is a whole number, 0 or more",
        [r["_n"] for r in crow if not re.fullmatch(r"\d+", r["row_count"])])
    seen, dup = set(), []
    for r in crow:
        k = rk(r) + (r["customer_id"],)
        if k in seen:
            dup.append(r["_n"])
        seen.add(k)
    chk("customer_rows: no customer counted twice for one table", dup)

    def stale(r):
        d = pdate(r["measured_on"])
        return d is None or d > asof or (asof - d).days > ROW_COUNTS_MAX_STALE_DAYS

    chk(f"customer_rows: measured_on is YYYY-MM-DD, not after AS_OF, within
{ROW_COUNTS_MAX_STALE_DAYS} days of it",
        [r["_n"] for r in crow if stale(r)])
    chk("customer_rows: no counts for a table the register says has no customer key",
        [r["_n"] for r in crow if key_of.get(rk(r)) == "" and rk(r) in key_of])

    # exceptions
    known_ids = {r["customer_id"] for r in custs} | {r["customer_id"] for r in crow if
r["customer_id"]}

    def exc_target(e):
        if e["kind"].lower() == "customer":
            return ("customer", e["id"])
        if e["kind"].lower() == "table" and ":" in e["id"]:
            p, t = e["id"].split(":", 1)
            return ("table", (p.strip().lower(), norm_table(t)))
```

```
        return None

    chk("exceptions: kind is customer or table", [r["_n"] for r in excs if r["kind"].lower() not in
("customer", "table")])
    chk("exceptions: exception is one of the rule's three",
        [r["_n"] for r in excs if r["exception"].lower() not in EXCEPTIONS])
    chk("exceptions: every id is a customer in the register or the rows, or a table in
table_register",
        [r["_n"] for r in excs if exc_target(r) is None
         or (exc_target(r)[0] == "customer" and exc_target(r)[1] not in known_ids)
         or (exc_target(r)[0] == "table" and exc_target(r)[1] not in key_of)])
    chk("exceptions: ends_on and review_by are YYYY-MM-DD or blank",
        [r["_n"] for r in excs if (r["ends_on"] and pdate(r["ends_on"]) is None)
         or (r["review_by"] and pdate(r["review_by"]) is None)])

    # review log
    chk("review_log: review_date is YYYY-MM-DD and not after AS_OF",
        [r["_n"] for r in revs if pdate(r["review_date"]) is None or pdate(r["review_date"]) >
asof])

    # platform records
    if "snowflake" in PLATFORMS_EXPORTED:
        # only the live rows of tables in the register: a view or a dropped table in the export is
not a fault
        sf_reg = {t for p, t in key_of if p == "snowflake"}
        chk("snowflake_tables: for each register table's live row, RETENTION_TIME is a whole number
and IS_TRANSIENT "
            "is YES or NO",
            [r["_n"] for r in pdata.get("snowflake_tables", [])
             if not r["deleted"] and norm_table(f"{r['table_catalog']}.{r['table_schema']}.
{r['table_name']}") in sf_reg
             and (not re.fullmatch(r"\d+", r["retention_time"]) or r["is_transient"].upper() not in
("YES", "NO"))])
        chk("SNOWFLAKE_MIN_DATA_RETENTION_DAYS is a whole number, 0 or more",
            [] if isinstance(SNOWFLAKE_MIN_DATA_RETENTION_DAYS, int) and
SNOWFLAKE_MIN_DATA_RETENTION_DAYS >= 0
            else ["setting"])
    if "databricks" in PLATFORMS_EXPORTED:
        chk("databricks_table_properties: deleted_file_retention_days is a whole number or blank",
            [r["_n"] for r in pdata.get("databricks_table_properties", [])
             if r["deleted_file_retention_days"] and not re.fullmatch(r"\d+",
r["deleted_file_retention_days"])])
        chk("databricks_table_properties: deletion_vectors is true or false",
            [r["_n"] for r in pdata.get("databricks_table_properties", [])
             if r["deletion_vectors"].lower() not in ("true", "false")])
        chk(f"databricks_history: every VACUUM operation is {DATABRICKS_VACUUM_DONE} or VACUUM
START",
            [r["_n"] for r in pdata.get("databricks_history", [])
             if r["operation"].upper().startswith("VACUUM") and r["operation"].upper() not in
(DATABRICKS_VACUUM_DONE, "VACUUM START")])
        chk("databricks_history: timestamp is YYYY-MM-DD HH:MM:SS",
            [r["_n"] for r in pdata.get("databricks_history", []) if pts(r["timestamp"]) is None])
    if "redshift" in PLATFORMS_EXPORTED:
        snaps = pdata.get("redshift_snapshots", [])
        chk("redshift_snapshots: SnapshotCreateTime is YYYY-MM-DD HH:MM:SS and the type is manual
or automated",
            [r["_n"] for r in snaps if pts(r["snapshotcreatetime"]) is None or
r["snapshottype"].lower() not in ("manual", "automated")])
```

```
        chk("redshift_snapshots: a manual snapshot's ManualSnapshotRetentionPeriod is a whole
number or -1",
            [r["_n"] for r in snaps if r["snapshottype"].lower() == "manual"
             and not re.fullmatch(r"-?\d+", r["manualsnapshotretentionperiod"])])
        chk("redshift_snapshots: REDSHIFT_AUTOMATED_RETENTION_DAYS is set when automated snapshots
exist",
            ["setting is None"] if REDSHIFT_AUTOMATED_RETENTION_DAYS is None
            and any(r["snapshottype"].lower() == "automated" for r in snaps) else [])

    failed = [(l, b) for l, b in checks if b]
    print(f"\n== Input checks ({len(checks)} run, {len(failed)} failed)")
    for label, bad in checks:
        if bad:
            shown = ", ".join(str(x) for x in bad[:8]) + (" ..." if len(bad) > 8 else "")
            print(f"[FAIL] {label}: {'data rows' if isinstance(bad[0], int) else 'detail'}
{shown}")
        else:
            print(f"[ok]   {label}")
    incomplete = bool(failed)
    flagged = len(failed)

    # ---- usable records ---------------------------------------------------------------
    con.execute("CREATE TABLE cust(customer_id VARCHAR, last_use DATE, rel_open BOOLEAN)")
    seen = set()
    for r in custs:
        if r["customer_id"] and r["customer_id"] not in seen:
            seen.add(r["customer_id"])
            con.execute("INSERT INTO cust VALUES (?,?,?)",
                        [r["customer_id"], pdate(r["last_use_date"]),
                         {"true": True, "false": False}.get(r["relationship_open"].lower())])
    con.execute("CREATE TABLE rws(tkey VARCHAR, customer_id VARCHAR, n BIGINT)")
    seen, nokey = set(), {}
    for r in crow:
        k = rk(r)
        if k not in key_of or not re.fullmatch(r"\d+", r["row_count"]) or k + (r["customer_id"],)
in seen:
            continue
        seen.add(k + (r["customer_id"],))
        if not r["customer_id"]:
            nokey[k] = nokey.get(k, 0) + int(r["row_count"])
        elif int(r["row_count"]) > 0:
            con.execute("INSERT INTO rws VALUES (?,?,?)", [f"{k[0]}:{k[1]}", r["customer_id"],
int(r["row_count"])])

    # exceptions: in force only when the rule's reason is one of three, a basis is written, it has
not ended
    # and its review date has not passed
    exc_rows, c_exc, t_exc = [], {}, {}
    for e in sorted(excs, key=lambda x: (x["kind"].lower(), x["id"], x["exception"], x["_n"])):
        tgt = exc_target(e)
        if tgt is None or e["exception"].lower() not in EXCEPTIONS:
            continue
        ends, review = pdate(e["ends_on"]), pdate(e["review_by"])
        flags = []
        if not e["basis"]:
            flags.append("NO BASIS")
        if e["ends_on"] and ends is None:
            flags.append("END DATE UNREADABLE")
```

```
        if ends and ends < asof:
            flags.append("EXPIRED")
        if review is None or review < asof:
            flags.append("NOT REVIEWED")
        exc_rows.append([tgt, e, flags])
        if not flags:
            (c_exc if tgt[0] == "customer" else t_exc).setdefault(tgt[1], e["exception"].lower())
    con.execute("CREATE TABLE cexc(customer_id VARCHAR, exc VARCHAR)")
    for k, v in c_exc.items():
        con.execute("INSERT INTO cexc VALUES (?,?)", [k, v])
    con.execute("CREATE TABLE texc(tkey VARCHAR, exc VARCHAR)")
    for k, v in t_exc.items():
        con.execute("INSERT INTO texc VALUES (?,?)", [f"{k[0]}:{k[1]}", v])

    # ---- list 1: customers past two years ---------------------------------------------------
    sql = """
    SELECT r.customer_id, c.last_use, c.rel_open, c.customer_id IS NOT NULL AS in_register,
           c.last_use + INTERVAL 2 YEAR AS two_years,
           SUM(r.n) AS rows_total,
           SUM(CASE WHEN te.tkey IS NULL THEN r.n ELSE 0 END) AS rows_not_excepted,
           string_agg(r.tkey || '=' || CAST(r.n AS VARCHAR), '; ' ORDER BY r.tkey) AS tables,
           ANY_VALUE(ce.exc) AS cust_exc
    FROM rws r
    LEFT JOIN cust c ON c.customer_id = r.customer_id
    LEFT JOIN texc te ON te.tkey = r.tkey
    LEFT JOIN cexc ce ON ce.customer_id = r.customer_id
    GROUP BY r.customer_id, c.last_use, c.rel_open, c.customer_id, two_years
    ORDER BY r.customer_id"""
    res = con.execute(sql).fetchall()
    l1, within = [], 0
    for cid, last, op, inreg, two, tot, notex, tbls, cexc_ in res:
        two = two.date() if two is not None else None
        if not inreg:
            status = "NOT IN REGISTER"
        elif last is None:
            status = "LAST USE UNCLEAR"
        elif asof <= two:
            within += 1
            continue
        elif op is not False:
            status = "LAST USE UNCLEAR"      # an open relationship with an old last use is never
cleared
        elif cexc_ or notex == 0:
            status = "EXCEPTION IN FORCE"
        else:
            status = "PAST TWO YEARS"
        if cexc_:
            note = cexc_
        elif notex == 0:
            note = "every table has a table exception"
        else:
            note = ""
        l1.append([cid, status, str(last or ""), str(two or ""), str(tot), str(notex), tbls, note])
    n_reg = len({r["customer_id"] for r in custs if r["customer_id"]})
    with_rows = len(res)
    print(f"\nPopulation: {n_reg} customers in the register; {with_rows} customers have rows in a
register table; "
          f"{within} of them are inside two years as of {AS_OF}; {with_rows - within} are listed in
```

```
1.")

    def emit(title, fname, header, rows, filters, flag_col=None, ok_status=()):
        """Print and write one list; returns how many rows need follow-up (status not starting with
an ok word)."""
        rows = sorted(rows)
        digest = hashlib.sha256("\n".join("\x1f".join(r) for r in rows).encode("utf-
8")).hexdigest()
        tag = "  [INCOMPLETE: an input check failed]" if incomplete else ""
        print(f"\n== {title}: {fname}{tag}")
        print(f"   as of {AS_OF}; {filters}")
        print(f"   rows: {len(rows)}   sha256: {digest}")
        print("   " + " | ".join(header))
        for r in rows:
            print("   " + " | ".join(r))
        out_dir.mkdir(parents=True, exist_ok=True)
        write_csv(out_dir / f"{fname}.csv", header, rows)
        n = 0
        if flag_col is not None:
            n = sum(1 for r in rows if not r[flag_col].startswith(ok_status))
        return n

    stamp = dt.datetime.now(dt.timezone.utc).strftime("%Y%m%d-%H%M%S")
    out_dir = work / OUT_DIR / f"run-{stamp}"
    k = 1
    while out_dir.exists():
        k += 1
        out_dir = work / OUT_DIR / f"run-{stamp}-{k}"
    flagged += emit("Customers past two years", "ftc_1_customers_past_two_years",
                    ["customer_id", "status", "last_use_date", "two_years_after", "rows",
"rows_not_excepted", "tables", "exception"],
                    l1, "customers with at least one row in a register table, more than two years
after the last date of use",
                    flag_col=1, ok_status=("EXCEPTION IN FORCE",))

    # ---- list 1b: tables read by customer ------------------------------------------------------
    l1b = []
    totals = {}
    for tkey, n in con.execute("SELECT tkey, SUM(n) FROM rws GROUP BY tkey").fetchall():
        totals[tkey] = n
    ncust = dict(con.execute("SELECT tkey, COUNT(DISTINCT customer_id) FROM rws GROUP BY
tkey").fetchall())
    for (p, t), key in sorted(key_of.items()):
        tk = f"{p}:{t}"
        if key == "":
            if (p, t) in t_exc:
                status = f"NO CUSTOMER KEY, EXCEPTION IN FORCE ({t_exc[(p, t)]})"
            else:
                status = "NOT MEASURED: no customer key and no exception in force"
        elif (p, t) not in {rk(r) for r in crow}:
            status = "NOT MEASURED: no row counts for this table"
        else:
            status = "MEASURED"
            if nokey.get((p, t)):
                status += f"; {nokey[(p, t)]} ROWS WITH NO CUSTOMER KEY"
        l1b.append([p, t, key, status, str(ncust.get(tk, 0)), str(totals.get(tk, 0))])
    flagged += emit("Tables read by customer", "ftc_1b_tables_by_customer",
                    ["platform", "table_name", "customer_key_column", "status",
```

```
"customers_with_rows", "rows"], l1b,
                    "every table in table_register", flag_col=3, ok_status=("MEASURED", "NO
CUSTOMER KEY, EXCEPTION IN FORCE"))
    # a MEASURED table that also has keyless rows still needs follow-up
    flagged += sum(1 for r in l1b if r[3].startswith("MEASURED;"))

    # ---- list 2: exceptions ------------------------------------------------------------
    rows_by_c = dict(con.execute("SELECT customer_id, SUM(n) FROM rws GROUP BY
customer_id").fetchall())
    l2 = []
    for tgt, e, flags in exc_rows:
        if tgt[0] == "customer":
            held, ident = rows_by_c.get(tgt[1], 0), tgt[1]
        else:
            ident = f"{tgt[1][0]}:{tgt[1][1]}"
            held = totals.get(ident, 0) if key_of.get(tgt[1]) else None   # a table with no key is
not counted by customer
        l2.append([tgt[0], ident, e["exception"].lower(), "yes" if e["basis"] else "no",
e["ends_on"], e["review_by"],
                   ", ".join(flags) if flags else "IN FORCE", "not counted" if held is None else
str(held)])
    flagged += emit("Exceptions", "ftc_2_exceptions",
                    ["kind", "id", "exception", "basis_written", "ends_on", "review_by", "status",
"rows_held"], l2,
                    "every exception in exceptions.csv with one of the rule's three reasons; the
script never judges the stated need",
                    flag_col=6, ok_status=("IN FORCE",))

    # ---- list 3: disposed but still recoverable ----------------------------------------
-
    sf = {}
    for r in pdata.get("snowflake_tables", []):
        if not r["deleted"]:
            sf.setdefault(norm_table(f"{r['table_catalog']}.{r['table_schema']}.
{r['table_name']}"), r)
    dbp = {norm_table(r["table_name"]): r for r in pdata.get("databricks_table_properties", [])}
    vac, purge = {}, {}
    for r in pdata.get("databricks_history", []):
        if r["operation"].upper() == DATABRICKS_VACUUM_DONE and pts(r["timestamp"]):
            vac.setdefault(norm_table(r["table_name"]), []).append(pts(r["timestamp"]))
        if r["operation"].upper() in [o.upper() for o in DATABRICKS_PURGE_OPS] and
pts(r["timestamp"]):
            purge.setdefault(norm_table(r["table_name"]), []).append(pts(r["timestamp"]))
    # Redshift: what a row deleted on AS_OF can still be brought back from
    rs_days, rs_indef, rs_basis = 0, False, []
    for r in pdata.get("redshift_snapshots", []):
        created = pts(r["snapshotcreatetime"])
        if created is None or created > asof:
            continue
        if r["snapshottype"].lower() == "manual" and re.fullmatch(r"-?\d+",
r["manualsnapshotretentionperiod"]):
            n = int(r["manualsnapshotretentionperiod"])
            if n == -1:
                rs_indef = True
                rs_basis.append("a manual snapshot with retention -1 is kept indefinitely")
            elif (created + dt.timedelta(days=n) - asof).days > 0:
                left = (created + dt.timedelta(days=n) - asof).days
                rs_days = max(rs_days, left)
```

```
                rs_basis.append(f"a manual snapshot has {dn(left).lower()} left")
        elif r["snapshottype"].lower() == "automated" and REDSHIFT_AUTOMATED_RETENTION_DAYS is not
None:
            rs_days = max(rs_days, int(REDSHIFT_AUTOMATED_RETENTION_DAYS))
            rs_basis.append(f"automated snapshots up to
{dn(REDSHIFT_AUTOMATED_RETENTION_DAYS).lower()}")
    rs_basis = sorted(set(rs_basis))
    l3 = []
    for (p, t) in sorted(key_of):
        if p not in PLATFORMS_EXPORTED:
            l3.append([p, t, "", "NOT CHECKED", "platform not in PLATFORMS_EXPORTED"])
        elif p == "snowflake":
            r = sf.get(t)
            if r is None or not re.fullmatch(r"\d+", r["retention_time"]):
                l3.append([p, t, "", "NOT MEASURED", "no usable live row in snowflake_tables.csv"])
            elif r["is_transient"].upper() == "YES":
                l3.append([p, t, r["retention_time"], dn(r["retention_time"]),
                           f"transient: Time Travel {dn(r['retention_time']).lower()}, no Fail-
safe"])
            else:
                tt = max(int(r["retention_time"]), SNOWFLAKE_MIN_DATA_RETENTION_DAYS if isinstance(
                    SNOWFLAKE_MIN_DATA_RETENTION_DAYS, int) else 0)
                raised = " (raised by MIN_DATA_RETENTION_TIME_IN_DAYS)" if tt >
int(r["retention_time"]) else ""
                n = tt + FAILSAFE_DAYS
                l3.append([p, t, str(n), dn(n), f"Time Travel {dn(tt).lower()}{raised} +
{dn(FAILSAFE_DAYS).lower()} Fail-safe"])
        elif p == "databricks":
            r = dbp.get(t)
            if r is None or (r["deleted_file_retention_days"] and not re.fullmatch(r"\d+",
r["deleted_file_retention_days"])) \
                    or r["deletion_vectors"].lower() not in ("true", "false"):
                l3.append([p, t, "", "NOT MEASURED", "no usable row in
databricks_table_properties.csv"])
                continue
            n = int(r["deleted_file_retention_days"] or DATABRICKS_DEFAULT_RETENTION_DAYS)
            last = max(vac[t]) if t in vac else None
            if r["deletion_vectors"].lower() == "true":
                lp = max(purge[t]) if t in purge else None
                if lp is None:
                    l3.append([p, t, "", "DELETION VECTORS, NO PURGE IN HISTORY",
                               "a DELETE only marks rows: they stay in the table's files until
REORG TABLE ... APPLY (PURGE), "
                               f"then at least {dn(n).lower()} until a VACUUM; history is kept 30
days by default"])
                else:
                    l3.append([p, t, "", f"UNTIL THE NEXT PURGE, THEN AT LEAST {dn(n)}",
                               f"deletion vectors: rows stay until REORG TABLE ... APPLY (PURGE)
(last {lp}), then a VACUUM"])
            elif last is None:
                l3.append([p, t, str(n), f"NO VACUUM IN HISTORY, AT LEAST {dn(n)}",
                           "old files stay until a VACUUM removes them; history is kept 30 days by
default, so an older "
                           "VACUUM may not show"])
            else:
                l3.append([p, t, str(n), f"AT LEAST {dn(n)}",
                           f"until the first VACUUM after that; last VACUUM {last} ({dn((asof -
last).days).lower()} before AS_OF)"])
```

```
        else:
            if rs_indef:
                l3.append([p, t, "", "INDEFINITE", "; ".join(rs_basis)])
            elif rs_basis:
                l3.append([p, t, str(rs_days), f"UP TO {dn(rs_days)}", "; ".join(rs_basis)])
            else:
                l3.append([p, t, "0", "NO SNAPSHOT IN FORCE", "no snapshot in the export holds a
row deleted on AS_OF"])
    flagged += emit("Disposed but still recoverable", "ftc_3_recoverable_after_disposal",
                    ["platform", "table_name", "recoverable_days", "status", "basis"], l3,
                    "days a row deleted on AS_OF can still be brought back; a count to look at, not
a finding, except "
                    "INDEFINITE, DELETION VECTORS WITH NO PURGE and NOT MEASURED",
                    flag_col=3, ok_status=("INDEFINITE-NEVER",))
    # only INDEFINITE, DELETION VECTORS WITH NO PURGE and NOT MEASURED are findings here; the day
counts are for reading
    flagged -= sum(1 for r in l3 if not r[3].startswith(("INDEFINITE", "NOT MEASURED", "DELETION
VECTORS, NO PURGE")))

    # ---- list 4: retention policy reviews ----------------------------------------------------
----
    l4 = []
    print()
    if RETENTION_REVIEW_DAYS is None:
        print("== Retention policy reviews: ftc_4_retention_policy_reviews  NOT RUN:
RETENTION_REVIEW_DAYS is not set")
        out_dir.mkdir(parents=True, exist_ok=True)
        write_csv(out_dir / "ftc_4_retention_policy_reviews.csv", ["date", "days_since_previous",
"reviewer_named", "status"],
                  [[AS_OF, "", "", "NOT RUN: RETENTION_REVIEW_DAYS is not set"]])
        flagged += 1
    else:
        dates = sorted((pdate(r["review_date"]), r["reviewed_by"]) for r in revs if
pdate(r["review_date"]) and pdate(r["review_date"]) <= asof)
        prev = None
        for d, who in dates:
            gap = (d - prev).days if prev else None
            flags = []
            if gap is not None and gap > RETENTION_REVIEW_DAYS:
                flags.append(f"GAP OVER {RETENTION_REVIEW_DAYS} DAYS")
            if not who:
                flags.append("NO REVIEWER")
            l4.append([str(d), "" if gap is None else str(gap), "yes" if who else "no", ",
".join(flags) or "OK"])
            prev = d
        if not dates:
            l4.append([AS_OF, "", "", "NO REVIEW ON RECORD"])
        else:
            since = (asof - dates[-1][0]).days
            l4.append([AS_OF, str(since), "", f"OVERDUE: {since} DAYS SINCE THE LAST REVIEW" if
since > RETENTION_REVIEW_DAYS
                       else f"OK: {since} DAYS SINCE THE LAST REVIEW"])
        flagged += emit("Retention policy reviews", "ftc_4_retention_policy_reviews",
                        ["date", "days_since_previous", "reviewer_named", "status"], l4,
                        f"reviews on record; the allowed gap is your setting,
{RETENTION_REVIEW_DAYS} days; the last row is AS_OF itself",
                        flag_col=3, ok_status=("OK",))
```

```
    print(f"\nWrote the lists to {out_dir.relative_to(work)}/ (CSV, one file per list).")
    print(f"Result: {'NEEDS FOLLOW-UP' if flagged else 'nothing needs follow-up'} ({flagged}
items)")
    return 1 if (a.strict and flagged) else 0


if __name__ == "__main__":
    sys.exit(main())
```

The runs

The full output of each run, exactly as the script printed it.

Run 1: the clean generated records (--demo)

```
Demo records are GENERATED (seed 234); not a company's data.
Demo settings: AS_OF 2026-11-01, RETENTION_REVIEW_DAYS 365, REDSHIFT_AUTOMATED_RETENTION_DAYS 1

== Input checks (27 run, 0 failed)
[ok]   input files are present, readable and have every column
[ok]   customer_register: no customer_id twice or blank
[ok]   customer_register: last_use_date is YYYY-MM-DD or blank
[ok]   customer_register: no last_use_date after AS_OF
[ok]   customer_register: relationship_open is true or false
[ok]   table_register: no table twice
[ok]   table_register: platform is snowflake, databricks or redshift
[ok]   table_register: customer_key_column is a plain column name or blank
[ok]   customer_rows: every table is in table_register
[ok]   customer_rows: row_count is a whole number, 0 or more
[ok]   customer_rows: no customer counted twice for one table
[ok]   customer_rows: measured_on is YYYY-MM-DD, not after AS_OF, within 3 days of it
[ok]   customer_rows: no counts for a table the register says has no customer key
[ok]   exceptions: kind is customer or table
[ok]   exceptions: exception is one of the rule's three
[ok]   exceptions: every id is a customer in the register or the rows, or a table in table_register
[ok]   exceptions: ends_on and review_by are YYYY-MM-DD or blank
[ok]   review_log: review_date is YYYY-MM-DD and not after AS_OF
[ok]   snowflake_tables: for each register table's live row, RETENTION_TIME is a whole number and
IS_TRANSIENT is YES or NO
[ok]   SNOWFLAKE_MIN_DATA_RETENTION_DAYS is a whole number, 0 or more
[ok]   databricks_table_properties: deleted_file_retention_days is a whole number or blank
[ok]   databricks_table_properties: deletion_vectors is true or false
[ok]   databricks_history: every VACUUM operation is VACUUM END or VACUUM START
[ok]   databricks_history: timestamp is YYYY-MM-DD HH:MM:SS
[ok]   redshift_snapshots: SnapshotCreateTime is YYYY-MM-DD HH:MM:SS and the type is manual or
automated
[ok]   redshift_snapshots: a manual snapshot's ManualSnapshotRetentionPeriod is a whole number or
-1
[ok]   redshift_snapshots: REDSHIFT_AUTOMATED_RETENTION_DAYS is set when automated snapshots exist

Population: 59 customers in the register; 53 customers have rows in a register table; 51 of them
are inside two years as of 2026-11-01; 2 are listed in 1.

== Customers past two years: ftc_1_customers_past_two_years
```

```
   as of 2026-11-01; customers with at least one row in a register table, more than two years after
the last date of use
   rows: 2   sha256: 1930484dbc301a3b987908f28bdb0e110dd208b9328360442182717bb5c3d10c
   customer_id | status | last_use_date | two_years_after | rows | rows_not_excepted | tables |
exception
   C-100057 | EXCEPTION IN FORCE | 2024-05-15 | 2026-05-15 | 14 | 14 |
databricks:main.crm.customers=5; databricks:main.loans.payments=9 | required by law or regulation
   C-100058 | EXCEPTION IN FORCE | 2023-10-28 | 2025-10-28 | 46 | 46 |
databricks:main.loans.payments=37; redshift:loans.applications=9 | business operations or other
legitimate business purposes

== Tables read by customer: ftc_1b_tables_by_customer
   as of 2026-11-01; every table in table_register
   rows: 7   sha256: 3f397ab75ae02bdbcbf6f2f29168fe04d10abf6b3af8aa67e9f48a0118381543
   platform | table_name | customer_key_column | status | customers_with_rows | rows
   databricks | main.crm.customers | customer_id | MEASURED | 29 | 498
   databricks | main.loans.payments | borrower_id | MEASURED | 26 | 563
   redshift | crm.customers | customer_id | MEASURED | 26 | 489
   redshift | loans.applications | applicant_id | MEASURED | 23 | 448
   snowflake | sf_prod.crm.customers | customer_id | MEASURED | 28 | 649
   snowflake | sf_prod.loans.applications | cust_id | MEASURED | 22 | 400
   snowflake | sf_prod.ops.app_logs |  | NO CUSTOMER KEY, EXCEPTION IN FORCE (targeted disposal is
not reasonably feasible) | 0 | 0

== Exceptions: ftc_2_exceptions
   as of 2026-11-01; every exception in exceptions.csv with one of the rule's three reasons; the
script never judges the stated need
   rows: 3   sha256: cd3461b6ad9bf5b8bfb7d74fa4d62f4f0f66ff6e85f43f501b584bce552fd4b5
   kind | id | exception | basis_written | ends_on | review_by | status | rows_held
   customer | C-100057 | required by law or regulation | yes |  | 2027-03-31 | IN FORCE | 14
   customer | C-100058 | business operations or other legitimate business purposes | yes |  | 2027-
01-15 | IN FORCE | 46
   table | snowflake:sf_prod.ops.app_logs | targeted disposal is not reasonably feasible | yes |  |
2027-02-28 | IN FORCE | not counted

== Disposed but still recoverable: ftc_3_recoverable_after_disposal
   as of 2026-11-01; days a row deleted on AS_OF can still be brought back; a count to look at, not
a finding, except INDEFINITE, DELETION VECTORS WITH NO PURGE and NOT MEASURED
   rows: 7   sha256: 2e1a1036fe6156583eb8449ac5e50b43c1c7d7ccace55639ac2df6cc61aed8bf
   platform | table_name | recoverable_days | status | basis
   databricks | main.crm.customers | 7 | AT LEAST 7 DAYS | until the first VACUUM after that; last
VACUUM 2026-10-24 (8 days before AS_OF)
   databricks | main.loans.payments | 30 | AT LEAST 30 DAYS | until the first VACUUM after that;
last VACUUM 2026-10-30 (2 days before AS_OF)
   redshift | crm.customers | 28 | UP TO 28 DAYS | a manual snapshot has 28 days left; automated
snapshots up to 1 day
   redshift | loans.applications | 28 | UP TO 28 DAYS | a manual snapshot has 28 days left;
automated snapshots up to 1 day
   snowflake | sf_prod.crm.customers | 21 | 21 DAYS | Time Travel 14 days + 7 days Fail-safe
   snowflake | sf_prod.loans.applications | 37 | 37 DAYS | Time Travel 30 days + 7 days Fail-safe
   snowflake | sf_prod.ops.app_logs | 1 | 1 DAY | transient: Time Travel 1 day, no Fail-safe


== Retention policy reviews: ftc_4_retention_policy_reviews
   as of 2026-11-01; reviews on record; the allowed gap is your setting, 365 days; the last row is
AS_OF itself
   rows: 4   sha256: 3e7839c943a0b0c77505b26df2969b22ffdc38a0af27a71ef22c0a7fd3948825
   date | days_since_previous | reviewer_named | status
```

```
   2024-05-15 |  | yes | OK
   2025-04-02 | 322 | yes | OK
   2026-03-09 | 341 | yes | OK
   2026-11-01 | 237 |  | OK: 237 DAYS SINCE THE LAST REVIEW

Wrote the lists to evidence-out/run-20261101-211206/ (CSV, one file per list).
Result: nothing needs follow-up (0 items)
```

Run 2: findings planted (--demo --plant)

```
Demo records are GENERATED (seed 234); not a company's data.
Demo settings: AS_OF 2026-11-01, RETENTION_REVIEW_DAYS 365, REDSHIFT_AUTOMATED_RETENTION_DAYS 1

== Input checks (27 run, 0 failed)
[ok]   input files are present, readable and have every column
[ok]   customer_register: no customer_id twice or blank
[ok]   customer_register: last_use_date is YYYY-MM-DD or blank
[ok]   customer_register: no last_use_date after AS_OF
[ok]   customer_register: relationship_open is true or false
[ok]   table_register: no table twice
[ok]   table_register: platform is snowflake, databricks or redshift
[ok]   table_register: customer_key_column is a plain column name or blank
[ok]   customer_rows: every table is in table_register
[ok]   customer_rows: row_count is a whole number, 0 or more
[ok]   customer_rows: no customer counted twice for one table
[ok]   customer_rows: measured_on is YYYY-MM-DD, not after AS_OF, within 3 days of it
[ok]   customer_rows: no counts for a table the register says has no customer key
[ok]   exceptions: kind is customer or table
[ok]   exceptions: exception is one of the rule's three
[ok]   exceptions: every id is a customer in the register or the rows, or a table in table_register
[ok]   exceptions: ends_on and review_by are YYYY-MM-DD or blank
[ok]   review_log: review_date is YYYY-MM-DD and not after AS_OF
[ok]   snowflake_tables: for each register table's live row, RETENTION_TIME is a whole number and
IS_TRANSIENT is YES or NO
[ok]   SNOWFLAKE_MIN_DATA_RETENTION_DAYS is a whole number, 0 or more
[ok]   databricks_table_properties: deleted_file_retention_days is a whole number or blank
[ok]   databricks_table_properties: deletion_vectors is true or false
[ok]   databricks_history: every VACUUM operation is VACUUM END or VACUUM START
[ok]   databricks_history: timestamp is YYYY-MM-DD HH:MM:SS
[ok]   redshift_snapshots: SnapshotCreateTime is YYYY-MM-DD HH:MM:SS and the type is manual or
automated
[ok]   redshift_snapshots: a manual snapshot's ManualSnapshotRetentionPeriod is a whole number or
-1
[ok]   redshift_snapshots: REDSHIFT_AUTOMATED_RETENTION_DAYS is set when automated snapshots exist

Population: 67 customers in the register; 62 customers have rows in a register table; 51 of them
are inside two years as of 2026-11-01; 11 are listed in 1.

== Customers past two years: ftc_1_customers_past_two_years
   as of 2026-11-01; customers with at least one row in a register table, more than two years after
the last date of use
   rows: 11   sha256: d791670b7bd77a1514c739b01605f58b73d7f4668f83fe6dafcb09d9afc5c5b4
   customer_id | status | last_use_date | two_years_after | rows | rows_not_excepted | tables |
exception
   C-100057 | EXCEPTION IN FORCE | 2024-05-15 | 2026-05-15 | 5 | 5 |
databricks:main.crm.customers=5 | required by law or regulation
```

```
   C-100058 | EXCEPTION IN FORCE | 2023-10-28 | 2025-10-28 | 9 | 9 | redshift:loans.applications=9
| business operations or other legitimate business purposes
   C-100060 | PAST TWO YEARS | 2023-05-04 | 2025-05-04 | 67 | 67 |
databricks:main.crm.customers=37; redshift:loans.applications=30 |
   C-100061 | PAST TWO YEARS | 2022-12-19 | 2024-12-19 | 21 | 21 | redshift:loans.applications=6;
snowflake:sf_prod.crm.customers=15 |
   C-100062 | PAST TWO YEARS | 2024-10-31 | 2026-10-31 | 16 | 16 | redshift:loans.applications=16 |
   C-100063 | PAST TWO YEARS | 2024-02-29 | 2026-02-28 | 7 | 7 | redshift:loans.applications=7 |
   C-100064 | LAST USE UNCLEAR | 2023-01-15 | 2025-01-15 | 23 | 23 |
snowflake:sf_prod.loans.applications=23 |
   C-100065 | PAST TWO YEARS | 2023-08-20 | 2025-08-20 | 52 | 52 |
databricks:main.crm.customers=16; redshift:crm.customers=36 |
   C-100066 | PAST TWO YEARS | 2023-09-02 | 2025-09-02 | 1 | 1 | databricks:main.crm.customers=1 |
   C-100067 | PAST TWO YEARS | 2023-10-10 | 2025-10-10 | 42 | 42 |
databricks:main.crm.customers=12; redshift:loans.applications=30 |
   C-100099 | NOT IN REGISTER |  |  | 4 | 4 | redshift:loans.applications=4 |

== Tables read by customer: ftc_1b_tables_by_customer
   as of 2026-11-01; every table in table_register
   rows: 7   sha256: 7c17ad7b03475f12786cf1ba5e8c9fccb0bf8172a7c6b5f60a93000dc7e04b6b
   platform | table_name | customer_key_column | status | customers_with_rows | rows
   databricks | main.crm.customers | customer_id | MEASURED | 33 | 564
   databricks | main.loans.payments | borrower_id | NOT MEASURED: no row counts for this table | 0
| 0
   redshift | crm.customers | customer_id | MEASURED | 27 | 525
   redshift | loans.applications | applicant_id | MEASURED | 29 | 541
   snowflake | sf_prod.crm.customers | customer_id | MEASURED; 12 ROWS WITH NO CUSTOMER KEY | 29 |
664
   snowflake | sf_prod.loans.applications | cust_id | MEASURED | 23 | 423
   snowflake | sf_prod.ops.app_logs |  | NOT MEASURED: no customer key and no exception in force |
0 | 0

== Exceptions: ftc_2_exceptions
   as of 2026-11-01; every exception in exceptions.csv with one of the rule's three reasons; the
script never judges the stated need
   rows: 5   sha256: 77cae8f1670d90af98191d0e196573112dc7f90a1a714e2b0f513c62cc4acddc
   kind | id | exception | basis_written | ends_on | review_by | status | rows_held
   customer | C-100057 | required by law or regulation | yes |  | 2027-03-31 | IN FORCE | 5
   customer | C-100058 | business operations or other legitimate business purposes | yes |  | 2027-
01-15 | IN FORCE | 9
   customer | C-100065 | business operations or other legitimate business purposes | yes | 2026-06-
30 | 2026-12-01 | EXPIRED | 52
   customer | C-100066 | required by law or regulation | no |  | 2027-01-01 | NO BASIS | 1
   customer | C-100067 | required by law or regulation | yes |  | 2026-09-30 | NOT REVIEWED | 42

== Disposed but still recoverable: ftc_3_recoverable_after_disposal
   as of 2026-11-01; days a row deleted on AS_OF can still be brought back; a count to look at, not
a finding, except INDEFINITE, DELETION VECTORS WITH NO PURGE and NOT MEASURED
   rows: 7   sha256: 94064534853d96518c946be3beb28dac44e1e9396285d357315a2d25255ae724
   platform | table_name | recoverable_days | status | basis
   databricks | main.crm.customers |  | DELETION VECTORS, NO PURGE IN HISTORY | a DELETE only marks
rows: they stay in the table's files until REORG TABLE ... APPLY (PURGE), then at least 7 days
until a VACUUM; history is kept 30 days by default
   databricks | main.loans.payments | 30 | NO VACUUM IN HISTORY, AT LEAST 30 DAYS | old files stay
until a VACUUM removes them; history is kept 30 days by default, so an older VACUUM may not show
   redshift | crm.customers |  | INDEFINITE | a manual snapshot has 28 days left; a manual snapshot
with retention -1 is kept indefinitely; automated snapshots up to 1 day
   redshift | loans.applications |  | INDEFINITE | a manual snapshot has 28 days left; a manual
```

```
snapshot with retention -1 is kept indefinitely; automated snapshots up to 1 day
   snowflake | sf_prod.crm.customers | 21 | 21 DAYS | Time Travel 14 days + 7 days Fail-safe
   snowflake | sf_prod.loans.applications | 37 | 37 DAYS | Time Travel 30 days + 7 days Fail-safe
   snowflake | sf_prod.ops.app_logs | 1 | 1 DAY | transient: Time Travel 1 day, no Fail-safe


== Retention policy reviews: ftc_4_retention_policy_reviews
   as of 2026-11-01; reviews on record; the allowed gap is your setting, 365 days; the last row is
AS_OF itself
   rows: 3   sha256: e3a51e56ac1fd59bbcaf337b5f79b4b68142f50a0aaa722acb79ee55bea101c8
   date | days_since_previous | reviewer_named | status
   2024-05-15 |  | yes | OK
   2025-08-14 | 456 | yes | GAP OVER 365 DAYS
   2026-11-01 | 444 |  | OVERDUE: 444 DAYS SINCE THE LAST REVIEW

Wrote the lists to evidence-out/run-20261101-211207/ (CSV, one file per list).
Result: NEEDS FOLLOW-UP (20 items)
```

Run 3: faults planted in the inputs (--demo --plant-input)

```
Demo records are GENERATED (seed 234); not a company's data.
Demo settings: AS_OF 2026-11-01, RETENTION_REVIEW_DAYS 365, REDSHIFT_AUTOMATED_RETENTION_DAYS None

== Input checks (27 run, 26 failed)
[ok]   input files are present, readable and have every column
[FAIL] customer_register: no customer_id twice or blank: data rows 60
[FAIL] customer_register: last_use_date is YYYY-MM-DD or blank: data rows 61
[FAIL] customer_register: no last_use_date after AS_OF: data rows 62
[FAIL] customer_register: relationship_open is true or false: data rows 63
[FAIL] table_register: no table twice: data rows 8
[FAIL] table_register: platform is snowflake, databricks or redshift: data rows 9
[FAIL] table_register: customer_key_column is a plain column name or blank: data rows 10
[FAIL] customer_rows: every table is in table_register: data rows 155
[FAIL] customer_rows: row_count is a whole number, 0 or more: data rows 156
[FAIL] customer_rows: no customer counted twice for one table: data rows 157
[FAIL] customer_rows: measured_on is YYYY-MM-DD, not after AS_OF, within 3 days of it: data rows 2
[FAIL] customer_rows: no counts for a table the register says has no customer key: data rows 158
[FAIL] exceptions: kind is customer or table: data rows 4
[FAIL] exceptions: exception is one of the rule's three: data rows 5
[FAIL] exceptions: every id is a customer in the register or the rows, or a table in
table_register: data rows 4, 6
[FAIL] exceptions: ends_on and review_by are YYYY-MM-DD or blank: data rows 7
[FAIL] review_log: review_date is YYYY-MM-DD and not after AS_OF: data rows 4
[FAIL] snowflake_tables: for each register table's live row, RETENTION_TIME is a whole number and
IS_TRANSIENT is YES or NO: data rows 2
[FAIL] SNOWFLAKE_MIN_DATA_RETENTION_DAYS is a whole number, 0 or more: detail setting
[FAIL] databricks_table_properties: deleted_file_retention_days is a whole number or blank: data
rows 2
[FAIL] databricks_table_properties: deletion_vectors is true or false: data rows 1
[FAIL] databricks_history: every VACUUM operation is VACUUM END or VACUUM START: data rows 5
[FAIL] databricks_history: timestamp is YYYY-MM-DD HH:MM:SS: data rows 6
[FAIL] redshift_snapshots: SnapshotCreateTime is YYYY-MM-DD HH:MM:SS and the type is manual or
automated: data rows 3
[FAIL] redshift_snapshots: a manual snapshot's ManualSnapshotRetentionPeriod is a whole number or
-1: data rows 4
[FAIL] redshift_snapshots: REDSHIFT_AUTOMATED_RETENTION_DAYS is set when automated snapshots exist:
```

```
detail setting is None

Population: 62 customers in the register; 53 customers have rows in a register table; 51 of them
are inside two years as of 2026-11-01; 2 are listed in 1.

== Customers past two years: ftc_1_customers_past_two_years  [INCOMPLETE: an input check failed]
   as of 2026-11-01; customers with at least one row in a register table, more than two years after
the last date of use
   rows: 2   sha256: 1930484dbc301a3b987908f28bdb0e110dd208b9328360442182717bb5c3d10c
   customer_id | status | last_use_date | two_years_after | rows | rows_not_excepted | tables |
exception
   C-100057 | EXCEPTION IN FORCE | 2024-05-15 | 2026-05-15 | 14 | 14 |
databricks:main.crm.customers=5; databricks:main.loans.payments=9 | required by law or regulation
   C-100058 | EXCEPTION IN FORCE | 2023-10-28 | 2025-10-28 | 46 | 46 |
databricks:main.loans.payments=37; redshift:loans.applications=9 | business operations or other
legitimate business purposes

== Tables read by customer: ftc_1b_tables_by_customer  [INCOMPLETE: an input check failed]
   as of 2026-11-01; every table in table_register
   rows: 9   sha256: f6b86a85ff899928de5e0ef04c68afba28a148dabd0a05bc4918dfb1baf80a47
   platform | table_name | customer_key_column | status | customers_with_rows | rows
   databricks | main.crm.customers | customer_id | MEASURED | 29 | 498
   databricks | main.loans.payments | borrower_id | MEASURED | 26 | 563
   redshift | crm.customers | customer_id | MEASURED | 26 | 489
   redshift | crm.keyed_oddly | cust id | NOT MEASURED: no row counts for this table | 0 | 0
   redshift | loans.applications | applicant_id | MEASURED | 23 | 448
   snowflak | sf_prod.x.y | id | NOT MEASURED: no row counts for this table | 0 | 0
   snowflake | sf_prod.crm.customers | customer_id | MEASURED | 28 | 649
   snowflake | sf_prod.loans.applications | cust_id | MEASURED | 22 | 400
   snowflake | sf_prod.ops.app_logs |  | NO CUSTOMER KEY, EXCEPTION IN FORCE (targeted disposal is
not reasonably feasible) | 1 | 2

== Exceptions: ftc_2_exceptions  [INCOMPLETE: an input check failed]
   as of 2026-11-01; every exception in exceptions.csv with one of the rule's three reasons; the
script never judges the stated need
   rows: 5   sha256: 44ffa1a0cf12024a0e8b3f97326594aef95a34b8f38615200616392c5fed5857
   kind | id | exception | basis_written | ends_on | review_by | status | rows_held
   customer | C-100003 | required by law or regulation | yes | 31/12/2026 | 2027-01-01 | END DATE
UNREADABLE | 49
   customer | C-100057 | required by law or regulation | yes |  | 2027-03-31 | IN FORCE | 14
   customer | C-100058 | business operations or other legitimate business purposes | yes |  | 2027-
01-15 | IN FORCE | 46
   customer | C-999999 | required by law or regulation | yes |  | 2027-01-01 | IN FORCE | 0
   table | snowflake:sf_prod.ops.app_logs | targeted disposal is not reasonably feasible | yes |  |
2027-02-28 | IN FORCE | not counted

== Disposed but still recoverable: ftc_3_recoverable_after_disposal  [INCOMPLETE: an input check
failed]
   as of 2026-11-01; days a row deleted on AS_OF can still be brought back; a count to look at, not
a finding, except INDEFINITE, DELETION VECTORS WITH NO PURGE and NOT MEASURED
   rows: 9   sha256: b616409242954c912dec7e7bfb1e806551d839825171e549a7fccf14113db721
   platform | table_name | recoverable_days | status | basis
   databricks | main.crm.customers |  | NOT MEASURED | no usable row in
databricks_table_properties.csv
   databricks | main.loans.payments |  | NOT MEASURED | no usable row in
databricks_table_properties.csv
   redshift | crm.customers | 28 | UP TO 28 DAYS | a manual snapshot has 28 days left
   redshift | crm.keyed_oddly | 28 | UP TO 28 DAYS | a manual snapshot has 28 days left
```

```
   redshift | loans.applications | 28 | UP TO 28 DAYS | a manual snapshot has 28 days left
   snowflak | sf_prod.x.y |  | NOT CHECKED | platform not in PLATFORMS_EXPORTED
   snowflake | sf_prod.crm.customers | 21 | 21 DAYS | Time Travel 14 days + 7 days Fail-safe
   snowflake | sf_prod.loans.applications |  | NOT MEASURED | no usable live row in
snowflake_tables.csv
   snowflake | sf_prod.ops.app_logs | 1 | 1 DAY | transient: Time Travel 1 day, no Fail-safe


== Retention policy reviews: ftc_4_retention_policy_reviews  [INCOMPLETE: an input check failed]
   as of 2026-11-01; reviews on record; the allowed gap is your setting, 365 days; the last row is
AS_OF itself
   rows: 4   sha256: 3e7839c943a0b0c77505b26df2969b22ffdc38a0af27a71ef22c0a7fd3948825
   date | days_since_previous | reviewer_named | status
   2024-05-15 |  | yes | OK
   2025-04-02 | 322 | yes | OK
   2026-03-09 | 341 | yes | OK
   2026-11-01 | 237 |  | OK: 237 DAYS SINCE THE LAST REVIEW

Wrote the lists to evidence-out/run-20261101-211207/ (CSV, one file per list).
Result: NEEDS FOLLOW-UP (32 items)
```

Run 4: a missing column and a missing file (--demo --plant-files)

```
Demo records are GENERATED (seed 234); not a company's data.
Demo settings: AS_OF 2026-11-01, RETENTION_REVIEW_DAYS 365, REDSHIFT_AUTOMATED_RETENTION_DAYS 1

== Input checks (27 run, 1 failed)
[FAIL] input files are present, readable and have every column: detail review_log.csv: missing
column reviewed_by, snowflake_tables.csv: not found
[ok]   customer_register: no customer_id twice or blank
[ok]   customer_register: last_use_date is YYYY-MM-DD or blank
[ok]   customer_register: no last_use_date after AS_OF
[ok]   customer_register: relationship_open is true or false
[ok]   table_register: no table twice
[ok]   table_register: platform is snowflake, databricks or redshift
[ok]   table_register: customer_key_column is a plain column name or blank
[ok]   customer_rows: every table is in table_register
[ok]   customer_rows: row_count is a whole number, 0 or more
[ok]   customer_rows: no customer counted twice for one table
[ok]   customer_rows: measured_on is YYYY-MM-DD, not after AS_OF, within 3 days of it
[ok]   customer_rows: no counts for a table the register says has no customer key
[ok]   exceptions: kind is customer or table
[ok]   exceptions: exception is one of the rule's three
[ok]   exceptions: every id is a customer in the register or the rows, or a table in table_register
[ok]   exceptions: ends_on and review_by are YYYY-MM-DD or blank
[ok]   review_log: review_date is YYYY-MM-DD and not after AS_OF
[ok]   snowflake_tables: for each register table's live row, RETENTION_TIME is a whole number and
IS_TRANSIENT is YES or NO
[ok]   SNOWFLAKE_MIN_DATA_RETENTION_DAYS is a whole number, 0 or more
[ok]   databricks_table_properties: deleted_file_retention_days is a whole number or blank
[ok]   databricks_table_properties: deletion_vectors is true or false
[ok]   databricks_history: every VACUUM operation is VACUUM END or VACUUM START
[ok]   databricks_history: timestamp is YYYY-MM-DD HH:MM:SS
[ok]   redshift_snapshots: SnapshotCreateTime is YYYY-MM-DD HH:MM:SS and the type is manual or
automated
[ok]   redshift_snapshots: a manual snapshot's ManualSnapshotRetentionPeriod is a whole number or
```

```
-1
[ok]   redshift_snapshots: REDSHIFT_AUTOMATED_RETENTION_DAYS is set when automated snapshots exist

Population: 59 customers in the register; 53 customers have rows in a register table; 51 of them
are inside two years as of 2026-11-01; 2 are listed in 1.

== Customers past two years: ftc_1_customers_past_two_years  [INCOMPLETE: an input check failed]
   as of 2026-11-01; customers with at least one row in a register table, more than two years after
the last date of use
   rows: 2   sha256: 1930484dbc301a3b987908f28bdb0e110dd208b9328360442182717bb5c3d10c
   customer_id | status | last_use_date | two_years_after | rows | rows_not_excepted | tables |
exception
   C-100057 | EXCEPTION IN FORCE | 2024-05-15 | 2026-05-15 | 14 | 14 |
databricks:main.crm.customers=5; databricks:main.loans.payments=9 | required by law or regulation
   C-100058 | EXCEPTION IN FORCE | 2023-10-28 | 2025-10-28 | 46 | 46 |
databricks:main.loans.payments=37; redshift:loans.applications=9 | business operations or other
legitimate business purposes

== Tables read by customer: ftc_1b_tables_by_customer  [INCOMPLETE: an input check failed]
   as of 2026-11-01; every table in table_register
   rows: 7   sha256: 3f397ab75ae02bdbcbf6f2f29168fe04d10abf6b3af8aa67e9f48a0118381543
   platform | table_name | customer_key_column | status | customers_with_rows | rows
   databricks | main.crm.customers | customer_id | MEASURED | 29 | 498
   databricks | main.loans.payments | borrower_id | MEASURED | 26 | 563
   redshift | crm.customers | customer_id | MEASURED | 26 | 489
   redshift | loans.applications | applicant_id | MEASURED | 23 | 448
   snowflake | sf_prod.crm.customers | customer_id | MEASURED | 28 | 649
   snowflake | sf_prod.loans.applications | cust_id | MEASURED | 22 | 400
   snowflake | sf_prod.ops.app_logs |  | NO CUSTOMER KEY, EXCEPTION IN FORCE (targeted disposal is
not reasonably feasible) | 0 | 0

== Exceptions: ftc_2_exceptions  [INCOMPLETE: an input check failed]
   as of 2026-11-01; every exception in exceptions.csv with one of the rule's three reasons; the
script never judges the stated need
   rows: 3   sha256: cd3461b6ad9bf5b8bfb7d74fa4d62f4f0f66ff6e85f43f501b584bce552fd4b5
   kind | id | exception | basis_written | ends_on | review_by | status | rows_held
   customer | C-100057 | required by law or regulation | yes |  | 2027-03-31 | IN FORCE | 14
   customer | C-100058 | business operations or other legitimate business purposes | yes |  | 2027-
01-15 | IN FORCE | 46
   table | snowflake:sf_prod.ops.app_logs | targeted disposal is not reasonably feasible | yes |  |
2027-02-28 | IN FORCE | not counted

== Disposed but still recoverable: ftc_3_recoverable_after_disposal  [INCOMPLETE: an input check
failed]
   as of 2026-11-01; days a row deleted on AS_OF can still be brought back; a count to look at, not
a finding, except INDEFINITE, DELETION VECTORS WITH NO PURGE and NOT MEASURED
   rows: 7   sha256: 48e5a6795a1c292c5adb29ebc096cf80502a3a1a601aeaf3eb77ab308d0719fb
   platform | table_name | recoverable_days | status | basis
   databricks | main.crm.customers | 7 | AT LEAST 7 DAYS | until the first VACUUM after that; last
VACUUM 2026-10-24 (8 days before AS_OF)
   databricks | main.loans.payments | 30 | AT LEAST 30 DAYS | until the first VACUUM after that;
last VACUUM 2026-10-30 (2 days before AS_OF)
   redshift | crm.customers | 28 | UP TO 28 DAYS | a manual snapshot has 28 days left; automated
snapshots up to 1 day
   redshift | loans.applications | 28 | UP TO 28 DAYS | a manual snapshot has 28 days left;
automated snapshots up to 1 day
   snowflake | sf_prod.crm.customers |  | NOT MEASURED | no usable live row in snowflake_tables.csv
   snowflake | sf_prod.loans.applications |  | NOT MEASURED | no usable live row in
```

```
snowflake_tables.csv
   snowflake | sf_prod.ops.app_logs |  | NOT MEASURED | no usable live row in snowflake_tables.csv


== Retention policy reviews: ftc_4_retention_policy_reviews  [INCOMPLETE: an input check failed]
   as of 2026-11-01; reviews on record; the allowed gap is your setting, 365 days; the last row is
AS_OF itself
   rows: 1   sha256: 7152d0faae79069db50135764d3893e2b4b51efadc0e84836544c1698b19fc1f
   date | days_since_previous | reviewer_named | status
   2026-11-01 |  |  | NO REVIEW ON RECORD

Wrote the lists to evidence-out/run-20261101-211207/ (CSV, one file per list).
Result: NEEDS FOLLOW-UP (5 items)
```

The row-count queries (--demo --queries)

```
Demo records are GENERATED (seed 234); not a company's data.
Demo settings: AS_OF 2026-11-01, RETENTION_REVIEW_DAYS 365, REDSHIFT_AUTOMATED_RETENTION_DAYS 1
SELECT 'snowflake' AS platform, 'sf_prod.crm.customers' AS table_name, TRIM(CAST(customer_id AS
VARCHAR)) AS customer_id,
       COUNT(*) AS row_count, CURRENT_DATE AS measured_on
FROM sf_prod.crm.customers
GROUP BY 1, 2, 3;

SELECT 'snowflake' AS platform, 'sf_prod.loans.applications' AS table_name, TRIM(CAST(cust_id AS
VARCHAR)) AS customer_id,
       COUNT(*) AS row_count, CURRENT_DATE AS measured_on
FROM sf_prod.loans.applications
GROUP BY 1, 2, 3;

SELECT 'databricks' AS platform, 'main.crm.customers' AS table_name, TRIM(CAST(customer_id AS
STRING)) AS customer_id,
       COUNT(*) AS row_count, CURRENT_DATE AS measured_on
FROM main.crm.customers
GROUP BY 1, 2, 3;

SELECT 'databricks' AS platform, 'main.loans.payments' AS table_name, TRIM(CAST(borrower_id AS
STRING)) AS customer_id,
       COUNT(*) AS row_count, CURRENT_DATE AS measured_on
FROM main.loans.payments
GROUP BY 1, 2, 3;

SELECT 'redshift' AS platform, 'crm.customers' AS table_name, TRIM(CAST(customer_id AS VARCHAR)) AS
customer_id,
       COUNT(*) AS row_count, CURRENT_DATE AS measured_on
FROM crm.customers
GROUP BY 1, 2, 3;

SELECT 'redshift' AS platform, 'loans.applications' AS table_name, TRIM(CAST(applicant_id AS
VARCHAR)) AS customer_id,
       COUNT(*) AS row_count, CURRENT_DATE AS measured_on
FROM loans.applications
GROUP BY 1, 2, 3;
```

How to read the results

- **INCOMPLETE** on a list means an input check failed, so the list may be missing rows. Fix the export before you show it to anyone.
- **PAST TWO YEARS** is a customer more than two years after the last date of use, with rows left, and no exception in force. The rows are the counts as of the day the query ran.
- **LAST USE UNCLEAR** is a customer whose relationship is still open but whose last-use date is old, or whose date is blank. The script does not clear it. Your system of record has to say whether the information was used.
- **NOT IN REGISTER** is a customer id in the platform rows that your register does not know. **ROWS WITH NO CUSTOMER KEY** are rows no customer's clock can be applied to.
- **NOT MEASURED** is a table that was not counted, or has no customer key and no exception in force: the lists cannot say it is clean.
- **EXCEPTION IN FORCE** means an exception with a written basis that has not ended and has not passed its review date. It is your record of why, and the script has not judged whether the reason is real.
- **NO BASIS, EXPIRED, NOT REVIEWED, END DATE UNREADABLE** are exceptions that do not count as in force. NOT REVIEWED includes an exception with no review date. In list 1, `rows_not_excepted` counts the rows outside any table exception; a customer's own exception covers all of them.
- **DELETION VECTORS, NO PURGE IN HISTORY** is a Databricks table whose deleted rows stay in its files until someone runs `REORG TABLE ... APPLY (PURGE)`, and the history shows none. **INDEFINITE** is a Redshift manual snapshot with no end date, so rows deleted from the table would still be held in it, with no end date, if the snapshot was taken before the delete. The other recoverable counts are for reading: a table whose days are longer than you expected is a question for whoever sets that table's retention.
- **OVERDUE** and **GAP OVER** compare your review dates with the gap you set. The rule sets none. NO REVIEWER is a review with nobody named. With `RETENTION_REVIEW_DAYS` unset, list 4 prints NOT RUN and writes a CSV that says so.
- A clean run on your own exports means only that those exports agree with your register and your exceptions. It says nothing about copies outside the three platforms. Keep each run's settings, its SHA-256 lines and the CSVs it wrote. Your lawyer or assessor decides what they show.

Run it on your own data

To run it on your own data, change only the settings block at the top of the download. Set EXPORT_DIR to the folder holding your exports. Set AS_OF to the day you took them. Set PLATFORMS_EXPORTED to the platforms you exported. Set RETENTION_REVIEW_DAYS to the longest gap between retention policy reviews that you allow. If you export Redshift, set REDSHIFT_AUTOMATED_RETENTION_DAYS to your cluster's automated snapshot retention in days. Write your customer register, table register, exceptions and review log as the CSV files named in FILES, with the column names the download lists, and save the answers of the queries that --queries prints as customer_rows.csv. Export each platform's own records with the platform's own column names. If a file has another name, change its entry in FILES. Change nothing else, and run it without --demo.

We tested those steps on a second, separate set of generated exports: another file name for the customer register and the review log, other schema, table and column names, names in mixed case, a quoted table name, a blank customer id where the key was NULL, a customer with no last-use date, a dropped Snowflake table of the same name, a Snowflake view in the TABLES export, and Databricks left out of PLATFORMS_EXPORTED. We also ran `--queries` on that table register: it printed a query for each keyed table, with the quoted Snowflake name kept quoted. The script reported Databricks as NOT CHECKED, listed the customers past two years, the open relationship, the blank last-use date and the customer not in the register, flagged the expired exception, the rows with no customer key, the table with no customer key and no exception and the snapshot with no end date, reported the overdue review, and ran to the end. That test is also generated data.

Download safeguards_disposal_evidence.py · Download the full guide (PDF)

Method

- **Input:** generated records written by the script as CSV (seed 234), with the platforms' published column names where they exist (the links in step 1). Customers, tables, exceptions and reviews are invented. Customer ids look like C-100057 and stand for no one.
- **As-of day:** 2026-11-01. Our counting, not the rule's: two years after a last-use date is the same calendar date two years on, by DuckDB's date arithmetic. A customer is past two years only when the as-of day is after that date. Row counts must have been taken within three days of the as-of day.
- **Recoverable days, for a row deleted on the as-of day:** Snowflake from the table's settings (the larger of its retention time and your account minimum, plus 7 days of Fail-safe unless the table is transient); Databricks as at least the table's retention (7 days if it sets none), ending at the first VACUUM after that, with the last VACUUM from history shown, and on a table with deletion vectors, only after a REORG ... APPLY (PURGE); Redshift from the snapshots in the export: a manual snapshot with retention -1 is kept indefinitely, one with a number of days

ends that many days after it was created, and automated snapshots count as up to your setting `REDSHIFT_AUTOMATED_RETENTION_DAYS` (AWS CLI: describe-cluster-snapshots (read 2026-11-01)). The script reads records; it never measures bytes held.

- **Demo settings of ours:** the review gap is 365 days and the Redshift automated retention is one day. The rule sets neither, and the demo's values are not advice.
- **Fingerprints:** each list is sorted on every column before it is hashed, so the same input gives the same SHA-256 on every run. We ran the clean records three times and all five hashes matched.
- **Our assumptions, to check against one of your own rows:** Databricks history calls a finished VACUUM `VACUUM END` and a purge `REORG`; both are settings. The history schema page does not list VACUUM's operation names (Databricks: Table history schema (read 2026-11-01)), so it is a setting, and an unknown operation starting VACUUM fails an input check.

Checklist

Done	Item
☐	Customer register exported from the system of record, with last-use dates and open or closed
☐	Table register written: every table with customer information, and its customer key column
☐	Row-count query run on every keyed table and saved as customer_rows.csv
☐	Exceptions written with one of the rule's three reasons, a basis, an end and a review date
☐	Retention policy review dates recorded, and RETENTION_REVIEW_DAYS chosen
☐	Platform recovery records exported (Snowflake, Databricks, Redshift)
☐	Run with every input check passing, or each failure explained
☐	Every customer listed as PAST TWO YEARS, LAST USE UNCLEAR or NOT IN REGISTER followed up
☐	Every NOT MEASURED table counted, or given an exception
☐	Every INDEFINITE snapshot reviewed by its owner
☐	The run's settings, SHA-256 lines and CSVs kept

Limits

- **This is not a client result.** Every record here is generated. We have not run the script on a real Snowflake, Databricks or Redshift account, and we did not run the row-count queries on a platform.

- **Not legal advice, and not a verdict.** We quote the rule and do not decide whether you are a financial institution within the scope in 314.1(b), what your business needs are, or what an assessor will accept. The script produces evidence.
- **The last date of use is yours.** The script cannot know when a customer was last served, or what counts as "used". It trusts your register, and an open relationship is never cleared by it.
- **Exceptions are your record.** The script checks that an exception has one of the three reasons, a written basis, a review date that has not passed, and, if it has an end date, that the date has not passed. It never judges whether the stated need is real.
- **Row counts** are as good as the query and the day it ran. Rows with no customer key cannot be aged. Copies outside the three platforms (BI extracts, files, backups made by other systems) are not seen.
- **Recovery days** come from settings and snapshot lists, not bytes held. Fail-safe is recoverable by Snowflake only (Snowflake: Fail-safe (read 2026-11-01)); the script counts it. A Databricks VACUUM run with a shorter retention than the table's property could remove files earlier than assumed. Databricks history keeps 30 days by default, so an older VACUUM or REORG may not show. Snapshots come from `describe-cluster-snapshots`, which covers provisioned clusters, and every snapshot listed is counted, whatever its status; Redshift Serverless snapshots are not read, and the script cannot see tables created with BACKUP NO, so it counts a snapshot against every Redshift table (Amazon Redshift: snapshots and backups (read 2026-11-01)).
- **Reviews:** the script reads dates and whether someone is named. It does not judge the quality of a review or what the policy says.
- **Tested on duckdb 1.5.6 and 2.0.0,** with the same output on both. Other versions may behave differently.

Tested on

Platform
: DuckDB (the script reads exported CSV files; it never connects to Snowflake, Databricks or Redshift)

Edition
: duckdb 1.5.6 and 2.0.0 (the same output on both); Python 3

Date
: 2026-11-01

Dataset
: Generated records (seed 234) in each platform's published column names; not a company's data

When this is not enough

- Copies outside the three platforms: exports, extracts, files in cloud storage, logs and backups made by other systems. The lists see only what the three platforms record.
- Information in other forms. The rule says "in any format", and this script reads three warehouses.
- A table that holds customer information and is not in your register. The register is only as good as its list of tables.
- A single day. One run shows one day; rows come back and customers are served again. Run it on a schedule and keep every run's lists and hashes.
- Deciding what the exceptions should say and how often to review the policy. That is for you and your lawyer.

Sources

- AWS CLI: describe-cluster-snapshots: https://docs.aws.amazon.com/cli/latest/reference/redshift/describe-cluster-snapshots.html (read 2026-11-01)
- Amazon Redshift: snapshots and backups: https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-snapshots.html (read 2026-11-01)
- Databricks: Remove unused data files with VACUUM: https://docs.databricks.com/aws/en/tables/operations/vacuum (read 2026-11-01)
- Databricks: Table history schema: https://docs.databricks.com/aws/en/tables/history-schema (read 2026-11-01)
- Databricks: VACUUM (SQL): https://docs.databricks.com/aws/en/sql/language-manual/delta-vacuum (read 2026-11-01)
- Databricks: Work with table history: https://docs.databricks.com/aws/en/delta/history (read 2026-11-01)
- Snowflake: ACCOUNT_USAGE.TABLES: https://docs.snowflake.com/en/sql-reference/account-usage/tables (read 2026-11-01)
- Snowflake: Fail-safe: https://docs.snowflake.com/en/user-guide/data-failsafe (read 2026-11-01)
- Snowflake: Time Travel: https://docs.snowflake.com/en/user-guide/data-time-travel (read 2026-11-01)

- Snowflake: temporary and transient tables: https://docs.snowflake.com/en/user-guide/tables-temp-transient (read 2026-11-01)
- eCFR page for part 314: https://www.ecfr.gov/current/title-16/chapter-I/subchapter-C/part-314 (read 2026-11-01)
- eCFR: 16 CFR 314.4: https://www.ecfr.gov/current/title-16/section-314.4 (read 2026-11-01)

See Data Control Tower on a fictional company's data

The code in this guide is free and needs nothing from us to run. Separately, Data Control Tower deploys into your own Azure subscription from a template your IT team can read before it runs. Every day it reads the records your data platforms already keep, and it never writes to them. It covers Amazon Redshift, Databricks and Snowflake, and no platform passwords are stored.

We'll walk you through it on a fictional company's data, then show what you would share to see your own.

https://www.datameaning.com/solutions/data-control-tower.html