DATA CONTROL TOWER · GUIDE

PCI DSS 7.2.4 access review: Snowflake, Databricks, Redshift

The review packet, review timing, coverage and remove decisions checked against the platform records, from Snowflake, Databricks and Redshift exports.

Published 2026-11-01 · Online version and downloads: www.datameaning.com/guides/pci-dss-access-review-evidence-for-snowflake-databricks-redshift/

The short answer. The PCI DSS 7.2.4 review needs a record that user accounts and their access were reviewed at least every six months, wrong access was dealt with, and management confirmed what remains. Our free script reads Snowflake, Databricks and Redshift exports and prints four lists: the review packet, timing, coverage and decisions carried out. That is evidence; the assessor decides.

What 7.2.4 asks for, in our words

PCI DSS v4.0.1 (June 2024) is the Payment Card Industry Data Security Standard. Its text is licensed by the PCI Security Standards Council, so this page names requirement 7.2.4 by its id and describes it in our own words; it does not quote it. Get the standard from the PCI Security Standards Council: document library (read 2026-11-01) and read 7.2.4, its testing procedures and the notes around it yourself. What the text supports:

- **Who and how often.** Every user account, and the access tied to it, third-party and vendor accounts included, is reviewed at least once every six months.
- **What the review decides.** Whether the access still fits the person's job. Access found to be wrong is dealt with, and management confirms the access that remains is still appropriate.
- **What an assessor reads.** The testing procedures have the assessor read the review policy and the documented results of past reviews, and talk to the people who run them. So the evidence is the review's own record: what was reviewed, when, by whom, and what came of it.

- **Not 7.2.4.** Application and system accounts fall under 7.2.5 and 7.2.5.1, and under 8.6.1 to 8.6.3. The script lists them apart so none drops out of sight.
- **What "every six months" means.** Section 7, Table 4 of the standard defines six months as a gap of no more than 180 to 184 days, or the same day of the month six months later. The script's REVIEW_INTERVAL_DAYS is 184, the longest gap that table allows. The same section says a late activity is not automatically a failure where a documented process caught and handled it. It also says that in a first assessment against the requirement, a review in the most recent six-month period is enough, provided you also have documented policies and procedures for keeping to the schedule; the assessor checks both. The assessor decides about a late review; the script only shows the dates.
- **What the standard leaves to you.** It sets no deadline for addressing wrong access. So REMEDIATION_DAYS has no default: until you set it, list 4 does not run.
- **Since when.** The requirement became mandatory after 31 March 2025.

A review is something people do. A data platform keeps the lists they review and the access they change. This page shows how to read those lists, and how to check the review's own record against them. It does not run the review, and it does not say any review was enough. Neighbors already published: when a leaver's access ended and who approved a grant are in the SOX access guide; the user listing, administrators and sign-ins are in the SOC 2 user access guide; inactive, temporary and transferred accounts are in the NIST AC-2 guide. This one is about the review itself.

What you will have at the end

A Python script, pci_access_review_evidence.py, that reads CSV exports of each platform's account, grant and audit records, with column names taken from the platforms' documentation (Databricks names written in snake_case), plus lists you keep and the review's own record. It runs 18 input checks first (fewer if you leave a platform out), then prints four lists, each with its period, its filters, the run time, its row count and a SHA-256 of its rows, and writes each as a CSV. A platform you do not export is printed as not checked. The details are also in a PDF: Download the full guide (PDF).

The four lists

List	What the review needs	What it flags
1. The review packet	What the reviewer is given to review	Every user account on each platform with each role or group it holds, direct and inherited, its owner, department and job function. NO OWNER flags an account that matches no employee and no third party. Each platform's packet gets a SHA-256, which the review records. List 1b shows system accounts apart (7.2.5 and 7.2.5.1).

List	What the review needs	What it flags
2. Review timing	At least every six months, acknowledged for management	Each completed review per platform, with the days since the one before. GAP OVER 184 DAYS, OVERDUE (the last review is more than 184 days before the export), NO REVIEW RECORDED, NO ACKNOWLEDGMENT.
3. Review coverage	The review covered everything it was given	The latest review against the packet it was given. NOT REVIEWED (a packet row with no decision), REVIEWED OWN ACCESS (the reviewer is the account's owner), NO REVIEWER.
4. Decisions carried out	Wrong access is dealt with	Every remove decision, whether the access is still held now, and when it went where the platform records that. STILL HELD PAST DEADLINE and REMOVED LATE are the findings.

The records we ran it on are **generated**: fixed rows, 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. The script produces evidence; it does not say a requirement is met or that anyone will accept it.

What you will need

- Snowflake: `ACCOUNT_USAGE.USERS` has `NAME`, `TYPE`, `DISABLED`, `CREATED_ON`, `DELETED_ON` and `LAST_SUCCESS_LOGIN` (Snowflake: ACCOUNT_USAGE.USERS (read 2026-11-01)), and `GRANTS_TO_USERS` has `ROLE`, `GRANTEE_NAME` and `DELETED_ON`, which is NULL while the grant is active (Snowflake: ACCOUNT_USAGE.GRANTS_TO_USERS (read 2026-11-01)). Both can lag by up to 120 minutes. The user types, including the service types the script lists apart, are described on Snowflake: user management (read 2026-11-01): SERVICE, SERVICE_AGENT, LEGACY_SERVICE and SNOWFLAKE_SERVICE, and an empty type works like PERSON. The views are in the `SNOWFLAKE` database, whose schemas an ACCOUNTADMIN must grant you access to, and their history is kept for 365 days (Snowflake: Account Usage (read 2026-11-01)).
- Snowflake role inheritance: `GRANTS_TO_ROLES` rows with `GRANTED_ON` = ROLE (Snowflake: ACCOUNT_USAGE.GRANTS_TO_ROLES (read 2026-11-01)). The page says `NAME` is the object the privilege is granted on, here the role granted, and `GRANTEE_NAME` the recipient role. Only a USAGE row is inheritance: Snowflake says a role's owner, the role with the OWNERSHIP privilege on it, does not inherit the owned role's privileges (Snowflake: overview of access control (read 2026-11-01)), so the script reads `PRIVILEGE` and skips OWNERSHIP rows. Check one row of your own before you rely on the inherited rows.
- Databricks: account users with `id`, `userName` and `active` (Databricks: list account users (read 2026-11-01)); service principals with `id`, `applicationId`, `displayName` and `active` (Databricks: list account service principals (read 2026-11-01)); and each group with its `id`,

`displayName` and `members`, which the list call no longer returns, so you get them one group at a time (Databricks: get account group (read 2026-11-01)). The page does not say whether a member can be a group; the script treats a member whose value is another group's id as a nested group and counts members it cannot place. For dates it reads `system.access.audit` (Databricks: audit log system table (read 2026-11-01)): account-level events carry `workspace_id` 0. The actions `removePrincipalFromGroup`, `removePrincipalsFromGroup` and `delete` are the ones the script reads; all three are on the audit log reference (Databricks: audit log reference (read 2026-11-01); Azure Databricks: audit log reference (read 2026-11-01)), where `removePrincipalsFromGroup` is how SCIM provisioning records removals. Databricks keeps the table for 365 days (Databricks: system tables overview (read 2026-11-01)), and most audit logs are only available in the region of the workspace (Databricks: audit log system table (read 2026-11-01)).

- Redshift: `SVV_USER_INFO` (`user_name`, `user_id`, `superuser`; all rows only for superusers) (Amazon Redshift: SVV_USER_INFO (read 2026-11-01)); `SVV_USER_GRANTS` (roles explicitly granted to users) (Amazon Redshift: SVV_USER_GRANTS (read 2026-11-01)) and `SVV_ROLE_GRANTS` (roles explicitly granted to roles) (Amazon Redshift: SVV_ROLE_GRANTS (read 2026-11-01)), both visible in full to superusers and users with the ACCESS SYSTEM TABLE permission; and `SYS_USERLOG`, visible only to superusers (Amazon Redshift: SYS_USERLOG (read 2026-11-01)). Its page gives no retention period. AWS's reference says “STL system views retain seven days of log history” (Amazon Redshift: system tables and views reference (read 2026-11-01)), so plan for about seven days of `SYS_USERLOG` unless you keep copies. These views list grants, not who owns what, and a role revoke has no date in them.
- Your own lists: HR people (with department and job function), third-party accounts (vendor and sponsor), a login map, the system accounts you name, and the review record: one row per review and one row per decision. The files and their columns are in step 1.
- Python 3 and the `duckdb` module (we ran it with duckdb 1.5.6 and 2.0.0).

Step 1. Export each platform’s views into one folder

The script reads these seventeen files from `EXPORT_DIR`, plus the review packets in a folder named `packets` inside it. Column names are the platforms' own; the script lower-cases the header, so `NAME` and `name` both work, and a missing column fails an input check that names it. Names are compared whole and without case. Dates are read as `YYYY-MM-DD`, timestamps as `YYYY-MM-DD HH:MM:SS` in UTC (a fractional second, a `T` and a `Z`, `+00:00` or `+0000` suffix are accepted; any other offset is not). A value that cannot be read is counted by an input check, never skipped.

The seventeen files

File	Columns the script reads	Defined in

File	Columns the script reads	Defined in
`snowflake_users.csv`	name, type, disabled, created_on, deleted_on, last_success_login	Snowflake: ACCOUNT_USAGE.USERS (read 2026-11-01)
`snowflake_grants_to_users.csv`	created_on, deleted_on, role, grantee_name	Snowflake: ACCOUNT_USAGE.GRANTS_TO_USERS (read 2026-11-01)
`snowflake_grants_to_roles.csv`	created_on, deleted_on, privilege, granted_on, name, grantee_name	Snowflake: ACCOUNT_USAGE.GRANTS_TO_ROLES (read 2026-11-01)
`databricks_users.csv`	id, username, active (the SCIM id, userName and active)	Databricks: list account users (read 2026-11-01)
`databricks_service_principals.csv`	id, application_id, display_name, active	Databricks: list account service principals (read 2026-11-01)
`databricks_group_members.csv`	group_id, group_name, member_id (one row per member of each group: its id and displayName, and the member's value)	Databricks: get account group (read 2026-11-01)
`databricks_audit.csv`	event_time, event_date, workspace_id, user_identity, service_name, action_name, request_params, response (maps and structs written as JSON text)	Databricks: audit log system table (read 2026-11-01)
`redshift_svv_user_info.csv`	user_name, user_id, superuser	Amazon Redshift: SVV_USER_INFO (read 2026-11-01)
`redshift_svv_user_grants.csv`	user_name, role_name	Amazon Redshift: SVV_USER_GRANTS (read 2026-11-01)
`redshift_svv_role_grants.csv`	role_name, granted_role_name (role_name includes granted_role_name)	Amazon Redshift: SVV_ROLE_GRANTS (read 2026-11-01)

File	Columns the script reads	Defined in
`redshift_sys_userlog.csv`	user_id, user_name, original_user_name, action, record_time	Amazon Redshift: SYS_USERLOG (read 2026-11-01)
`hr_people.csv`	employee_email, department, job_function	You keep it: your HR file
`login_map.csv`	platform, login, employee_email	You keep it: which login belongs to which person
`third_party_accounts.csv`	platform, login, vendor, sponsor_email	You keep it: vendor and contractor accounts and who sponsors each
`system_accounts.csv`	platform, login	You keep it: the application and system accounts you name (7.2.5 and 7.2.5.1)
`reviews.csv`	review_id, platform, packet_sha256, completed_date, acknowledged_by, acknowledged_date	You keep it: one row per review per platform
`review_decisions.csv`	review_id, platform, login, access, decision (keep or remove), reviewer_email, decision_date	You keep it: one row per decision

Databricks keeps workspace-level events in each workspace's region; the script reads only account-level rows (`workspace_id` 0) for dates, because only those say an account or a group membership changed. A workspace-level row is a change to that workspace and does not count as removal.

Run `python3 pci_access_review_evidence.py --export-sql` to print the queries below. They are what we assume each export looks like; run each one as a user who sees every row, and check one row of each against the linked page.

```
-- Snowflake (run as a role that can read SNOWFLAKE.ACCOUNT_USAGE; set the session to UTC first)
ALTER SESSION SET TIMEZONE = 'UTC';
SELECT name, type, disabled, created_on, deleted_on, last_success_login FROM
SNOWFLAKE.ACCOUNT_USAGE.USERS;
SELECT created_on, deleted_on, role, grantee_name FROM SNOWFLAKE.ACCOUNT_USAGE.GRANTS_TO_USERS;
-- every grant on a role; the script counts only USAGE as inheritance (an OWNERSHIP row is not)
SELECT created_on, deleted_on, privilege, granted_on, name, grantee_name FROM
SNOWFLAKE.ACCOUNT_USAGE.GRANTS_TO_ROLES
WHERE granted_on = 'ROLE';

-- Databricks: system.access.audit (maps and structs written as JSON text). Databricks keeps 365
```

```
days: take all of
-- them. Workspace-level rows are kept in the workspace's region: run it in each region you use.
SELECT event_time, event_date, workspace_id, to_json(user_identity) AS user_identity, service_name,
action_name,
       to_json(request_params) AS request_params, to_json(response) AS response
FROM system.access.audit WHERE service_name = 'accounts' AND event_date >= date_sub(current_date(),
365);
-- databricks_users.csv (id, username, active): GET /api/2.0/accounts/{account_id}/scim/v2/Users
--   -> id, userName, active, one row per user
-- databricks_service_principals.csv (id, application_id, display_name, active):
--   GET .../scim/v2/ServicePrincipals -> id, applicationId, displayName, active
-- databricks_group_members.csv (group_id, group_name, member_id): GET .../scim/v2/Groups, then
--   GET .../scim/v2/Groups/{id} for each group (the list no longer returns members) -> id,
displayName and each
--   member's value, one row per member

-- Redshift (as a superuser, or a user who can see every row of these views)
SELECT user_name, user_id, superuser FROM svv_user_info;
SELECT user_name, role_name FROM svv_user_grants;
SELECT role_name, granted_role_name FROM svv_role_grants;
SELECT user_id, user_name, original_user_name, action, record_time FROM sys_userlog;
```

A review needs its packet. Each run writes its lists and each platform's packet to its own folder, `evidence-out/run-<UTC time>/`, with the packet at `packets/packet_<platform>_<sha256>.csv`, prints the SHA-256, and deletes nothing from earlier runs. Give the reviewer that packet, record the SHA-256 on the review's row in `reviews.csv`, and keep the packet file: put it in the `packets` folder next to your exports. A packet file whose rows no longer hash to its name fails an input check. Only each platform's latest review needs its packet; an older review, one from before you used the script included, is read for its dates and decisions.

Step 2. Set your settings

PCI sets the six-month rhythm and leaves the deadline for addressing wrong access to you, so the script has a default for the first and none for the second. Here is the settings block at the top of the download. The demo fills in 14 for REMEDIATION_DAYS only so there is something to measure; that is not a recommendation, and nothing on this page says what your number should be. REVIEW_INTERVAL_DAYS is 184, the longest gap Table 4 allows; lower it if your policy is stricter.

```
# ---- CHANGE THESE FOR YOUR OWN DATA -------------------------------------------------------
AS_OF = "2026-10-30"                # 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
REVIEW_INTERVAL_DAYS = 184          # longest gap between reviews you will accept (PCI DSS v4.0.1
section 7, Table 4)
REMEDIATION_DAYS = None             # no default: days after a remove decision by which the access
must be gone
# -------------------------------------------------------------------------------------------
```

Step 3. How each list is decided

- **Accounts.** Snowflake: every user in `USERS` with no `DELETED_ON`. Databricks: every SCIM user, and every service principal as a system account. Redshift: every user in `SVV_USER_INFO`. An account is a system account if its Snowflake type is one of the service types, if it is a Databricks service principal, or if you name it in `system_accounts.csv`. Every other account is in the packet. List 1b labels the system accounts 7.2.5 and 7.2.5.1; the standard's notes also point to 8.6.1 to 8.6.3 for them.
- **Access.** Roles and groups the account holds now: Snowflake `GRANTS_TO_USERS` with no `DELETED_ON`; Databricks group members, joined on the member's value and the user's `id`; Redshift `SVV_USER_GRANTS`. Inherited access follows role-in-role grants (Snowflake `GRANTS_TO_ROLES` rows with `GRANTED_ON` ROLE and `PRIVILEGE` USAGE, since an OWNERSHIP row is not inheritance; Redshift `SVV_ROLE_GRANTS`) and Databricks nested groups, as far as they go; each inherited row says `via` the role the account holds. A Snowflake grant created before the live user with that name was created belonged to an earlier holder of the name: it is left out and counted. A Redshift superuser gets a row, `(superuser)`, because that access is a flag and not a role. An account with nothing the script reads gets a row, `(no role or group read)`, so it cannot drop out of the review. Not read, so not in the packet: object privileges granted straight to a user, Snowflake database roles, Redshift user groups, the Databricks account admin role and workspace-local groups (each workspace's admins group included), and a Redshift superuser's reach beyond the flag. An account shown as `(no role or group read)` may still hold access through one of them.
- **Owner.** The login map gives each login an employee, who must be in the HR list; otherwise the login is looked up in the third-party list. An account that matches neither is NO OWNER. Last sign-in is shown for Snowflake only; the other two exports here carry none.
- **The packet's SHA-256.** Each platform's packet is its platform, login, access and held_how, sorted and hashed. It is what a review records, and the review's packet file must hash back to it.
- **Timing.** Completed reviews per platform, in date order. A review is a GAP OVER 184 DAYS if more than `REVIEW_INTERVAL_DAYS` days passed since the one before it, and NO ACKNOWLEDGMENT if nobody is named, or no date given, for management's confirmation. After the last review the clock keeps running to the export day: more than `REVIEW_INTERVAL_DAYS` days is OVERDUE.
- **Coverage.** Only the latest review per platform, against the packet file it names. Each packet row needs one decision (keep or remove). No decision is NOT REVIEWED; a decision with no reviewer is NO REVIEWER; a reviewer whose email is the one the login map gives for that login is REVIEWED OWN ACCESS. The same person may review many logins; the script only

flags the one reviewing themselves. Access added since that packet is counted, not flagged: it goes into the next review.

- **Decisions carried out.** Every remove decision in any review. Held now means the person still holds that access in today's packet: through the login itself; through its new name after a Redshift rename in `SYS_USERLOG` made since the review's packet could have been taken, that is since the platform's previous review, or `REVIEW_INTERVAL_DAYS` before this one if there was none (along every rename in a row, and a drop under the new name dates the removal), unless the new name is already its own row in the packet the review was given, or the review decided on that login, since then it is a different user; or through another user login the login map gives the same person, not a system account, unless a review on or after that one decided to keep that login's access. The `held_now_by` column names the logins that hold it, and `account_enabled` is true if any of them is enabled. When nothing holds it, `account_enabled` is the decided login's state, renames followed. If it does and a later review decided to keep it, the row says KEPT BY A LATER REVIEW. Otherwise it is STILL HELD PAST DEADLINE once the decision date plus `REMEDIATION_DAYS` has passed, and STILL HELD, NOT YET DUE before that. If it is gone, the date is the first removal the platform recorded on or after the decision: Snowflake grant `DELETED_ON` or user `DELETED_ON`; Databricks account-level `removePrincipalFromGroup`, `removePrincipalsFromGroup` or `delete` with status 200 (a refused attempt is not a removal); Redshift drop of the user in `SYS_USERLOG`. A removal after the deadline is REMOVED LATE. Gone with no recorded date, such as a Redshift role revoke, is GONE, DATE NOT RECORDED, or GONE, BEFORE THE RECORDS START when the decision is older than the earliest Databricks audit row or Redshift user log row in your export. Databricks keeps 365 days, so a Databricks decision inside that window needs the records to reach it, and an input check says so if they do not. A deactivated Databricks account that still holds the group is STILL HELD, with `account_enabled` false: whether deactivation was your way of carrying out the decision is for you and the assessor.

Step 4. Run it on the clean generated records

`python3 pci_access_review_evidence.py --demo` writes generated exports in each platform's column names, with a review record, and reads them back. In this run every account has an owner, both reviews are within 184 days, every packet row has a decision by someone other than the account's owner, and all five remove decisions are carried out within the 14 days the demo sets. So nothing needs follow-up. That says only that these generated records agree with these settings. The generated Redshift log reaches back further than the seven days to plan for; on a real account, a drop older than your log keeps shows as GONE, DATE NOT RECORDED. (Lists 1 and 1b are left out of the excerpts below; the script prints them first.)

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

`python3 pci_access_review_evidence.py --demo --plant` writes the same records with problems planted. All 18 input checks pass; the lists find what was planted:

- List 1: one Snowflake account with no HR or third-party match (NO OWNER).
- List 2: the Snowflake reviews are 192 days apart (GAP OVER 184 DAYS); Redshift's only review is unacknowledged (NO ACKNOWLEDGMENT) and is 199 days old at the export (OVERDUE).
- List 3: in Snowflake's packet of 8 rows, one inherited role was never decided on (NOT REVIEWED), one decision was made by the account's own holder (REVIEWED OWN ACCESS), and one has no reviewer (NO REVIEWER).
- List 4: of five remove decisions, four are STILL HELD PAST DEADLINE (one on Snowflake, one on Databricks where the account is deactivated, and two on Redshift, where jon was renamed jon_x and kept the access) and one is REMOVED LATE: Databricks recorded the removal on 2026-10-20, after a refused attempt on 2026-09-29 that does not count.

Step 6. Run it with faults planted in the inputs

`python3 pci_access_review_evidence.py --demo --plant-input` writes the clean records with faults planted in the inputs, and leaves REMEDIATION_DAYS unset. 17 of the 18 input checks fail, each naming what it found; lists 1 to 3 are marked INCOMPLETE and list 4 does not run. Because the edited Snowflake packet cannot be trusted, list 3 says Snowflake is NOT CHECKED rather than reading it.

What each input check caught in this run

Input check	Planted fault that trips it
dates, interval and platforms in the settings	Not tripped in run 3. It fails if AS_OF is written 30/10/2026, REVIEW_INTERVAL_DAYS is 0, or PLATFORMS_EXPORTED names a platform that is not one of the three.
REMEDIATION_DAYS is set	REMEDIATION_DAYS left unset: list 4 prints NOT RUN.
every file is there with every column	The Redshift role-grants export came with granted_role_name renamed.
every date and time is readable	A review dated 4/14/26 and another dated 2026-02-30.
every value is one the script knows	DISABLED set to maybe, a Snowflake TYPE of ROBOT, a decision of maybe, a SYS_USERLOG action of suspend.

Input check	Planted fault that trips it
every Databricks removal row is readable	A delete row whose response is not JSON.
no login appears twice in an account list	The Snowflake user ANA listed twice.
no login is mapped to two people	ANA mapped to a second email.
no person twice in the HR list, no login twice in the third-party list	One employee listed twice.
every system_accounts login is an account	redshift/svc_billing, which is not an account.
every third_party_accounts login is an account	A Snowflake login that does not exist.
every login that holds access is in the account list	A Snowflake grant to a user, ZED, who is not in the user list.
each review once per platform, none after the export	A review dated 2027-01-15.
each platform's latest review has its packet, and it hashes to the recorded SHA-256	One row of Snowflake's packet file edited after the review.
no access has two decisions in one review	ANA's FIN_ANALYST decided as both keep and remove.
every decision belongs to a review	A decision for review R-9999.
every decision is about a row of its packet	A Databricks decision on an account that is not in the packet.
Databricks records reach back to every remove decision of the last 365 days	The oldest audit row removed, so the records start after a decision.

The code

The complete download, `pci_access_review_evidence.py`, as it ran. The online copy is at www.datameaning.com/guides/pci-dss-access-review-evidence-for-snowflake-databricks-redshift/pci_access_review_evidence.py.

```
#!/usr/bin/env python3
"""PCI DSS 7.2.4 six-month access review evidence from Snowflake, Databricks and Redshift exports,
in DuckDB.
```

```
    python3 pci_access_review_evidence.py --demo                # clean generated records
    python3 pci_access_review_evidence.py --demo --plant        # generated records with planted
findings
    python3 pci_access_review_evidence.py --demo --plant-input  # generated records with planted
input faults
    python3 pci_access_review_evidence.py --export-sql          # print the export queries for each
platform
    python3 pci_access_review_evidence.py                       # your exports: set the settings
below first
    python3 pci_access_review_evidence.py --strict              # exit 1 unless nothing needs
follow-up

Requirement 7.2.4 (PCI DSS v4.0.1) is about the review itself: user accounts and the access tied to
them are
reviewed at least once every six months, access that is wrong is addressed, and management confirms
the access is
still appropriate. The script reads CSV exports of each platform's account, grant and audit
records, plus lists you keep (HR
people, third-party accounts, a login map, the system accounts you name, and the review record: one
row per review
and one row per decision). It prints, each with its filters, run time, row count and a SHA-256 of
its rows:
  1.  the review packet: every user account on each platform with each role or group it holds,
direct and inherited,
      its owner, department and job function. NO OWNER flags an account that matches nobody. The
packet's SHA-256 is
      what a review records. 1b lists system accounts separately: they fall under 7.2.5 and
7.2.5.1, not 7.2.4.
  2.  review timing: each completed review per platform, the days since the one before it, and who
acknowledged it
      for management. GAP OVER n DAYS, OVERDUE, NO REVIEW RECORDED, NO ACKNOWLEDGMENT.
  3.  review coverage: the latest review per platform against the packet it was given (matched by
SHA-256, so an
      edited packet fails an input check). NOT REVIEWED, REVIEWED OWN ACCESS, NO REVIEWER.
  4.  decisions carried out: every remove decision, whether the access is still held now, and when
it went where the
      platform records that, through any login the login map gives the same person. STILL HELD PAST
DEADLINE and
      REMOVED LATE are the findings.
Each run writes its lists and packets to its own folder, evidence-out/run-<UTC time>/, and deletes
nothing.
Input checks run first; if one fails, the lists are marked INCOMPLETE. A platform you do not export
is printed as
NOT CHECKED. The script reads files; it never connects to a platform and never writes to one. It
produces evidence;
the assessor decides whether a review was late, complete or enough.

PCI sets no deadline for addressing wrong access, so REMEDIATION_DAYS has no default: unset, list 4
does not run. The demo uses 14
so the demo has something to measure; it is not a recommendation. REVIEW_INTERVAL_DAYS is 184, the
longest gap the
standard's section 7, Table 4 allows for "every six months".

The demo records are GENERATED (fixed rows, no randomness): not a company's data. Column names come
from:
  Snowflake  https://docs.snowflake.com/en/sql-reference/account-usage/users
             https://docs.snowflake.com/en/sql-reference/account-usage/grants_to_users
             https://docs.snowflake.com/en/sql-reference/account-usage/grants_to_roles
```

```
  Databricks https://docs.databricks.com/aws/en/admin/account-settings/audit-logs
             https://docs.databricks.com/api/account/accountusers/list
             https://docs.databricks.com/api/account/accountgroups/get
             https://docs.databricks.com/api/account/accountserviceprincipals/list
  Redshift   https://docs.aws.amazon.com/redshift/latest/dg/r_SVV_USER_INFO.html
             https://docs.aws.amazon.com/redshift/latest/dg/r_SVV_USER_GRANTS.html
             https://docs.aws.amazon.com/redshift/latest/dg/r_SVV_ROLE_GRANTS.html
             https://docs.aws.amazon.com/redshift/latest/dg/SYS_USERLOG.html
"""
import csv
import datetime as dt
import hashlib
import json
import pathlib
import re
import sys

import duckdb

# ---- CHANGE THESE FOR YOUR OWN DATA -----------------------------------------------------------
AS_OF = "2026-10-30"                # 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
REVIEW_INTERVAL_DAYS = 184          # longest gap between reviews you will accept (PCI DSS v4.0.1
section 7, Table 4)
REMEDIATION_DAYS = None             # no default: days after a remove decision by which the access
must be gone
# ------------------------------------------------------------------------------------------------

FILES = {
    "snowflake_users": ["name", "type", "disabled", "created_on", "deleted_on",
"last_success_login"],
    "snowflake_grants_to_users": ["created_on", "deleted_on", "role", "grantee_name"],
    "snowflake_grants_to_roles": ["created_on", "deleted_on", "privilege", "granted_on", "name",
"grantee_name"],
    "databricks_users": ["id", "username", "active"],
    "databricks_service_principals": ["id", "application_id", "display_name", "active"],
    "databricks_group_members": ["group_id", "group_name", "member_id"],
    "databricks_audit": ["event_time", "event_date", "workspace_id", "user_identity",
"service_name", "action_name",
                         "request_params", "response"],
    "redshift_svv_user_info": ["user_name", "user_id", "superuser"],
    "redshift_svv_user_grants": ["user_name", "role_name"],
    "redshift_svv_role_grants": ["role_name", "granted_role_name"],
    "redshift_sys_userlog": ["user_id", "user_name", "original_user_name", "action",
"record_time"],
    "hr_people": ["employee_email", "department", "job_function"],
    "login_map": ["platform", "login", "employee_email"],
    "third_party_accounts": ["platform", "login", "vendor", "sponsor_email"],
    "system_accounts": ["platform", "login"],
    "reviews": ["review_id", "platform", "packet_sha256", "completed_date", "acknowledged_by",
"acknowledged_date"],
    "review_decisions": ["review_id", "platform", "login", "access", "decision", "reviewer_email",
"decision_date"],
}
PLATFORM_FILES = {
    "snowflake": ["snowflake_users", "snowflake_grants_to_users", "snowflake_grants_to_roles"],
```

```
    "databricks": ["databricks_users", "databricks_service_principals", "databricks_group_members",
"databricks_audit"],
    "redshift": ["redshift_svv_user_info", "redshift_svv_user_grants", "redshift_svv_role_grants",
                 "redshift_sys_userlog"],
}
PLATFORMS = ["snowflake", "databricks", "redshift"]
SNOWFLAKE_SYSTEM_TYPES = ["SERVICE", "SERVICE_AGENT", "LEGACY_SERVICE", "SNOWFLAKE_SERVICE"]
SNOWFLAKE_TYPES = ["", "PERSON"] + SNOWFLAKE_SYSTEM_TYPES      # a blank TYPE works like PERSON
DATABRICKS_REMOVAL_ACTIONS = {"removePrincipalFromGroup": "group", "removePrincipalsFromGroup":
"group",
                              "delete": "account"}
OUT_DIR = pathlib.Path("evidence-out")
GEN_DIR = pathlib.Path("generated")
TS_RE = "\\d{4}-\\d{2}-\\d{2}[ T]\\d{2}:\\d{2}:\\d{2}(\\.\\d+)?( ?(Z|\\+00:00|\\+0000))?"
DATE_RE = "\\d{4}-\\d{2}-\\d{2}"
SHA_RE = "[0-9a-f]{64}"
RUN_AT = dt.datetime.now(dt.timezone.utc).strftime("%Y-%m-%d %H:%M:%S UTC")


def ts(col):
    """A timestamp read strictly: YYYY-MM-DD HH:MM:SS taken as UTC (a Z, +00:00 or +0000 suffix is
accepted, any
    other offset is NULL and counted). Export from a session set to UTC."""
    return (f"CASE WHEN regexp_full_match(trim({col}), '{TS_RE}') THEN "
            f"try_strptime(substr(replace(trim({col}), 'T', ' '), 1, 19), '%Y-%m-%d %H:%M:%S')
END")


def day(col):
    return f"CASE WHEN regexp_full_match(trim({col}), '{DATE_RE}') THEN try_strptime(trim({col}),
'%Y-%m-%d')::DATE END"


def jv(col, key):
    """One key of a JSON text column, NULL when the text is not valid JSON."""
    return f"CASE WHEN json_valid({col}) THEN json_extract_string({col}, '$.{key}') END"


def status_ok(col="response"):
    return f"coalesce({jv(col, 'status_code')}, {jv(col, 'statusCode')}) = '200'"


def blank(col):
    return f"nullif(trim({col}), '')"


def quote_list(items):
    return ", ".join("'" + str(i).replace("'", "''") + "'" for i in items)


def fingerprint(rows):
    h = hashlib.sha256()
    for r in sorted("|".join("" if v is None else str(v) for v in row) for row in rows):
        h.update(r.encode() + b"\n")
    return h.hexdigest()


def write_packet(folder, platform, rows):
```

```
    """A platform's packet as the review is given it: platform, login, access, held_how. Named by
its SHA-256."""
    sha = fingerprint(rows)
    folder.mkdir(parents=True, exist_ok=True)
    with open(folder / f"packet_{platform}_{sha}.csv", "w", newline="", encoding="utf-8") as f:
        w = csv.writer(f)
        w.writerow(["platform", "login", "access", "held_how"])
        w.writerows(rows)
    return sha


# ---------------------------------------------------------------------------------------------
# Generated demo records (fixed rows). Everything down to "ENGINE" builds files only.
# ---------------------------------------------------------------------------------------------
def write_csv(folder, name, rows):
    with open(folder / f"{name}.csv", "w", newline="", encoding="utf-8") as f:
        w = csv.writer(f)
        w.writerow(FILES[name])
        w.writerows(rows)


def generate(folder, plant, plant_input):
    folder.mkdir(parents=True, exist_ok=True)
    pk = folder / "packets"
    for stale in pk.glob("packet_*.csv"):     # packets of an earlier demo run would otherwise sit
beside these
        stale.unlink()
    E =lambda n: f"{n}@example.test"
    LEE = E("lee")

    # ---- the packets each platform's last review was given (what the reviewer saw)
    sfp = [("snowflake", "ana", "fin_analyst", "direct"), ("snowflake", "ben", "sales_reader",
"direct"),
           ("snowflake", "chen", "data_engineer", "direct"), ("snowflake", "chen", "analyst_base",
"via data_engineer"),
           ("snowflake", "dara", "data_engineer", "direct"), ("snowflake", "dara", "analyst_base",
"via data_engineer"),
           ("snowflake", "dara", "fin_analyst", "direct"), ("snowflake", "admin_sam",
"accountadmin", "direct")]
    dbp = [("databricks", E("ana"), "finance-readers", "direct"), ("databricks", E("ben"), "sales-
readers", "direct"),
           ("databricks", E("chen"), "data-engineers", "direct"), ("databricks", E("chen"), "all-
analysts", "via data-engineers"),
           ("databricks", E("dara"), "data-engineers", "direct"), ("databricks", E("dara"), "all-
analysts", "via data-engineers"),
           ("databricks", E("dara"), "finance-readers", "direct"), ("databricks", E("ivy"), "sales-
readers", "direct"),
           ("databricks", E("sam"), "(no role or group read)", "none")]
    rsp = [("redshift", "ana", "fin_ro", "direct"), ("redshift", "chen", "data_rw", "direct"),
           ("redshift", "chen", "base_ro", "via data_rw"), ("redshift", "jon", "data_rw",
"direct"),
           ("redshift", "jon", "base_ro", "via data_rw"), ("redshift", "sam", "(superuser)",
"superuser flag")]
    sf_date, db_date = "2026-09-28", "2026-09-28"
    rs_date = "2026-04-14" if plant else "2026-09-28"
    rs_id = "R-2026-04" if plant else "R-2026-09"
    sf_sha, db_sha, rs_sha = (write_packet(pk, "snowflake", sfp), write_packet(pk, "databricks",
dbp),
```

```
                                write_packet(pk, "redshift", rsp))

    # ---- reviews and decisions
    sf_prev = "2026-03-20" if plant else "2026-04-14"
    reviews = [["R-2026-04", "snowflake", "", sf_prev, LEE, "2026-04-16"],
               ["R-2026-09", "snowflake", sf_sha, sf_date, LEE, "2026-09-30"],
               ["R-2026-04", "databricks", "", "4/14/26" if plant_input else "2026-04-14", LEE,
"2026-04-16"],
               ["R-2026-09", "databricks", db_sha, db_date, LEE, "2026-09-30"]]
    if plant:
        reviews.append(["R-2026-04", "redshift", rs_sha, rs_date, "", ""])
    else:
        reviews += [["R-2026-04", "redshift", "", "2026-04-14", LEE, "2026-04-16"],
                    ["R-2026-09", "redshift", rs_sha, rs_date, LEE, "2026-09-30"]]
    removes = {("snowflake", "dara", "fin_analyst"), ("databricks", E("dara"), "finance-readers"),
               ("databricks", E("ivy"), "sales-readers"), ("redshift", "jon", "data_rw"),
("redshift", "jon", "base_ro")}
    decisions = []
    for plat, packet, rid, d in [("snowflake", sfp, "R-2026-09", sf_date), ("databricks", dbp, "R-
2026-09", db_date),
                                 ("redshift", rsp, rs_id, rs_date)]:
        for _, login, access, _ in packet:
            who = LEE
            if plant and (plat, login, access) == ("snowflake", "chen", "analyst_base"):
                continue                                    # a packet row nobody decided on
            if plant and (plat, login, access) == ("snowflake", "admin_sam", "accountadmin"):
                who = E("sam")                              # the account holder reviewed his own
access
            if plant and (plat, login, access) == ("snowflake", "ben", "sales_reader"):
                who = ""                                    # no reviewer recorded
            if plat == "snowflake":
                login, access = login.upper(), access.upper()    # the review record is in another
case
            decisions.append([rid, plat, login, access,
                              "remove" if (plat, login.lower(), access.lower()) in removes else
"keep", who, d])

    # ---- Snowflake
    users = [[n, "PERSON", "false", "2026-01-10 09:00:00", "", "2026-10-28 09:00:00"]
             for n in ["ANA", "BEN", "CHEN", "DARA", "ADMIN_SAM"]]
    users += [["VENDOR_BI", "PERSON", "false", "2026-10-05 09:00:00", "", "2026-10-27 09:00:00"],
              ["SVC_LOAD", "SERVICE", "false", "2026-01-10 09:00:00", "", "2026-10-30 02:00:00"],
              ["HAL", "PERSON", "true", "2025-11-01 09:00:00", "2026-08-02 10:00:00", "2026-07-30
09:00:00"]]
    grants = [["2026-01-15 09:00:00", "", "FIN_ANALYST", "ANA"], ["2026-01-15 09:00:00", "",
"SALES_READER", "BEN"],
              ["2026-01-15 09:00:00", "", "DATA_ENGINEER", "CHEN"], ["2026-09-10 10:00:00", "",
"DATA_ENGINEER", "DARA"],
              ["2026-02-01 09:00:00", "" if plant else "2026-09-30 10:00:00", "FIN_ANALYST",
"DARA"],
              ["2026-01-15 09:00:00", "", "ACCOUNTADMIN", "ADMIN_SAM"], ["2026-10-05 10:00:00", "",
"SALES_READER", "VENDOR_BI"],
              ["2026-01-15 09:00:00", "", "LOADER", "SVC_LOAD"], ["2025-11-03 09:00:00", "2026-08-
02 10:00:00", "FIN_ANALYST", "HAL"]]
    role_grants = [["2026-01-01 09:00:00", "", "USAGE", "ROLE", "ANALYST_BASE", "DATA_ENGINEER"],
                   ["2026-01-01 09:00:00", "", "SELECT", "TABLE", "ORDERS", "FIN_ANALYST"],   # a
table privilege
                   ["2026-01-01 09:00:00", "2026-05-01 09:00:00", "USAGE", "ROLE", "OLD_BASE",
```

```
"FIN_ANALYST"],  # revoked
                    ["2026-01-01 09:00:00", "", "OWNERSHIP", "ROLE", "FIN_ANALYST", "ACCOUNTADMIN"]]
# owning is not holding
    if plant:
        users.append(["GHOST", "PERSON", "false", "2026-10-10 09:00:00", "", "2026-10-12
09:00:00"])
        grants.append(["2026-10-10 10:00:00", "", "SALES_READER", "GHOST"])

    # ---- Databricks
    ids = {n: str(7001 + i) for i, n in enumerate(["ana", "ben", "chen", "dara", "sam", "ivy"])}
    dbu = [[ids[n], E(n), "false" if n == "ivy" else "true"] for n in ids]
    sps = [["8001", "11111111-2222-3333-4444-555555555555", "svc-etl", "true"]]
    members = [["9001", "finance-readers", ids["ana"]], ["9002", "sales-readers", ids["ben"]],
               ["9002", "sales-readers", "8001"], ["9003", "data-engineers", ids["chen"]],
               ["9003", "data-engineers", ids["dara"]], ["9004", "all-analysts", "9003"],
               ["9002", "sales-readers", ids["ivy"]] if plant else None]
    members = [m for m in members if m]
    WS = "1234567890123456"
    audit = []

    def aud(d, ws, action, params, code=200):
        audit.append([f"{d} 10:00:00", d, ws, json.dumps({"email": E("sam")}), "accounts", action,
json.dumps(params),
                      json.dumps({"status_code": code}) if code else "not json"])

    if not plant_input:
        aud("2026-07-01", "0", "updateUser", {"targetUserName": E("sam"), "endpoint":
"accountsConsole"})
    rm = lambda d, who, ws="0", code=200: aud(d, ws, "removePrincipalFromGroup",
        {"targetUserName": E(who), "targetGroupName": "finance-readers" if who == "dara" else
"sales-readers"}, code)
    if not plant:
        for who in ("dara", "ivy"):
            rm("2026-09-29", who)                       # the account-level record
            rm("2026-09-29", who, ws=WS)                # and the workspace-level copy
    else:
        rm("2026-09-29", "dara", code=403)              # refused: not a removal
        rm("2026-10-20", "dara")                        # carried out, late
        rm("2026-09-29", "ivy", ws=WS)                  # the workspace only; the account keeps the
group
    # ---- Redshift
    rsu = [[n, str(i), "true" if n == "sam" else "false"] for i, n in enumerate(["ana", "chen",
"kim", "sam", "svc_bi"], 101)]
    rsg = [["ana", "fin_ro"], ["chen", "data_rw"], ["kim", "data_rw"]]
    rsr = [["data_rw", "base_ro"]]
    ulog = [["101", "ana", "", "create", "2025-12-01 10:00:00"], ["106", "jon", "", "create",
"2025-12-01 10:00:00"],
            ["103", "kim", "", "create", "2026-10-06 10:00:00"]]
    if plant:
        rsu.append(["jon_x", "106", "false"]); rsg.append(["jon_x", "data_rw"])     # jon renamed,
access kept
        ulog.append(["106", "jon_x", "jon", "rename", "2026-10-01 10:00:00"])
    else:
        ulog.append(["106", "jon", "", "drop", "2026-09-30 10:00:00"])

    # ---- lists you keep
    hr = [[E(n), d, j] for n, d, j in [("ana", "Finance", "Analyst"), ("ben", "Sales", "Analyst"),
          ("chen", "Data", "Engineer"), ("dara", "Data", "Engineer"), ("sam", "IT",
```

```
"Administrator"),
          ("kim", "Data", "Engineer"), ("jon", "Data", "Engineer"), ("ivy", "Sales", "Analyst"),
          ("lee", "Finance", "Manager"), ("hal", "Finance", "Analyst")]]
    lm = [["snowflake", n, E(e)] for n, e in [("ANA", "ana"), ("BEN", "ben"), ("CHEN", "chen"),
("DARA", "dara"),
                                              ("ADMIN_SAM", "sam"), ("HAL", "hal")]]
    lm += [["databricks", E(n), E(n)] for n in ["ana", "ben", "chen", "dara", "sam", "ivy"]]
    lm += [["redshift", n, E(n)] for n in ["ana", "chen", "kim", "sam", "jon"]]
    if plant:
        lm.append(["redshift", "jon_x", E("jon")])
    third = [["snowflake", "VENDOR_BI", "ExampleVendor Analytics", E("ana")]]
    system = [["databricks", "11111111-2222-3333-4444-555555555555"], ["redshift", "svc_bi"]]

    if plant_input:
        users.append(["ANA", "PERSON", "false", "2026-01-10 09:00:00", "", "2026-10-28 09:00:00"])
# a login twice
        users[1][2] = "maybe"                                                    # DISABLED
neither true nor false
        users[2][1] = "ROBOT"                                                    # a TYPE the
script does not know
        grants.append(["2026-10-01 10:00:00", "", "SALES_READER", "ZED"])         # holds a
role, not an account
        system.append(["redshift", "svc_billing"])                               # matches no
account
        third.append(["snowflake", "NO_SUCH_VENDOR_LOGIN", "ExampleVendor Analytics", E("ana")])
# matches no account
        lm.append(["snowflake", "ANA", E("someone.else")])                       # one login,
two people
        hr.append([E("ana"), "Finance", "Analyst"])                              # a person
twice
        decisions.append(["R-2026-09", "databricks", E("nobody"), "finance-readers", "keep", LEE,
"2026-09-28"])  # not in packet
        decisions.append(["R-9999", "redshift", "ana", "fin_ro", "keep", LEE, "2026-09-28"])    #
no such review
        members.append(["9002", "sales-readers", "5555"])                          # a member
that is nothing we know
        decisions.append(["R-2026-09", "snowflake", "ANA", "FIN_ANALYST", "remove", LEE, "2026-09-
28"])   # a second, conflicting
        decisions[3][4] = "maybe"                                                # a decision
that is neither
        aud("2026-09-29", "0", "delete", {"targetUserName": E("hal")}, code=0)       # response not
JSON
        ulog.append(["103", "kim", "", "suspend", "2026-10-07 10:00:00"])          # an action the
script does not know
        reviews.append(["R-2026-04", "snowflake", "", "2026-02-30", LEE, "2026-04-16"])   # not a
real date
        reviews.append(["R-2027-01", "snowflake", "", "2027-01-15", LEE, "2027-01-16"])   # after
the export

    write_csv(folder, "snowflake_users", users)
    write_csv(folder, "snowflake_grants_to_users", grants)
    write_csv(folder, "snowflake_grants_to_roles", role_grants)
    write_csv(folder, "databricks_users", dbu)
    write_csv(folder, "databricks_service_principals", sps)
    write_csv(folder, "databricks_group_members", members)
    write_csv(folder, "databricks_audit", audit)
    write_csv(folder, "redshift_svv_user_info", rsu)
    write_csv(folder, "redshift_svv_user_grants", rsg)
```

```
    write_csv(folder, "redshift_svv_role_grants", rsr)
    write_csv(folder, "redshift_sys_userlog", ulog)
    write_csv(folder, "hr_people", hr)
    write_csv(folder, "login_map", lm)
    write_csv(folder, "third_party_accounts", third)
    write_csv(folder, "system_accounts", system)
    write_csv(folder, "reviews", reviews)
    write_csv(folder, "review_decisions", decisions)
    if plant_input:
        p = pk / f"packet_snowflake_{sf_sha}.csv"       # the packet was edited after the review:
name and rows differ
        p.write_text(p.read_text(encoding="utf-8").replace("ben,sales_reader", "ben,sales_writer",
1), encoding="utf-8")
        p = folder / "redshift_svv_role_grants.csv"     # the export came with a column renamed
        p.write_text(p.read_text(encoding="utf-8").replace("role_name,granted_role_name",
"role_name,granted_role", 1),
                     encoding="utf-8")


EXPORT_SQL = """-- Snowflake (run as a role that can read SNOWFLAKE.ACCOUNT_USAGE; set the session
to UTC first)
ALTER SESSION SET TIMEZONE = 'UTC';
SELECT name, type, disabled, created_on, deleted_on, last_success_login FROM
SNOWFLAKE.ACCOUNT_USAGE.USERS;
SELECT created_on, deleted_on, role, grantee_name FROM SNOWFLAKE.ACCOUNT_USAGE.GRANTS_TO_USERS;
-- every grant on a role; the script counts only USAGE as inheritance (an OWNERSHIP row is not)
SELECT created_on, deleted_on, privilege, granted_on, name, grantee_name FROM
SNOWFLAKE.ACCOUNT_USAGE.GRANTS_TO_ROLES
WHERE granted_on = 'ROLE';

-- Databricks: system.access.audit (maps and structs written as JSON text). Databricks keeps 365
days: take all of
-- them. Workspace-level rows are kept in the workspace's region: run it in each region you use.
SELECT event_time, event_date, workspace_id, to_json(user_identity) AS user_identity, service_name,
action_name,
       to_json(request_params) AS request_params, to_json(response) AS response
FROM system.access.audit WHERE service_name = 'accounts' AND event_date >= date_sub(current_date(),
365);
-- databricks_users.csv (id, username, active): GET /api/2.0/accounts/{account_id}/scim/v2/Users
--   -> id, userName, active, one row per user
-- databricks_service_principals.csv (id, application_id, display_name, active):
--   GET .../scim/v2/ServicePrincipals -> id, applicationId, displayName, active
-- databricks_group_members.csv (group_id, group_name, member_id): GET .../scim/v2/Groups, then
--   GET .../scim/v2/Groups/{id} for each group (the list no longer returns members) -> id,
displayName and each
--   member's value, one row per member

-- Redshift (as a superuser, or a user who can see every row of these views)
SELECT user_name, user_id, superuser FROM svv_user_info;
SELECT user_name, role_name FROM svv_user_grants;
SELECT role_name, granted_role_name FROM svv_role_grants;
SELECT user_id, user_name, original_user_name, action, record_time FROM sys_userlog;
"""


# ---------------------------------------------------------------------------------------------
# ENGINE
# ---------------------------------------------------------------------------------------------
```

```
class Report:
    def __init__(self):
        self.checks, self.follow_up = [], {}

    def check(self, name, ok, detail=""):
        self.checks.append((name, bool(ok), detail))
        print(f"  [{'ok' if ok else 'FAIL'}] {name}" + (f": {detail}" if detail else ""))
        return ok


def main():
    args = sys.argv[1:]
    global EXPORT_DIR, REMEDIATION_DAYS, OUT_DIR, GEN_DIR
    if "--out" in args:       # write generated/ and evidence-out/ under this folder instead of the
current one
        base = pathlib.Path(args[args.index("--out") + 1])
        OUT_DIR, GEN_DIR = base / "evidence-out", base / "generated"
    if "--export-sql" in args:
        print(EXPORT_SQL)
        return 0
    if "--demo" in args:
        plant, plant_input = "--plant" in args, "--plant-input" in args
        generate(GEN_DIR, plant, plant_input)
        EXPORT_DIR = str(GEN_DIR)
        REMEDIATION_DAYS = None if plant_input else 14      # an unset setting: list 4 must not run
        print(f"Generated records written to {GEN_DIR}/ (fixed rows; not a company's data)."
              f" Demo settings: REVIEW_INTERVAL_DAYS={REVIEW_INTERVAL_DAYS}, REMEDIATION_DAYS=
{REMEDIATION_DAYS}.")
    if EXPORT_DIR is None:
        print("Set EXPORT_DIR at the top of the script (or run with --demo).")
        return 2
    rep = Report()
    con = duckdb.connect()
    folder = pathlib.Path(EXPORT_DIR)
    # this run's own folder: it holds this run only, and earlier runs' lists and packets are never
touched
    run_dir = OUT_DIR / ("run-" + RUN_AT.replace("-", "").replace(":", "").replace(" UTC",
"Z").replace(" ", "T"))
    n_same = 1
    while run_dir.exists():   # a second run in the same second gets its own folder too
        n_same += 1
        run_dir = run_dir.with_name(run_dir.name.split("+")[0] + f"+{n_same}")
    run_dir.mkdir(parents=True)
    print(f"\nRun at {RUN_AT}. Exports taken {AS_OF} (UTC); platforms exported: {',
'.join(PLATFORMS_EXPORTED) or 'none'}; "
          f"not checked: {', '.join(p for p in PLATFORMS if p not in PLATFORMS_EXPORTED) or
'none'}.")
    print(f"This run's lists and packets go to {run_dir}/")
    print("\nInput checks")

    # --- settings
    problems = []
    if not (isinstance(AS_OF, str) and re.fullmatch(DATE_RE, AS_OF)):
        problems.append("AS_OF is not written as YYYY-MM-DD")
    else:
        try:
            dt.date.fromisoformat(AS_OF)
        except ValueError:
```

```
            problems.append("AS_OF is not a real calendar date")
    if not (isinstance(REVIEW_INTERVAL_DAYS, int) and REVIEW_INTERVAL_DAYS > 0):
        problems.append("REVIEW_INTERVAL_DAYS is not a whole number above 0")
    for p in PLATFORMS_EXPORTED:
        if p not in PLATFORMS:
            problems.append(f"PLATFORMS_EXPORTED has '{p}', not one of {PLATFORMS}")
    rep.check("dates, interval and platforms in the settings", not problems, "; ".join(problems))
    if problems:
        return 2
    rem_set = isinstance(REMEDIATION_DAYS, int) and REMEDIATION_DAYS >= 0
    rep.check("REMEDIATION_DAYS is set (PCI sets no deadline for addressing wrong access: it is
yours)", rem_set,
              "" if rem_set else "not set to a whole number of days: list 4 will not run")

    # --- load every file; missing file or column fails
    bad_files = []
    for name, cols in FILES.items():
        plat = next((p for p, fs in PLATFORM_FILES.items() if name in fs), None)
        path = folder / f"{name}.csv"
        have = []
        unreadable_file = False
        if (plat is None or plat in PLATFORMS_EXPORTED) and path.is_file():
            try:
                have = [d[0] for d in con.execute("SELECT * FROM read_csv(?, header=true,
all_varchar=true) LIMIT 0",
                                                  [str(path)]).description]
            except duckdb.Error:
                unreadable_file = True
        lowered = [h.strip().lower() for h in have]
        missing = [c for c in cols if c not in lowered]
        if plat is not None and plat not in PLATFORMS_EXPORTED:
            have, missing = [], []     # not exported: an empty table, reported as NOT CHECKED
        elif not path.is_file():
            bad_files.append(f"{name}.csv is missing")
        elif unreadable_file:
            bad_files.append(f"{name}.csv cannot be read as CSV")
        elif missing:
            bad_files.append(f"{name}.csv has no column {', '.join(missing)}")
        loaded = False
        if have and not missing:
            sel = ", ".join(f'"{h}" AS "{h.strip().lower()}"' for h in have)
            try:
                con.execute(f"CREATE TABLE t_{name} AS SELECT {sel} FROM read_csv(?, header=true,
all_varchar=true)",
                            [str(path)])
                loaded = True
            except duckdb.Error:
                bad_files.append(f"{name}.csv cannot be read as CSV")
        if not loaded:
            con.execute(f"CREATE TABLE t_{name} ({', '.join(c + ' VARCHAR' for c in cols)})")
    rep.check("every file is there with every column the script reads", not bad_files, ";
".join(bad_files))
    for name in FILES:
        n = con.execute(f"SELECT count(*) FROM t_{name}").fetchone()[0]
        print(f"      read {n} rows into {name}")

    # --- unreadable values are counted, never skipped
    must_ts = []
```

```
    opt_ts = [("snowflake_users", c) for c in ("created_on", "deleted_on", "last_success_login")] +
\
             [(t, c) for t in ("snowflake_grants_to_users", "snowflake_grants_to_roles") for c in
("created_on", "deleted_on")]
    must_day = [("reviews", "completed_date"), ("review_decisions", "decision_date")]
    opt_day = [("reviews", "acknowledged_date")]
    unreadable = []
    for tbl, col in must_ts + opt_ts:
        need = (tbl, col) in must_ts
        n = con.execute(f"SELECT count(*) FROM t_{tbl} WHERE ({ts(col)}) IS NULL AND "
                        f"{'TRUE' if need else blank(col) + ' IS NOT NULL'}").fetchone()[0]
        if n:
            unreadable.append(f"{tbl}.{col}: {n}")
    n = con.execute(f"SELECT count(*) FROM t_databricks_audit WHERE ({ts('event_time')}) IS
NULL").fetchone()[0]
    n2 = con.execute(f"SELECT count(*) FROM t_redshift_sys_userlog WHERE ({ts('record_time')}) IS
NULL").fetchone()[0]
    for tbl, col, c in [("databricks_audit", "event_time", n), ("redshift_sys_userlog",
"record_time", n2)]:
        if c:
            unreadable.append(f"{tbl}.{col}: {c}")
    for tbl, col in must_day + opt_day:
        need = (tbl, col) in must_day
        n = con.execute(f"SELECT count(*) FROM t_{tbl} WHERE ({day(col)}) IS NULL AND "
                        f"{'TRUE' if need else blank(col) + ' IS NOT NULL'}").fetchone()[0]
        if n:
            unreadable.append(f"{tbl}.{col}: {n}")
    rep.check("every date and time is written as YYYY-MM-DD [HH:MM:SS] (UTC)", not unreadable,
              "not readable, so left out: " + ", ".join(unreadable) if unreadable else "")

    # --- values the script knows
    odd = []
    n = con.execute("SELECT count(*) FROM t_snowflake_grants_to_roles WHERE privilege IS NULL OR "
                    "trim(privilege) = ''").fetchone()[0]
    if n:
        odd.append(f"snowflake_grants_to_roles.privilege blank: {n}")
    for tbl, col in [("snowflake_users", "disabled"), ("databricks_users", "active"),
                     ("databricks_service_principals", "active")]:
        n = con.execute(f"SELECT count(*) FROM t_{tbl} WHERE lower(trim({col})) NOT IN
('true','false') "
                        f"OR {col} IS NULL").fetchone()[0]
        if n:
            odd.append(f"{tbl}.{col} not true or false: {n}")
    n = con.execute("SELECT count(*) FROM t_redshift_svv_user_info WHERE lower(trim(superuser)) NOT
IN "
                    "('true','false','t','f','1','0') OR superuser IS NULL").fetchone()[0]
    if n:
        odd.append(f"redshift_svv_user_info.superuser not true or false: {n}")
    n = con.execute(f"SELECT count(*) FROM t_snowflake_users WHERE upper(trim(coalesce(type, ''))
NOT IN "
                    f"({quote_list(SNOWFLAKE_TYPES)})").fetchone()[0]
    if n:
        odd.append(f"snowflake_users.type not a type the script knows: {n}")
    for tbl in ("login_map", "third_party_accounts", "system_accounts", "reviews",
"review_decisions"):
        n = con.execute(f"SELECT count(*) FROM t_{tbl} WHERE lower(trim(platform)) NOT IN
({quote_list(PLATFORMS)}) "
                        f"OR platform IS NULL").fetchone()[0]
```

```
        if n:
            odd.append(f"{tbl}.platform not one of {', '.join(PLATFORMS)}: {n}")
    n = con.execute("SELECT count(*) FROM t_review_decisions WHERE lower(trim(decision)) NOT IN
('keep','remove') "
                    "OR decision IS NULL").fetchone()[0]
    if n:
        odd.append(f"review_decisions.decision not keep or remove: {n}")
    n = con.execute("SELECT count(*) FROM t_redshift_sys_userlog WHERE lower(trim(action)) NOT IN "
                    "('alter','create','drop','rename') OR action IS NULL").fetchone()[0]
    if n:
        odd.append(f"redshift_sys_userlog.action not alter, create, drop or rename: {n}")
    for tbl, col in [("hr_people", "employee_email"), ("login_map", "login"),
("third_party_accounts", "login"),
                     ("system_accounts", "login"), ("reviews", "review_id"), ("review_decisions",
"review_id"),
                     ("review_decisions", "login"), ("review_decisions", "access"),
("databricks_users", "id"),
                     ("databricks_group_members", "member_id"), ("databricks_group_members",
"group_id")]:
        n = con.execute(f"SELECT count(*) FROM t_{tbl} WHERE {blank(col)} IS NULL").fetchone()[0]
        if n:
            odd.append(f"{tbl}.{col} blank: {n}")
    rep.check("every value is one the script knows", not odd, "; ".join(odd))

    # --- Databricks removal records: the rows the dates come from must be readable
    in_scope = quote_list(DATABRICKS_REMOVAL_ACTIONS)
    nstat = con.execute(f"SELECT count(*) FROM t_databricks_audit WHERE
lower(trim(service_name))='accounts' AND "
                        f"trim(action_name) IN ({in_scope}) AND
coalesce({jv('response','status_code')}, "
                        f"{jv('response','statusCode')}) IS NULL").fetchone()[0]
    nws = con.execute(f"SELECT count(*) FROM t_databricks_audit WHERE
lower(trim(service_name))='accounts' AND "
                      f"trim(action_name) IN ({in_scope}) AND {blank('workspace_id')} IS
NULL").fetchone()[0]
    rep.check("every Databricks removal row has a readable response status and workspace_id",
              nstat == 0 and nws == 0,
              "; ".join(x for x in [f"{nstat} row(s) whose response is not JSON with a status code"
if nstat else "",
                                    f"{nws} row(s) with no workspace_id" if nws else ""] if x))

    # --- accounts, holdings, the packet
    con.execute(f"""CREATE TEMP TABLE accounts AS
        SELECT 'snowflake' AS platform, lower(trim(name)) AS login, lower(trim(disabled)) <> 'true'
AS enabled,
               upper(trim(coalesce(type, ''))) IN ({quote_list(SNOWFLAKE_SYSTEM_TYPES)}) AS
is_system,
               FALSE AS superuser, ({ts('last_success_login')})::DATE::VARCHAR AS last_signin
          FROM t_snowflake_users WHERE {blank('deleted_on')} IS NULL
        UNION ALL SELECT 'databricks', lower(trim(username)), lower(trim(active)) = 'true', FALSE,
FALSE,
                         'not exported' FROM t_databricks_users
        UNION ALL SELECT 'databricks', lower(trim(application_id)), lower(trim(active)) = 'true',
TRUE, FALSE,
                         'not exported' FROM t_databricks_service_principals
        UNION ALL SELECT 'redshift', lower(trim(user_name)), TRUE, FALSE,
                         lower(trim(superuser)) IN ('true','t','1'), 'not exported' FROM
t_redshift_svv_user_info""")
```

```
    con.execute("""UPDATE accounts SET is_system = TRUE WHERE EXISTS (SELECT 1 FROM
t_system_accounts s
        WHERE lower(trim(s.platform)) = accounts.platform AND lower(trim(s.login)) =
accounts.login)""")
    con.execute(f"""CREATE TEMP TABLE direct AS
        SELECT 'snowflake' AS platform, lower(trim(grantee_name)) AS login, lower(trim(role)) AS
role
          FROM t_snowflake_grants_to_users g WHERE {blank('deleted_on')} IS NULL AND NOT EXISTS (
            SELECT 1 FROM t_snowflake_users u WHERE lower(trim(u.name)) =
lower(trim(g.grantee_name))
              AND {blank('u.deleted_on')} IS NULL AND ({ts('g.created_on')}) <
({ts('u.created_on')}))
        UNION SELECT 'databricks', lower(trim(u.username)), lower(trim(g.group_name)) FROM
t_databricks_group_members g
          JOIN t_databricks_users u ON trim(u.id) = trim(g.member_id)
        UNION SELECT 'databricks', lower(trim(s.application_id)), lower(trim(g.group_name)) FROM
t_databricks_group_members g
          JOIN t_databricks_service_principals s ON trim(s.id) = trim(g.member_id)
        UNION SELECT 'redshift', lower(trim(user_name)), lower(trim(role_name)) FROM
t_redshift_svv_user_grants""")
    # parent holds child's access too: Snowflake a USAGE grant on a role to a role (GRANTED_ON =
ROLE: NAME is the
    # object, the role granted; GRANTEE_NAME the recipient role; an OWNERSHIP row is not
inheritance); Databricks a
    # group that is a member of another group; Redshift SVV_ROLE_GRANTS (role_name includes
granted_role_name)
    con.execute(f"""CREATE TEMP TABLE edges AS
        SELECT 'snowflake' AS platform, lower(trim(grantee_name)) AS parent, lower(trim(name)) AS
child
          FROM t_snowflake_grants_to_roles WHERE upper(trim(granted_on)) = 'ROLE' AND
upper(trim(privilege)) = 'USAGE'
            AND {blank('deleted_on')} IS NULL
        UNION SELECT 'databricks', lower(trim(c.group_name)), lower(trim(p.group_name)) FROM
t_databricks_group_members p
          JOIN (SELECT DISTINCT trim(group_id) AS group_id, group_name FROM
t_databricks_group_members) c
            ON c.group_id = trim(p.member_id)
        UNION SELECT 'redshift', lower(trim(role_name)), lower(trim(granted_role_name)) FROM
t_redshift_svv_role_grants""")
    con.execute("""CREATE TEMP TABLE reach AS WITH RECURSIVE r(platform, root, role) AS (
          SELECT DISTINCT platform, role, role FROM direct
          UNION SELECT r.platform, r.root, e.child FROM r JOIN edges e ON e.platform = r.platform
AND e.parent = r.role)
        SELECT * FROM r""")
    con.execute("""CREATE TEMP TABLE access_rows AS
        SELECT a.platform, a.login, r.role AS access, CASE WHEN r.role = r.root THEN 'direct' ELSE
'via ' || r.root END AS how
          FROM accounts a JOIN direct d ON d.platform = a.platform AND d.login = a.login
          JOIN reach r ON r.platform = d.platform AND r.root = d.role
        UNION ALL SELECT platform, login, '(superuser)', 'superuser flag' FROM accounts WHERE
superuser
        UNION ALL SELECT a.platform, a.login, '(no role or group read)', 'none' FROM accounts a
WHERE NOT a.superuser
          AND NOT EXISTS (SELECT 1 FROM direct d WHERE d.platform = a.platform AND d.login =
a.login)""")
    con.execute("""CREATE TEMP TABLE packet_all AS
        SELECT platform, login, access, string_agg(DISTINCT how, '; ' ORDER BY how) AS how
          FROM access_rows GROUP BY 1, 2, 3""")
```

```
    con.execute("""CREATE TEMP TABLE m AS SELECT lower(trim(platform)) AS platform,
lower(trim(login)) AS login,
        min(lower(trim(employee_email))) AS email FROM t_login_map GROUP BY 1, 2""")

    # --- keys and mappings
    dups = con.execute("SELECT platform, login, count(*) FROM accounts GROUP BY 1, 2 HAVING
count(*) > 1 "
                       "ORDER BY 1, 2").fetchall()
    rep.check("no login appears twice in one platform's account list", not dups,
              "; ".join(f"{p}/{l} x{n}" for p, l, n in dups))
    conflicts = con.execute("SELECT lower(trim(platform)), lower(trim(login)) FROM t_login_map
GROUP BY 1, 2 "
                            "HAVING count(DISTINCT lower(trim(employee_email))) > 1 ORDER BY 1,
2").fetchall()
    rep.check("no login is mapped to two people", not conflicts, "; ".join(f"{p}/{l}" for p, l in
conflicts))
    hrdups = con.execute("SELECT lower(trim(employee_email)) FROM t_hr_people GROUP BY 1 HAVING
count(*) > 1 "
                         "ORDER BY 1").fetchall()
    tpdups = con.execute("SELECT lower(trim(platform)) || '/' || lower(trim(login)) FROM
t_third_party_accounts "
                         "GROUP BY 1 HAVING count(*) > 1 ORDER BY 1").fetchall()
    rep.check("no person appears twice in the HR list, and no login twice in the third-party list",
              not hrdups and not tpdups, "; ".join([r[0] for r in hrdups] + [f"third party {r[0]}"
for r in tpdups]))
    for nm, tbl in [("system_accounts", "t_system_accounts"), ("third_party_accounts",
"t_third_party_accounts")]:
        nokey = con.execute(f"""SELECT lower(trim(platform)), lower(trim(login)) FROM {tbl} x
            WHERE lower(trim(platform)) IN ({quote_list(PLATFORMS_EXPORTED)}) AND NOT EXISTS
(SELECT 1 FROM accounts a
              WHERE a.platform = lower(trim(x.platform)) AND a.login = lower(trim(x.login))) ORDER
BY 1, 2""").fetchall()
        rep.check(f"every login in {nm} is an account on its platform", not nokey,
                  "matches no account: " + ", ".join(f"{p}/{l}" for p, l in nokey) if nokey else
"")
    noacct = con.execute("""SELECT DISTINCT d.platform, d.login FROM direct d WHERE NOT EXISTS
(SELECT 1 FROM accounts a
        WHERE a.platform = d.platform AND a.login = d.login) ORDER BY 1, 2""").fetchall()
    rep.check("every login that holds a role or group is in its platform's account list", not
noacct,
              "holds access, not in the account list: " + ", ".join(f"{p}/{l}" for p, l in noacct)
if noacct else "")
    if "databricks" in PLATFORMS_EXPORTED:
        nm = con.execute("""SELECT count(*) FROM t_databricks_group_members g WHERE NOT EXISTS
(SELECT 1 FROM t_databricks_users u
            WHERE trim(u.id) = trim(g.member_id)) AND NOT EXISTS (SELECT 1 FROM
t_databricks_service_principals s
            WHERE trim(s.id) = trim(g.member_id)) AND NOT EXISTS (SELECT 1 FROM
t_databricks_group_members c
            WHERE trim(c.group_id) = trim(g.member_id))""").fetchone()[0]
        print(f"      Databricks group members that are no user, service principal or group in the
export: {nm}")
    if "snowflake" in PLATFORMS_EXPORTED:
        old = con.execute(f"""SELECT count(*) FROM t_snowflake_grants_to_users g JOIN
t_snowflake_users u
            ON lower(trim(u.name)) = lower(trim(g.grantee_name)) AND {blank('u.deleted_on')} IS
NULL
            WHERE {blank('g.deleted_on')} IS NULL AND ({ts('g.created_on')}) <
```

```
({ts('u.created_on')})""").fetchone()[0]
        print(f"      Snowflake grants older than the user who now has that name (an earlier
holder's; left out): {old}")

    # --- reviews: the record the lists read
    con.execute(f"""CREATE TEMP TABLE reviews AS SELECT lower(trim(platform)) AS platform,
trim(review_id) AS review_id,
        lower(trim(packet_sha256)) AS sha, ({day('completed_date')}) AS completed,
{blank('acknowledged_by')} AS ack_by,
        ({day('acknowledged_date')}) AS ack_on FROM t_reviews WHERE ({day('completed_date')}) IS
NOT NULL
        AND lower(trim(platform)) IN ({quote_list(PLATFORMS_EXPORTED)})""")
    con.execute(f"""CREATE TEMP TABLE decisions AS SELECT lower(trim(d.platform)) AS platform,
trim(d.review_id) AS review_id,
        lower(trim(d.login)) AS login, lower(trim(d.access)) AS access, lower(trim(d.decision)) AS
decision,
        lower({blank('d.reviewer_email')}) AS reviewer, ({day('d.decision_date')}) AS decided_on
        FROM t_review_decisions d WHERE lower(trim(d.platform)) IN
({quote_list(PLATFORMS_EXPORTED)})""")
    dupr = con.execute("SELECT platform, review_id FROM reviews GROUP BY 1, 2 HAVING count(*) > 1
ORDER BY 1, 2").fetchall()
    late = con.execute(f"SELECT platform, review_id FROM reviews WHERE completed > DATE '{AS_OF}'
ORDER BY 1, 2").fetchall()
    rep.check("each review appears once per platform and none is dated after the export", not dupr
and not late,
              "; ".join([f"twice: {p}/{r}" for p, r in dupr] + [f"after {AS_OF}: {p}/{r}" for p, r
in late]))
    con.execute("""CREATE TEMP TABLE latest AS SELECT platform, review_id, sha, completed FROM (
        SELECT *, row_number() OVER (PARTITION BY platform ORDER BY completed DESC, review_id DESC)
AS rn
        FROM reviews WHERE completed <= DATE '""" + AS_OF + """') WHERE rn = 1""")
    # packets: only the latest review per platform needs the packet it was given (list 3 is its
only reader); an
    # older review, one from before this script included, is read for its dates and decisions only
    need = con.execute("SELECT platform, review_id, sha FROM latest ORDER BY 1, 2").fetchall()
    con.execute("CREATE TEMP TABLE old_packet (platform VARCHAR, review_id VARCHAR, login VARCHAR,
access VARCHAR)")
    bad_pk = []
    for plat, rid, sha in need:
        path = folder / "packets" / f"packet_{plat}_{sha}.csv"
        if not re.fullmatch(SHA_RE, sha or ""):
            bad_pk.append(f"{plat}/{rid}: no SHA-256 recorded")
        elif not path.is_file():
            bad_pk.append(f"{plat}/{rid}: no packets/{path.name}")
        else:
            try:
                rows = con.execute("SELECT platform, login, access, held_how FROM read_csv(?,
header=true, "
                                   "all_varchar=true)", [str(path)]).fetchall()
            except duckdb.Error:
                bad_pk.append(f"{plat}/{rid}: {path.name} cannot be read")
                continue
            if fingerprint(rows) != sha:
                bad_pk.append(f"{plat}/{rid}: {path.name} was edited (its rows hash to
{fingerprint(rows)[:12]}...)")
            else:
                con.executemany("INSERT INTO old_packet VALUES (?,?,?,?)",
                                [(plat, rid, str(r[1]).lower(), str(r[2]).lower()) for r in rows])
```

```
    rep.check("each platform's latest review has its packet file, and it still hashes to the
recorded SHA-256", not bad_pk,
              "; ".join(bad_pk))
    dd = con.execute("""SELECT platform, review_id, login, access FROM decisions GROUP BY 1, 2, 3,
4
        HAVING count(*) > 1 ORDER BY 1, 2, 3, 4""").fetchall()
    rep.check("no access has two decisions in one review", not dd,
              "; ".join(f"{p}/{r}/{l}/{a}" for p, r, l, a in dd))
    norev = con.execute("""SELECT DISTINCT d.platform, d.review_id FROM decisions d WHERE NOT
EXISTS (SELECT 1 FROM reviews r
        WHERE r.platform = d.platform AND r.review_id = d.review_id) ORDER BY 1, 2""").fetchall()
    rep.check("every decision belongs to a review in the review list", not norev,
              "no such review: " + ", ".join(f"{p}/{r}" for p, r in norev) if norev else "")
    notin = con.execute("""SELECT d.platform, d.review_id, d.login, d.access FROM decisions d WHERE
EXISTS (SELECT 1 FROM old_packet o
        WHERE o.platform = d.platform AND o.review_id = d.review_id) AND NOT EXISTS (SELECT 1 FROM
old_packet o
        WHERE o.platform = d.platform AND o.review_id = d.review_id AND o.login = d.login AND
o.access = d.access)
        ORDER BY 1, 2, 3, 4""").fetchall()
    rep.check("every decision is about a row of the packet its review was given", not notin,
              "not in the packet: " + "; ".join(f"{p}/{r}/{l}/{a}" for p, r, l, a in notin) if
notin else "")
    # Databricks keeps its audit table 365 days: the export must reach back to every remove
decision in that window.
    # An older decision with no recorded date says so on its row (GONE, BEFORE THE RECORDS START).
    rec_start = {}
    if "databricks" in PLATFORMS_EXPORTED:
        first_aud = con.execute(f"SELECT min(({ts('event_time')}))::DATE FROM t_databricks_audit
WHERE "
                                f"lower(trim(service_name)) = 'accounts'").fetchone()[0]
        rec_start["databricks"] = first_aud
        window = dt.date.fromisoformat(AS_OF) - dt.timedelta(days=365)
        first_dec = con.execute(f"SELECT min(decided_on) FROM decisions WHERE platform =
'databricks' AND "
                                f"decision = 'remove' AND decided_on >= DATE
'{window}'").fetchone()[0]
        ok = first_dec is None or (first_aud is not None and first_aud <= first_dec)
        rep.check("Databricks account records reach back to every remove decision of the last 365
days", ok,
                  "" if ok else f"earliest such remove decision {first_dec}; earliest 'accounts'
audit row {first_aud}")
    if "redshift" in PLATFORMS_EXPORTED:
        rec_start["redshift"] = con.execute(f"SELECT min(({ts('record_time')}))::DATE FROM "
                                            f"t_redshift_sys_userlog").fetchone()[0]

    failed = [n for n, ok, _ in rep.checks if not ok]
    flag = "" if not failed else f"  ** INCOMPLETE: {len(failed)} input check(s) failed; read the
lists with that in mind **"
    print(f"\n{len(rep.checks)} input checks, {len(failed)} failed.{flag}")

    def show(title, cols, rows, period, filters, csv_name, only=None):
        print(f"\n== {title}")
        print(f"   period: {period}; filters: {filters}; run at {RUN_AT}; rows: {len(rows)}; "
              f"sha256: {fingerprint(rows)}" + ("; INCOMPLETE" if failed else ""))
        shown = [r for r in rows if only is None or only(r)]
        if only is not None:
            print(f"   showing the {len(shown)} row(s) that need a look; all {len(rows)} are in
```

```
{csv_name}")
        if shown:
            w = [max(len(str(c)), *(len(str(r[i])) for r in shown)) for i, c in enumerate(cols)]
            print("   " + " | ".join(str(c).ljust(w[i]) for i, c in enumerate(cols)))
            for r in shown:
                print("   " + " | ".join(("" if v is None else str(v)).ljust(w[i]) for i, v in
enumerate(r)))
        with open(run_dir / csv_name, "w", newline="", encoding="utf-8") as f:
            cw = csv.writer(f)
            cw.writerow(cols)
            cw.writerows([["" if v is None else v for v in r] for r in rows])

    pl = quote_list(PLATFORMS_EXPORTED)

    # --- 1. the review packet
    rows = con.execute(f"""SELECT p.platform, p.login, p.access, p.how AS held_how,
a.enabled::VARCHAR AS enabled,
          CASE WHEN h.employee_email IS NOT NULL THEN 'employee' WHEN t.login IS NOT NULL THEN
'third party'
               ELSE 'NO OWNER' END AS owner_kind,
          coalesce(lower(trim(h.employee_email)), trim(t.vendor) || ' (sponsor ' ||
lower(trim(t.sponsor_email)) || ')', '') AS owner,
          coalesce(trim(h.department), '') AS department, coalesce(trim(h.job_function), '') AS
job_function,
          a.last_signin, CASE WHEN h.employee_email IS NULL AND t.login IS NULL THEN 'NO OWNER'
ELSE 'ok' END AS status
        FROM packet_all p JOIN accounts a ON a.platform = p.platform AND a.login = p.login
        LEFT JOIN m ON m.platform = p.platform AND m.login = p.login
        LEFT JOIN t_hr_people h ON lower(trim(h.employee_email)) = m.email
        LEFT JOIN t_third_party_accounts t ON lower(trim(t.platform)) = p.platform AND
lower(trim(t.login)) = p.login
        WHERE NOT a.is_system AND p.platform IN ({pl}) ORDER BY 1, 2, 3""").fetchall()
    show("1. The review packet (7.2.4)", ["platform", "login", "access", "held_how", "enabled",
"owner_kind", "owner",
         "department", "job_function", "last_signin", "status"], rows, f"state on {AS_OF}",
         f"user accounts (not system accounts) on {', '.join(PLATFORMS_EXPORTED)}; direct and
inherited access",
         "1_review_packet.csv")
    rep.follow_up["no_owner"] = len({(r[0], r[1]) for r in rows if r[10] == "NO OWNER"})
    print("   what a review records (SHA-256 of each platform's platform, login, access and
held_how):")
    for plat in PLATFORMS_EXPORTED:
        key = [(r[0], r[1], r[2], r[3]) for r in rows if r[0] == plat]
        sha = write_packet(run_dir / "packets", plat, key)
        print(f"      {plat}: {len(key)} rows, sha256 {sha}")
    print(f"   accounts with no HR or third-party match (NO OWNER): {rep.follow_up['no_owner']}")
    srows = con.execute(f"""SELECT a.platform, a.login, a.enabled::VARCHAR AS enabled,
          coalesce((SELECT string_agg(DISTINCT p.access, ', ' ORDER BY p.access) FROM packet_all p
WHERE p.platform = a.platform
                    AND p.login = a.login AND p.access NOT LIKE '(%'), '') AS access, '7.2.5 and
7.2.5.1' AS requirement
        FROM accounts a WHERE a.is_system AND a.platform IN ({pl}) ORDER BY 1, 2""").fetchall()
    show("1b. System accounts (listed so none drops out; not part of 7.2.4)",
         ["platform", "login", "enabled", "access", "requirement"], srows, f"state on {AS_OF}",
         "Snowflake TYPE service types, Databricks service principals, and your system_accounts
list",
         "1b_system_accounts.csv")
```

```
    # --- 2. review timing
    rv = con.execute("SELECT platform, review_id, sha, completed, ack_by, ack_on FROM reviews "
                     "ORDER BY platform, completed, review_id").fetchall()
    asof = dt.date.fromisoformat(AS_OF)
    trows, flagged = [], 0
    for plat in PLATFORMS_EXPORTED:
        mine = [r for r in rv if r[0] == plat and r[3] <= asof]
        prev = None
        for _, rid, sha, comp, ack, ack_on in mine:
            gap = (comp - prev).days if prev else None
            flags = []
            if gap is not None and gap > REVIEW_INTERVAL_DAYS:
                flags.append(f"GAP OVER {REVIEW_INTERVAL_DAYS} DAYS")
            if not ack or not ack_on:
                flags.append("NO ACKNOWLEDGMENT")
            trows.append((plat, rid, str(comp), "" if gap is None else gap, (sha or "")[:12], ack
or "",
                          str(ack_on or ""), " + ".join(flags) or "ok"))
            flagged += bool(flags)
            prev = comp
        if mine:
            since = (asof - mine[-1][3]).days
            over = since > REVIEW_INTERVAL_DAYS
            trows.append((plat, "(since the last review)", str(mine[-1][3]), since, "", "", "",
                          "OVERDUE" if over else "ok"))
            flagged += over
        else:
            trows.append((plat, "(none)", "", "", "", "", "", "NO REVIEW RECORDED"))
            flagged += 1
    show("2. Review timing (7.2.4: at least every six months)", ["platform", "review_id",
"completed", "days_since_previous",
         "packet_sha256", "acknowledged_by", "acknowledged_on", "status"], trows,
         f"reviews completed up to {AS_OF}", f"REVIEW_INTERVAL_DAYS={REVIEW_INTERVAL_DAYS}",
"2_review_timing.csv")
    rep.follow_up["timing"] = flagged

    # --- 3. review coverage
    crow = con.execute("""SELECT o.platform, l.review_id, o.login, o.access, coalesce(d.decision,
'') AS decision,
          coalesce(d.reviewer, '') AS reviewer, coalesce(d.decided_on::VARCHAR, '') AS decided_on,
          CASE WHEN d.login IS NULL THEN 'NOT REVIEWED' WHEN d.reviewer IS NULL THEN 'NO REVIEWER'
               WHEN d.reviewer = (SELECT mm.email FROM m mm WHERE mm.platform = o.platform AND
mm.login = o.login)
               THEN 'REVIEWED OWN ACCESS' ELSE 'ok' END AS status
        FROM latest l JOIN old_packet o ON o.platform = l.platform AND o.review_id = l.review_id
        LEFT JOIN decisions d ON d.platform = o.platform AND d.review_id = o.review_id AND d.login
= o.login
          AND d.access = o.access ORDER BY 1, 3, 4""").fetchall()
    show("3. Review coverage (the latest review against the packet it was given)",
         ["platform", "review_id", "login", "access", "decision", "reviewer", "decided_on",
"status"], crow,
         f"the latest review per platform up to {AS_OF}", "packet matched by its SHA-256; one
decision per packet row",
         "3_review_coverage.csv", only=lambda r: r[7] != "ok")
    rep.follow_up["coverage"] = sum(r[7] != "ok" for r in crow)
    for plat in PLATFORMS_EXPORTED:
        n = [r for r in crow if r[0] == plat]
        if not n:
```

```
            print(f"   {plat}: NOT CHECKED: no latest review with a packet that could be read")
        else:
            print(f"   {plat}: {len(n)} packet rows, {sum(r[7] == 'NOT REVIEWED' for r in n)} not
reviewed, "
                  f"{sum(r[7] == 'REVIEWED OWN ACCESS' for r in n)} reviewed by the account holder,
"
                  f"{sum(r[7] == 'NO REVIEWER' for r in n)} with no reviewer")
    since_rows = con.execute("""SELECT p.platform, count(*) FROM packet_all p JOIN accounts a ON
a.platform = p.platform
        AND a.login = p.login JOIN latest l ON l.platform = p.platform WHERE NOT a.is_system AND
NOT EXISTS (
        SELECT 1 FROM old_packet o WHERE o.platform = p.platform AND o.review_id = l.review_id AND
o.login = p.login
        AND o.access = p.access) AND EXISTS (SELECT 1 FROM old_packet o2 WHERE o2.platform =
p.platform
        AND o2.review_id = l.review_id) GROUP BY 1 ORDER BY 1""").fetchall()
    print("   access rows in today's packet that were not in the latest review's packet (they go
into the next review; "
          "a count to look at, not a finding): " + ("; ".join(f"{p} {n}" for p, n in since_rows) or
"none"))

    # --- 4. decisions carried out
    if rem_set:
        # a login carries on under its new name after a Redshift rename (every rename since the
review's packet could have
        # been taken, that is since the platform's previous review or one interval before this one,
along the whole chain);
        # the same person's other user logins (login map; not system accounts) count unless a
review decided to keep them
        con.execute(f"""CREATE TEMP TABLE renames AS SELECT 'redshift' AS platform,
lower(trim(original_user_name)) AS old,
            lower(trim(user_name)) AS new, ({ts('record_time')})::DATE AS on_date FROM
t_redshift_sys_userlog
            WHERE lower(trim(action)) = 'rename' AND {blank('original_user_name')} IS NOT NULL""")
        con.execute(f"""CREATE TEMP TABLE chain AS WITH RECURSIVE
            s AS (SELECT DISTINCT d.platform, d.review_id, d.login, coalesce((SELECT
max(p.completed) FROM reviews p
                    WHERE p.platform = rv.platform AND p.completed < rv.completed),
                    rv.completed - INTERVAL {REVIEW_INTERVAL_DAYS} DAY)::DATE AS since
                  FROM decisions d JOIN reviews rv ON rv.platform = d.platform AND rv.review_id =
d.review_id),
            c(platform, review_id, root, since, login) AS (
              SELECT platform, review_id, login, since, login FROM s
              UNION SELECT c.platform, c.review_id, c.root, c.since, n.new FROM c JOIN renames n ON
n.platform = c.platform
                AND n.old = c.login AND n.on_date >= c.since
                AND NOT EXISTS (SELECT 1 FROM old_packet o WHERE o.platform = c.platform AND
o.review_id = c.review_id
                                AND o.login = n.new)    -- already its own row in the packet: a
different user
                AND NOT EXISTS (SELECT 1 FROM decisions k WHERE k.platform = c.platform AND
k.review_id = c.review_id
                                AND k.login = n.new))   -- the review decided on it as a user of
its own
            SELECT * FROM c""")
        con.execute("""CREATE TEMP TABLE others AS SELECT a.platform, a.login, b.login AS other
FROM m a JOIN m b
            ON a.platform = b.platform AND a.email = b.email AND a.login <> b.login""")
```

```
        con.execute(f"""CREATE TEMP TABLE removals AS
            SELECT 'snowflake' AS platform, lower(trim(grantee_name)) AS login, lower(trim(role))
AS access,
                   ({ts('deleted_on')})::DATE AS on_date FROM t_snowflake_grants_to_users WHERE
{blank('deleted_on')} IS NOT NULL
            UNION ALL SELECT 'snowflake', lower(trim(name)), '*', ({ts('deleted_on')})::DATE FROM
t_snowflake_users
              WHERE {blank('deleted_on')} IS NOT NULL
            UNION ALL SELECT 'databricks', lower({jv('request_params', 'targetUserName')}),
                   CASE WHEN trim(action_name) = 'delete' THEN '*' ELSE lower({jv('request_params',
'targetGroupName')}) END,
                   ({ts('event_time')})::DATE FROM t_databricks_audit
              WHERE lower(trim(service_name)) = 'accounts' AND trim(action_name) IN ({in_scope})
AND trim(workspace_id) = '0'
                AND {status_ok()}
            UNION ALL SELECT 'redshift', lower(trim(user_name)), '*', ({ts('record_time')})::DATE
FROM t_redshift_sys_userlog
              WHERE lower(trim(action)) = 'drop'""")
        rs_case = " ".join(f"WHEN '{p}' THEN DATE '{d}'" for p, d in rec_start.items() if d) or
"WHEN '' THEN NULL"
        rows = con.execute(f"""WITH r AS (SELECT d.*, rv.completed FROM decisions d JOIN reviews rv
ON rv.platform = d.platform
              AND rv.review_id = d.review_id WHERE d.decision = 'remove' AND d.decided_on IS NOT
NULL),
            mine AS (SELECT r.platform, r.review_id, r.login AS root, r.access, c.login FROM r JOIN
chain c
                     ON c.platform = r.platform AND c.root = r.login AND c.review_id = r.review_id),
            x AS (SELECT r.*, (SELECT string_agg(DISTINCT p.login, ', ' ORDER BY p.login) FROM
packet_all p
                    WHERE p.platform = r.platform AND p.access = r.access AND (p.login = r.login
                      -- its new name after a rename, unless this review kept that login's access
as a user of its own
                      OR (p.login IN (SELECT mi.login FROM mine mi WHERE mi.platform = r.platform
AND mi.review_id = r.review_id
                                      AND mi.root = r.login AND mi.access = r.access)
                          AND NOT EXISTS (SELECT 1 FROM decisions k WHERE k.platform = r.platform
AND k.review_id = r.review_id
                            AND k.login = p.login AND k.access = r.access AND k.decision = 'keep'))
                      -- another user login of the same person, unless a review on or after this
one kept it
                      OR (p.login IN (SELECT o.other FROM others o JOIN accounts oa ON oa.platform
= o.platform
                                      AND oa.login = o.other AND NOT oa.is_system
                                      WHERE o.platform = r.platform AND o.login = r.login)
                          AND NOT EXISTS (SELECT 1 FROM decisions k JOIN reviews rk ON rk.platform
= k.platform
                            AND rk.review_id = k.review_id WHERE k.platform = r.platform AND
k.login = p.login
                            AND k.access = r.access AND k.decision = 'keep' AND rk.completed >=
r.completed)))) AS held_by,
                   (SELECT min(v.on_date) FROM removals v WHERE v.platform = r.platform
                     AND v.login IN (SELECT mi.login FROM mine mi WHERE mi.platform = r.platform AND
mi.review_id = r.review_id
                                     AND mi.root = r.login AND mi.access = r.access)
                     AND (v.access = r.access OR v.access = '*') AND v.on_date >= r.decided_on) AS
gone_on,
                   (SELECT bool_or(a.enabled) FROM accounts a WHERE a.platform = r.platform AND
a.login IN (
```

```
                    SELECT mi.login FROM mine mi WHERE mi.platform = r.platform AND mi.review_id =
r.review_id
                    AND mi.root = r.login AND mi.access = r.access)) AS enabled_now,
                  EXISTS (SELECT 1 FROM decisions k JOIN reviews rk ON rk.platform = k.platform AND
rk.review_id = k.review_id
                    WHERE k.platform = r.platform AND k.access = r.access AND k.decision = 'keep'
AND rk.completed > r.completed
                    AND k.login IN (SELECT mi.login FROM mine mi WHERE mi.platform = r.platform AND
mi.review_id = r.review_id
                                    AND mi.root = r.login AND mi.access = r.access)) AS kept_later
FROM r)
          SELECT x.platform, x.login, x.access, x.review_id, x.decided_on::VARCHAR AS decided_on,
coalesce(x.reviewer, '') AS reviewer,
            coalesce(x.held_by, '') AS held_now_by,
            coalesce(CASE WHEN x.held_by IS NOT NULL THEN (SELECT bool_or(a.enabled) FROM accounts
a  -- the holder's state
                       WHERE a.platform = x.platform AND list_contains(string_split(x.held_by, ',
'), a.login))
                     ELSE x.enabled_now END::VARCHAR, 'no account') AS account_enabled,
            coalesce(x.gone_on::VARCHAR, '') AS gone_on,
            CASE WHEN x.held_by IS NOT NULL AND x.kept_later THEN 'KEPT BY A LATER REVIEW'
                 WHEN x.held_by IS NOT NULL AND x.decided_on + {REMEDIATION_DAYS} < DATE '{AS_OF}'
THEN 'STILL HELD PAST DEADLINE'
                 WHEN x.held_by IS NOT NULL THEN 'STILL HELD, NOT YET DUE'
                 WHEN x.gone_on IS NULL AND x.decided_on < (CASE x.platform {rs_case} END) THEN
'GONE, BEFORE THE RECORDS START'
                 WHEN x.gone_on IS NULL THEN 'GONE, DATE NOT RECORDED'
                 WHEN x.gone_on > x.decided_on + {REMEDIATION_DAYS} THEN 'REMOVED LATE'
                 ELSE 'removed in time' END AS status
          FROM x WHERE x.platform IN ({pl}) ORDER BY 1, 2, 3, 4""").fetchall()
        show("4. Decisions carried out (every remove decision)", ["platform", "login", "access",
"review_id", "decided_on",
             "reviewer", "held_now_by", "account_enabled", "gone_on", "status"], rows, f"decisions
up to {AS_OF}; state on {AS_OF}",
             f"every remove decision in the review record; REMEDIATION_DAYS={REMEDIATION_DAYS}",
             "4_decisions_carried_out.csv")
        rep.follow_up["decisions"] = sum(r[9] in ("STILL HELD PAST DEADLINE", "REMOVED LATE") for r
in rows)
    else:
        print("\n== 4. Decisions carried out (every remove decision)\n   NOT RUN: REMEDIATION_DAYS
is not set")
        with open(run_dir / "4_decisions_carried_out.csv", "w", newline="", encoding="utf-8") as f:
            csv.writer(f).writerows([["status"], [f"NOT RUN: REMEDIATION_DAYS is not set; run at
{RUN_AT}"]])

    total = sum(rep.follow_up.values())
    parts = [f"{k.replace('_', ' ')} {rep.follow_up[k]}" if k in rep.follow_up else f"
{k.replace('_', ' ')} not run"
             for k in ["no_owner", "timing", "coverage", "decisions"]]
    print(f"\nNeeds follow-up: {', '.join(parts)}; input checks failed: {len(failed)}. "
          "The script produces evidence; the assessor decides whether the review was enough.")
    if "--strict" in args and (total or failed or not rem_set):
        return 1
    return 0
```

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

The runs

Each run is the script's full output, as the run file has it. Run 1 is the clean demo (`--demo`), run 2 has findings planted in the records (`--demo --plant`), run 3 has faults planted in the inputs (`--demo --plant-input`).

Run 1

```
Generated records written to generated/ (fixed rows; not a company's data). Demo settings:
REVIEW_INTERVAL_DAYS=184, REMEDIATION_DAYS=14.

Run at 2026-11-01 19:57:52 UTC. Exports taken 2026-10-30 (UTC); platforms exported: snowflake,
databricks, redshift; not checked: none.
This run's lists and packets go to evidence-out/run-20261101T195752Z/

Input checks
  [ok] dates, interval and platforms in the settings
  [ok] REMEDIATION_DAYS is set (PCI sets no deadline for addressing wrong access: it is yours)
  [ok] every file is there with every column the script reads
      read 8 rows into snowflake_users
      read 9 rows into snowflake_grants_to_users
      read 4 rows into snowflake_grants_to_roles
      read 6 rows into databricks_users
      read 1 rows into databricks_service_principals
      read 6 rows into databricks_group_members
      read 5 rows into databricks_audit
      read 5 rows into redshift_svv_user_info
      read 3 rows into redshift_svv_user_grants
      read 1 rows into redshift_svv_role_grants
      read 4 rows into redshift_sys_userlog
      read 10 rows into hr_people
      read 17 rows into login_map
      read 1 rows into third_party_accounts
      read 2 rows into system_accounts
      read 6 rows into reviews
      read 23 rows into review_decisions
  [ok] every date and time is written as YYYY-MM-DD [HH:MM:SS] (UTC)
  [ok] every value is one the script knows
  [ok] every Databricks removal row has a readable response status and workspace_id
  [ok] no login appears twice in one platform's account list
  [ok] no login is mapped to two people
  [ok] no person appears twice in the HR list, and no login twice in the third-party list
  [ok] every login in system_accounts is an account on its platform
  [ok] every login in third_party_accounts is an account on its platform
  [ok] every login that holds a role or group is in its platform's account list
      Databricks group members that are no user, service principal or group in the export: 0
      Snowflake grants older than the user who now has that name (an earlier holder's; left out): 0
  [ok] each review appears once per platform and none is dated after the export
  [ok] each platform's latest review has its packet file, and it still hashes to the recorded SHA-
256
```

```
  [ok] no access has two decisions in one review
  [ok] every decision belongs to a review in the review list
  [ok] every decision is about a row of the packet its review was given
  [ok] Databricks account records reach back to every remove decision of the last 365 days

18 input checks, 0 failed.

== 1. The review packet (7.2.4)
   period: state on 2026-10-30; filters: user accounts (not system accounts) on snowflake,
databricks, redshift; direct and inherited access; run at 2026-11-01 19:57:52 UTC; rows: 22;
sha256: 83290ebce29f91644032fd763fba8c045548643dfeb8da846139894eac12300b
   platform   | login             | access                  | held_how           | enabled |
owner_kind  | owner                                    | department | job_function  |
last_signin  | status
   databricks | ana@example.test  | finance-readers         | direct             | true    |
employee    | ana@example.test                         | Finance    | Analyst       | not
exported | ok
   databricks | ben@example.test  | sales-readers           | direct             | true    |
employee    | ben@example.test                         | Sales      | Analyst       | not
exported | ok
   databricks | chen@example.test | all-analysts            | via data-engineers | true    |
employee    | chen@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | chen@example.test | data-engineers          | direct             | true    |
employee    | chen@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | dara@example.test | all-analysts            | via data-engineers | true    |
employee    | dara@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | dara@example.test | data-engineers          | direct             | true    |
employee    | dara@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | ivy@example.test  | (no role or group read) | none               | false   |
employee    | ivy@example.test                         | Sales      | Analyst       | not
exported | ok
   databricks | sam@example.test  | (no role or group read) | none               | true    |
employee    | sam@example.test                         | IT         | Administrator | not
exported | ok
   redshift   | ana               | fin_ro                  | direct             | true    |
employee    | ana@example.test                         | Finance    | Analyst       | not
exported | ok
   redshift   | chen              | base_ro                 | via data_rw        | true    |
employee    | chen@example.test                        | Data       | Engineer      | not
exported | ok
   redshift   | chen              | data_rw                 | direct             | true    |
employee    | chen@example.test                        | Data       | Engineer      | not
exported | ok
   redshift   | kim               | base_ro                 | via data_rw        | true    |
employee    | kim@example.test                         | Data       | Engineer      | not
exported | ok
   redshift   | kim               | data_rw                 | direct             | true    |
employee    | kim@example.test                         | Data       | Engineer      | not
exported | ok
   redshift   | sam               | (superuser)             | superuser flag     | true    |
employee    | sam@example.test                         | IT         | Administrator | not
exported | ok
   snowflake  | admin_sam         | accountadmin            | direct             | true    |
employee    | sam@example.test                         | IT         | Administrator |
```

```
2026-10-28   | ok
   snowflake  | ana               | fin_analyst             | direct             | true    |
employee    | ana@example.test                                  | Finance    | Analyst       |
2026-10-28   | ok
   snowflake  | ben               | sales_reader            | direct             | true    |
employee    | ben@example.test                                  | Sales      | Analyst       |
2026-10-28   | ok
   snowflake  | chen              | analyst_base            | via data_engineer  | true    |
employee    | chen@example.test                                 | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | chen              | data_engineer           | direct             | true    |
employee    | chen@example.test                                 | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | dara              | analyst_base            | via data_engineer  | true    |
employee    | dara@example.test                                 | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | dara              | data_engineer           | direct             | true    |
employee    | dara@example.test                                 | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | vendor_bi         | sales_reader            | direct             | true    | third
party | ExampleVendor Analytics (sponsor ana@example.test) |            |               | 2026-10-
27   | ok
   what a review records (SHA-256 of each platform's platform, login, access and held_how):
      snowflake: 8 rows, sha256 39cdf7d0a3052560d07ab061b6da89bd7a1a273398224245d563c8a893e187cc
      databricks: 8 rows, sha256 e99b31509fc92e0132959c5d8c1f0793d696ab0ca91253b4b1ef3d4f2b05e5e5
      redshift: 6 rows, sha256 d9dd7d98b325ce8c5317a0e0453255e1f36e564531fc5f43b8964b7f59146a50
   accounts with no HR or third-party match (NO OWNER): 0

== 1b. System accounts (listed so none drops out; not part of 7.2.4)
   period: state on 2026-10-30; filters: Snowflake TYPE service types, Databricks service
principals, and your system_accounts list; run at 2026-11-01 19:57:52 UTC; rows: 3; sha256:
2aefab233778002a7f0c541505f89447a10b83adbaf71ac4d7377447f21be977
   platform   | login                                | enabled | access        | requirement
   databricks | 11111111-2222-3333-4444-555555555555 | true    | sales-readers | 7.2.5 and 7.2.5.1
   redshift   | svc_bi                               | true    |               | 7.2.5 and 7.2.5.1
   snowflake  | svc_load                             | true    | loader        | 7.2.5 and 7.2.5.1

== 2. Review timing (7.2.4: at least every six months)
   period: reviews completed up to 2026-10-30; filters: REVIEW_INTERVAL_DAYS=184; run at 2026-11-01
19:57:52 UTC; rows: 9; sha256: 33eb46c4e6acd840b9ebd3cfac58d53e61e469edae91fd718ff76906bbebf82a
   platform   | review_id              | completed  | days_since_previous | packet_sha256 |
acknowledged_by  | acknowledged_on | status
   snowflake  | R-2026-04              | 2026-04-14 |                     |               |
lee@example.test | 2026-04-16      | ok
   snowflake  | R-2026-09              | 2026-09-28 | 167                 | 6a52b4af9d20  |
lee@example.test | 2026-09-30      | ok
   snowflake  | (since the last review) | 2026-09-28 | 32                  |               |
|                 | ok
   databricks | R-2026-04              | 2026-04-14 |                     |               |
lee@example.test | 2026-04-16      | ok
   databricks | R-2026-09              | 2026-09-28 | 167                 | 733f41979e27  |
lee@example.test | 2026-09-30      | ok
   databricks | (since the last review) | 2026-09-28 | 32                  |               |
|                 | ok
   redshift   | R-2026-04              | 2026-04-14 |                     |               |
lee@example.test | 2026-04-16      | ok
   redshift   | R-2026-09              | 2026-09-28 | 167                 | df9f4d68b986  |
lee@example.test | 2026-09-30      | ok
```

```
   redshift   | (since the last review) | 2026-09-28 | 32                  |               |
|                 | ok

== 3. Review coverage (the latest review against the packet it was given)
   period: the latest review per platform up to 2026-10-30; filters: packet matched by its SHA-256;
one decision per packet row; run at 2026-11-01 19:57:52 UTC; rows: 23; sha256:
b37bf27438f4f8afa73e9c3c1f6bf9372157b4db3144ca65a2a3c995f955532a
   showing the 0 row(s) that need a look; all 23 are in 3_review_coverage.csv
   snowflake: 8 packet rows, 0 not reviewed, 0 reviewed by the account holder, 0 with no reviewer
   databricks: 9 packet rows, 0 not reviewed, 0 reviewed by the account holder, 0 with no reviewer
   redshift: 6 packet rows, 0 not reviewed, 0 reviewed by the account holder, 0 with no reviewer
   access rows in today's packet that were not in the latest review's packet (they go into the next
review; a count to look at, not a finding): databricks 1; redshift 2; snowflake 1

== 4. Decisions carried out (every remove decision)
   period: decisions up to 2026-10-30; state on 2026-10-30; filters: every remove decision in the
review record; REMEDIATION_DAYS=14; run at 2026-11-01 19:57:52 UTC; rows: 5; sha256:
8ed9535b5de80a0246a1c941d4c1253554266875dc34a3316ff346945837ef48
   platform   | login             | access          | review_id | decided_on | reviewer         |
held_now_by | account_enabled | gone_on    | status
   databricks | dara@example.test | finance-readers | R-2026-09 | 2026-09-28 | lee@example.test |
| true            | 2026-09-29 | removed in time
   databricks | ivy@example.test  | sales-readers   | R-2026-09 | 2026-09-28 | lee@example.test |
| false           | 2026-09-29 | removed in time
   redshift   | jon               | base_ro         | R-2026-09 | 2026-09-28 | lee@example.test |
| no account      | 2026-09-30 | removed in time
   redshift   | jon               | data_rw         | R-2026-09 | 2026-09-28 | lee@example.test |
| no account      | 2026-09-30 | removed in time
   snowflake  | dara              | fin_analyst     | R-2026-09 | 2026-09-28 | lee@example.test |
| true            | 2026-09-30 | removed in time

Needs follow-up: no owner 0, timing 0, coverage 0, decisions 0; input checks failed: 0. The script
produces evidence; the assessor decides whether the review was enough.
```

Run 2

```
Generated records written to generated/ (fixed rows; not a company's data). Demo settings:
REVIEW_INTERVAL_DAYS=184, REMEDIATION_DAYS=14.

Run at 2026-11-01 19:57:53 UTC. Exports taken 2026-10-30 (UTC); platforms exported: snowflake,
databricks, redshift; not checked: none.
This run's lists and packets go to evidence-out/run-20261101T195753Z/

Input checks
  [ok] dates, interval and platforms in the settings
  [ok] REMEDIATION_DAYS is set (PCI sets no deadline for addressing wrong access: it is yours)
  [ok] every file is there with every column the script reads
      read 9 rows into snowflake_users
      read 10 rows into snowflake_grants_to_users
      read 4 rows into snowflake_grants_to_roles
      read 6 rows into databricks_users
      read 1 rows into databricks_service_principals
      read 7 rows into databricks_group_members
      read 4 rows into databricks_audit
      read 6 rows into redshift_svv_user_info
      read 4 rows into redshift_svv_user_grants
```

```
      read 1 rows into redshift_svv_role_grants
      read 4 rows into redshift_sys_userlog
      read 10 rows into hr_people
      read 18 rows into login_map
      read 1 rows into third_party_accounts
      read 2 rows into system_accounts
      read 5 rows into reviews
      read 22 rows into review_decisions
  [ok] every date and time is written as YYYY-MM-DD [HH:MM:SS] (UTC)
  [ok] every value is one the script knows
  [ok] every Databricks removal row has a readable response status and workspace_id
  [ok] no login appears twice in one platform's account list
  [ok] no login is mapped to two people
  [ok] no person appears twice in the HR list, and no login twice in the third-party list
  [ok] every login in system_accounts is an account on its platform
  [ok] every login in third_party_accounts is an account on its platform
  [ok] every login that holds a role or group is in its platform's account list
      Databricks group members that are no user, service principal or group in the export: 0
      Snowflake grants older than the user who now has that name (an earlier holder's; left out): 0
  [ok] each review appears once per platform and none is dated after the export
  [ok] each platform's latest review has its packet file, and it still hashes to the recorded SHA-
256
  [ok] no access has two decisions in one review
  [ok] every decision belongs to a review in the review list
  [ok] every decision is about a row of the packet its review was given
  [ok] Databricks account records reach back to every remove decision of the last 365 days

18 input checks, 0 failed.

== 1. The review packet (7.2.4)
   period: state on 2026-10-30; filters: user accounts (not system accounts) on snowflake,
databricks, redshift; direct and inherited access; run at 2026-11-01 19:57:53 UTC; rows: 26;
sha256: cd5ab63505cf6be82cc267fcd4fcf0244662442f613f790c96ce75019c1cca03
   platform   | login             | access                  | held_how           | enabled |
owner_kind  | owner                                    | department | job_function  |
last_signin  | status
   databricks | ana@example.test  | finance-readers         | direct             | true    |
employee    | ana@example.test                         | Finance    | Analyst       | not
exported | ok
   databricks | ben@example.test  | sales-readers           | direct             | true    |
employee    | ben@example.test                         | Sales      | Analyst       | not
exported | ok
   databricks | chen@example.test | all-analysts            | via data-engineers | true    |
employee    | chen@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | chen@example.test | data-engineers          | direct             | true    |
employee    | chen@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | dara@example.test | all-analysts            | via data-engineers | true    |
employee    | dara@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | dara@example.test | data-engineers          | direct             | true    |
employee    | dara@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | ivy@example.test  | sales-readers           | direct             | false   |
employee    | ivy@example.test                         | Sales      | Analyst       | not
exported | ok
   databricks | sam@example.test  | (no role or group read) | none               | true    |
```

```
employee    | sam@example.test                                    | IT         | Administrator | not
exported | ok
   redshift   | ana               | fin_ro                  | direct             | true    |
employee    | ana@example.test                                    | Finance    | Analyst       | not
exported | ok
   redshift   | chen              | base_ro                 | via data_rw        | true    |
employee    | chen@example.test                                   | Data       | Engineer      | not
exported | ok
   redshift   | chen              | data_rw                 | direct             | true    |
employee    | chen@example.test                                   | Data       | Engineer      | not
exported | ok
   redshift   | jon_x             | base_ro                 | via data_rw        | true    |
employee    | jon@example.test                                    | Data       | Engineer      | not
exported | ok
   redshift   | jon_x             | data_rw                 | direct             | true    |
employee    | jon@example.test                                    | Data       | Engineer      | not
exported | ok
   redshift   | kim               | base_ro                 | via data_rw        | true    |
employee    | kim@example.test                                    | Data       | Engineer      | not
exported | ok
   redshift   | kim               | data_rw                 | direct             | true    |
employee    | kim@example.test                                    | Data       | Engineer      | not
exported | ok
   redshift   | sam               | (superuser)             | superuser flag     | true    |
employee    | sam@example.test                                    | IT         | Administrator | not
exported | ok
   snowflake  | admin_sam         | accountadmin            | direct             | true    |
employee    | sam@example.test                                    | IT         | Administrator |
2026-10-28   | ok
   snowflake  | ana               | fin_analyst             | direct             | true    |
employee    | ana@example.test                                    | Finance    | Analyst       |
2026-10-28   | ok
   snowflake  | ben               | sales_reader            | direct             | true    |
employee    | ben@example.test                                    | Sales      | Analyst       |
2026-10-28   | ok
   snowflake  | chen              | analyst_base            | via data_engineer  | true    |
employee    | chen@example.test                                   | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | chen              | data_engineer           | direct             | true    |
employee    | chen@example.test                                   | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | dara              | analyst_base            | via data_engineer  | true    |
employee    | dara@example.test                                   | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | dara              | data_engineer           | direct             | true    |
employee    | dara@example.test                                   | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | dara              | fin_analyst             | direct             | true    |
employee    | dara@example.test                                   | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | ghost             | sales_reader            | direct             | true    | NO
OWNER    |                                                       |            |               | 2026-
10-12   | NO OWNER
   snowflake  | vendor_bi         | sales_reader            | direct             | true    | third
party | ExampleVendor Analytics (sponsor ana@example.test) |            |               | 2026-10-
27   | ok
   what a review records (SHA-256 of each platform's platform, login, access and held_how):
      snowflake: 10 rows, sha256 005e9859cffed42b213717c3ad5dbf63a8013672798de6fd95493ecebc846923
```

```
      databricks: 8 rows, sha256 7f124a598b548b03d4f3cec3a9ced28951c0922bbd454acdebf81982193f9658
      redshift: 8 rows, sha256 7327d2692c161b6fa51217cf68b168181e9123710f57bbea2c1022e9ba169bc5
   accounts with no HR or third-party match (NO OWNER): 1

== 1b. System accounts (listed so none drops out; not part of 7.2.4)
   period: state on 2026-10-30; filters: Snowflake TYPE service types, Databricks service
principals, and your system_accounts list; run at 2026-11-01 19:57:53 UTC; rows: 3; sha256:
2aefab233778002a7f0c541505f89447a10b83adbaf71ac4d7377447f21be977
   platform   | login                                | enabled | access        | requirement
   databricks | 11111111-2222-3333-4444-555555555555 | true    | sales-readers | 7.2.5 and 7.2.5.1
   redshift   | svc_bi                               | true    |               | 7.2.5 and 7.2.5.1
   snowflake  | svc_load                             | true    | loader        | 7.2.5 and 7.2.5.1

== 2. Review timing (7.2.4: at least every six months)
   period: reviews completed up to 2026-10-30; filters: REVIEW_INTERVAL_DAYS=184; run at 2026-11-01
19:57:53 UTC; rows: 8; sha256: 66cf44f7945c60d7d3f7a5b7fa20fb94ad6d8e36a04e2f3e5d04d2be1d7f1411
   platform   | review_id               | completed  | days_since_previous | packet_sha256 |
acknowledged_by  | acknowledged_on | status
   snowflake  | R-2026-04               | 2026-03-20 |                     |               |
lee@example.test | 2026-04-16      | ok
   snowflake  | R-2026-09               | 2026-09-28 | 192                 | 6a52b4af9d20  |
lee@example.test | 2026-09-30      | GAP OVER 184 DAYS
   snowflake  | (since the last review) | 2026-09-28 | 32                  |               |
|                 | ok
   databricks | R-2026-04               | 2026-04-14 |                     |               |
lee@example.test | 2026-04-16      | ok
   databricks | R-2026-09               | 2026-09-28 | 167                 | 733f41979e27  |
lee@example.test | 2026-09-30      | ok
   databricks | (since the last review) | 2026-09-28 | 32                  |               |
|                 | ok
   redshift   | R-2026-04               | 2026-04-14 |                     | df9f4d68b986  |
|                 | NO ACKNOWLEDGMENT
   redshift   | (since the last review) | 2026-04-14 | 199                 |               |
|                 | OVERDUE

== 3. Review coverage (the latest review against the packet it was given)
   period: the latest review per platform up to 2026-10-30; filters: packet matched by its SHA-256;
one decision per packet row; run at 2026-11-01 19:57:53 UTC; rows: 23; sha256:
c75f0d6c70b9984832984941eb4b84ef9477d7912d6b9a8c62758fe494cd9af7
   showing the 3 row(s) that need a look; all 23 are in 3_review_coverage.csv
   platform  | review_id | login     | access       | decision | reviewer         | decided_on |
status
   snowflake | R-2026-09 | admin_sam | accountadmin | keep     | sam@example.test | 2026-09-28 |
REVIEWED OWN ACCESS
   snowflake | R-2026-09 | ben       | sales_reader | keep     |                  | 2026-09-28 | NO
REVIEWER
   snowflake | R-2026-09 | chen      | analyst_base |          |                  |            |
NOT REVIEWED
   snowflake: 8 packet rows, 1 not reviewed, 1 reviewed by the account holder, 1 with no reviewer
   databricks: 9 packet rows, 0 not reviewed, 0 reviewed by the account holder, 0 with no reviewer
   redshift: 6 packet rows, 0 not reviewed, 0 reviewed by the account holder, 0 with no reviewer
   access rows in today's packet that were not in the latest review's packet (they go into the next
review; a count to look at, not a finding): redshift 4; snowflake 2

== 4. Decisions carried out (every remove decision)
   period: decisions up to 2026-10-30; state on 2026-10-30; filters: every remove decision in the
review record; REMEDIATION_DAYS=14; run at 2026-11-01 19:57:53 UTC; rows: 5; sha256:
040a964aea3f010a5e06f4fdae30cea748144bd991d05079206abfc8b0b394ac
```

```
   platform   | login             | access          | review_id | decided_on | reviewer         |
held_now_by      | account_enabled | gone_on    | status
   databricks | dara@example.test | finance-readers | R-2026-09 | 2026-09-28 | lee@example.test |
| true            | 2026-10-20 | REMOVED LATE
   databricks | ivy@example.test  | sales-readers   | R-2026-09 | 2026-09-28 | lee@example.test |
ivy@example.test | false           |            | STILL HELD PAST DEADLINE
   redshift   | jon               | base_ro         | R-2026-04 | 2026-04-14 | lee@example.test |
jon_x            | true            |            | STILL HELD PAST DEADLINE
   redshift   | jon               | data_rw         | R-2026-04 | 2026-04-14 | lee@example.test |
jon_x            | true            |            | STILL HELD PAST DEADLINE
   snowflake  | dara              | fin_analyst     | R-2026-09 | 2026-09-28 | lee@example.test |
dara             | true            |            | STILL HELD PAST DEADLINE

Needs follow-up: no owner 1, timing 3, coverage 3, decisions 5; input checks failed: 0. The script
produces evidence; the assessor decides whether the review was enough.
```

Run 3

```
Generated records written to generated/ (fixed rows; not a company's data). Demo settings:
REVIEW_INTERVAL_DAYS=184, REMEDIATION_DAYS=None.

Run at 2026-11-01 19:57:53 UTC. Exports taken 2026-10-30 (UTC); platforms exported: snowflake,
databricks, redshift; not checked: none.
This run's lists and packets go to evidence-out/run-20261101T195753Z/

Input checks
  [ok] dates, interval and platforms in the settings
  [FAIL] REMEDIATION_DAYS is set (PCI sets no deadline for addressing wrong access: it is yours):
not set to a whole number of days: list 4 will not run
  [FAIL] every file is there with every column the script reads: redshift_svv_role_grants.csv has
no column granted_role_name
      read 9 rows into snowflake_users
      read 10 rows into snowflake_grants_to_users
      read 4 rows into snowflake_grants_to_roles
      read 6 rows into databricks_users
      read 1 rows into databricks_service_principals
      read 7 rows into databricks_group_members
      read 5 rows into databricks_audit
      read 5 rows into redshift_svv_user_info
      read 3 rows into redshift_svv_user_grants
      read 0 rows into redshift_svv_role_grants
      read 5 rows into redshift_sys_userlog
      read 11 rows into hr_people
      read 18 rows into login_map
      read 2 rows into third_party_accounts
      read 3 rows into system_accounts
      read 8 rows into reviews
      read 26 rows into review_decisions
  [FAIL] every date and time is written as YYYY-MM-DD [HH:MM:SS] (UTC): not readable, so left out:
reviews.completed_date: 2
  [FAIL] every value is one the script knows: snowflake_users.disabled not true or false: 1;
snowflake_users.type not a type the script knows: 1; review_decisions.decision not keep or remove:
1; redshift_sys_userlog.action not alter, create, drop or rename: 1
  [FAIL] every Databricks removal row has a readable response status and workspace_id: 1 row(s)
whose response is not JSON with a status code
  [FAIL] no login appears twice in one platform's account list: snowflake/ana x2
```

```
  [FAIL] no login is mapped to two people: snowflake/ana
  [FAIL] no person appears twice in the HR list, and no login twice in the third-party list:
ana@example.test
  [FAIL] every login in system_accounts is an account on its platform: matches no account:
redshift/svc_billing
  [FAIL] every login in third_party_accounts is an account on its platform: matches no account:
snowflake/no_such_vendor_login
  [FAIL] every login that holds a role or group is in its platform's account list: holds access,
not in the account list: snowflake/zed
      Databricks group members that are no user, service principal or group in the export: 1
      Snowflake grants older than the user who now has that name (an earlier holder's; left out): 0
  [FAIL] each review appears once per platform and none is dated after the export: after 2026-10-
30: snowflake/R-2027-01
  [FAIL] each platform's latest review has its packet file, and it still hashes to the recorded
SHA-256: snowflake/R-2026-09:
packet_snowflake_6a52b4af9d2052aa1cf2f71b4227270dd194908b6d53fd9bf7b5398b1606809b.csv was edited
(its rows hash to 1ffbf897f57e...)
  [FAIL] no access has two decisions in one review: snowflake/R-2026-09/ana/fin_analyst
  [FAIL] every decision belongs to a review in the review list: no such review: redshift/R-9999
  [FAIL] every decision is about a row of the packet its review was given: not in the packet:
databricks/R-2026-09/nobody@example.test/finance-readers
  [FAIL] Databricks account records reach back to every remove decision of the last 365 days:
earliest such remove decision 2026-09-28; earliest 'accounts' audit row 2026-09-29

18 input checks, 17 failed.  ** INCOMPLETE: 17 input check(s) failed; read the lists with that in
mind **

== 1. The review packet (7.2.4)
   period: state on 2026-10-30; filters: user accounts (not system accounts) on snowflake,
databricks, redshift; direct and inherited access; run at 2026-11-01 19:57:53 UTC; rows: 25;
sha256: a9804f390d4e399516acd270f5863f999e1b21da5249adc0fecc0170831c4ebf; INCOMPLETE
   platform   | login             | access                  | held_how           | enabled |
owner_kind  | owner                                    | department | job_function  |
last_signin  | status
   databricks | ana@example.test  | finance-readers         | direct             | true    |
employee    | ana@example.test                         | Finance    | Analyst       | not
exported | ok
   databricks | ana@example.test  | finance-readers         | direct             | true    |
employee    | ana@example.test                         | Finance    | Analyst       | not
exported | ok
   databricks | ben@example.test  | sales-readers           | direct             | true    |
employee    | ben@example.test                         | Sales      | Analyst       | not
exported | ok
   databricks | chen@example.test | all-analysts            | via data-engineers | true    |
employee    | chen@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | chen@example.test | data-engineers          | direct             | true    |
employee    | chen@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | dara@example.test | all-analysts            | via data-engineers | true    |
employee    | dara@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | dara@example.test | data-engineers          | direct             | true    |
employee    | dara@example.test                        | Data       | Engineer      | not
exported | ok
   databricks | ivy@example.test  | (no role or group read) | none               | false   |
employee    | ivy@example.test                         | Sales      | Analyst       | not
exported | ok
```

```
   databricks | sam@example.test  | (no role or group read) | none               | true    |
employee    | sam@example.test                                  | IT         | Administrator | not
exported | ok
   redshift   | ana               | fin_ro                  | direct             | true    |
employee    | ana@example.test                                  | Finance    | Analyst       | not
exported | ok
   redshift   | ana               | fin_ro                  | direct             | true    |
employee    | ana@example.test                                  | Finance    | Analyst       | not
exported | ok
   redshift   | chen              | data_rw                 | direct             | true    |
employee    | chen@example.test                                 | Data       | Engineer      | not
exported | ok
   redshift   | kim               | data_rw                 | direct             | true    |
employee    | kim@example.test                                  | Data       | Engineer      | not
exported | ok
   redshift   | sam               | (superuser)             | superuser flag     | true    |
employee    | sam@example.test                                  | IT         | Administrator | not
exported | ok
   snowflake  | admin_sam         | accountadmin            | direct             | true    |
employee    | sam@example.test                                  | IT         | Administrator |
2026-10-28   | ok
   snowflake  | ana               | fin_analyst             | direct             | true    |
employee    | ana@example.test                                  | Finance    | Analyst       |
2026-10-28   | ok
   snowflake  | ana               | fin_analyst             | direct             | true    |
employee    | ana@example.test                                  | Finance    | Analyst       |
2026-10-28   | ok
   snowflake  | ana               | fin_analyst             | direct             | true    |
employee    | ana@example.test                                  | Finance    | Analyst       |
2026-10-28   | ok
   snowflake  | ana               | fin_analyst             | direct             | true    |
employee    | ana@example.test                                  | Finance    | Analyst       |
2026-10-28   | ok
   snowflake  | ben               | sales_reader            | direct             | true    |
employee    | ben@example.test                                  | Sales      | Analyst       |
2026-10-28   | ok
   snowflake  | chen              | analyst_base            | via data_engineer  | true    |
employee    | chen@example.test                                 | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | chen              | data_engineer           | direct             | true    |
employee    | chen@example.test                                 | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | dara              | analyst_base            | via data_engineer  | true    |
employee    | dara@example.test                                 | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | dara              | data_engineer           | direct             | true    |
employee    | dara@example.test                                 | Data       | Engineer      |
2026-10-28   | ok
   snowflake  | vendor_bi         | sales_reader            | direct             | true    | third
party | ExampleVendor Analytics (sponsor ana@example.test) |            |               | 2026-10-
27   | ok
   what a review records (SHA-256 of each platform's platform, login, access and held_how):
      snowflake: 11 rows, sha256 daeb16207752e722c567d0cf7a6f6d2a621a38e693a822aefea7f5be376d99c9
      databricks: 9 rows, sha256 95f873868ca5173d492801c3fc8c2aac110f8332ab7c33af93d678c51b7fb2a8
      redshift: 5 rows, sha256 8dee4398e0c2497176267f9255865893333a08c8c16ebd21b0a4860e3e28cbcb
   accounts with no HR or third-party match (NO OWNER): 0

== 1b. System accounts (listed so none drops out; not part of 7.2.4)
```

```
   period: state on 2026-10-30; filters: Snowflake TYPE service types, Databricks service
principals, and your system_accounts list; run at 2026-11-01 19:57:53 UTC; rows: 3; sha256:
2aefab233778002a7f0c541505f89447a10b83adbaf71ac4d7377447f21be977; INCOMPLETE
   platform   | login                                | enabled | access        | requirement
   databricks | 11111111-2222-3333-4444-555555555555 | true    | sales-readers | 7.2.5 and 7.2.5.1
   redshift   | svc_bi                               | true    |               | 7.2.5 and 7.2.5.1
   snowflake  | svc_load                             | true    | loader        | 7.2.5 and 7.2.5.1

== 2. Review timing (7.2.4: at least every six months)
   period: reviews completed up to 2026-10-30; filters: REVIEW_INTERVAL_DAYS=184; run at 2026-11-01
19:57:53 UTC; rows: 8; sha256: dcee0d8f11687623c8d3730fccab44c3e30ba53a965e028c6cd843f0b9a851df;
INCOMPLETE
   platform   | review_id               | completed  | days_since_previous | packet_sha256 |
acknowledged_by  | acknowledged_on | status
   snowflake  | R-2026-04               | 2026-04-14 |                     |               |
lee@example.test | 2026-04-16      | ok
   snowflake  | R-2026-09               | 2026-09-28 | 167                 | 6a52b4af9d20  |
lee@example.test | 2026-09-30      | ok
   snowflake  | (since the last review) | 2026-09-28 | 32                  |               |
|                 | ok
   databricks | R-2026-09               | 2026-09-28 |                     | 733f41979e27  |
lee@example.test | 2026-09-30      | ok
   databricks | (since the last review) | 2026-09-28 | 32                  |               |
|                 | ok
   redshift   | R-2026-04               | 2026-04-14 |                     |               |
lee@example.test | 2026-04-16      | ok
   redshift   | R-2026-09               | 2026-09-28 | 167                 | df9f4d68b986  |
lee@example.test | 2026-09-30      | ok
   redshift   | (since the last review) | 2026-09-28 | 32                  |               |
|                 | ok

== 3. Review coverage (the latest review against the packet it was given)
   period: the latest review per platform up to 2026-10-30; filters: packet matched by its SHA-256;
one decision per packet row; run at 2026-11-01 19:57:53 UTC; rows: 15; sha256:
911c32340af28f392539f2eab80ffeaabcb9515b5ddfbec3c7e0442f34639467; INCOMPLETE
   showing the 0 row(s) that need a look; all 15 are in 3_review_coverage.csv
   snowflake: NOT CHECKED: no latest review with a packet that could be read
   databricks: 9 packet rows, 0 not reviewed, 0 reviewed by the account holder, 0 with no reviewer
   redshift: 6 packet rows, 0 not reviewed, 0 reviewed by the account holder, 0 with no reviewer
   access rows in today's packet that were not in the latest review's packet (they go into the next
review; a count to look at, not a finding): databricks 1; redshift 1

== 4. Decisions carried out (every remove decision)
   NOT RUN: REMEDIATION_DAYS is not set

Needs follow-up: no owner 0, timing 0, coverage 0, decisions not run; input checks failed: 17. The
script produces evidence; the assessor decides whether the review was enough.
```

How to read the results

- **Input checks first.** If any check failed, the lists say INCOMPLETE. A list built on a record with an unreadable date or an edited packet is not a record to rely on until the input check is fixed.

- **List 1 is the packet.** It is what the reviewer should have been given. Record its SHA-256 on the review's row. Later you can check that the packet file you kept is the one the review recorded; the hash shows which file that was, not what the reviewer looked at. NO OWNER is a question for whoever runs the platform: an account nobody can name is hard to review.
- **List 2 shows dates, not a verdict.** A gap over 184 days is a question for the review's owner. The standard says a late activity is not automatically a failure where a documented process caught and handled it; the assessor decides, not the script.
- **List 3 shows what the review decided on.** NOT REVIEWED is a packet row with no decision. The packet is the script's own and may hold rows your reviewer reasonably treated as one (an inherited role decided with its parent); a flag is a question, not a finding. REVIEWED OWN ACCESS means the reviewer was the account's owner.
- **List 4 is the one to read first.** A remove decision whose access is still there, past the days you set, is a decision the platform's records do not show as carried out. Check STILL HELD rows against the account state first: a deactivated account still holds the group.
- **Not checked is not clean.** A platform left out of `PLATFORMS_EXPORTED` is printed as not checked, and list 4 prints NOT RUN while REMEDIATION_DAYS is blank.

Run it on your own data

Save each export query's result as a CSV named as in FILES, in one folder, with the platform's own column names (a header in capitals is fine), and write your own lists (hr_people, login_map, third_party_accounts, system_accounts, reviews and review_decisions) as CSVs with the columns FILES names. Put each review's packet file, as the script wrote it to the packets folder of its run in evidence-out, in a folder named packets inside that folder. Then change four lines at the top of the script: set EXPORT_DIR to the path of that folder, AS_OF to the day you took the exports, PLATFORMS_EXPORTED to the platforms you exported, and REMEDIATION_DAYS to the number of days you allow. Change nothing else.

We tested those steps on a second, separate set of generated exports: other logins in mixed case (a login written Mia.Ortiz in one file and MIA.ORTIZ in another), upper-case Snowflake headers, a Snowflake user with an empty TYPE, a blank email in the login map, a blank reviewer, and no Databricks files at all, with Databricks left out of `PLATFORMS_EXPORTED` (you do not need the files for a platform you leave out). The script listed Databricks as not checked, ran 17 input checks with none failing, and reported one account with no owner, three timing flags (a Snowflake gap of 211 days, a Redshift review nobody acknowledged, and Redshift overdue at 243 days), three coverage flags and two remove decisions still held past the 10 days that test set. That test is also generated data.

Checklist

Done	Item	Where
□	Export the 11 platform files and write the 6 lists you keep; name each as the table in step 1 does	Step 1
□	Set EXPORT_DIR, AS_OF, PLATFORMS_EXPORTED and REMEDIATION_DAYS (your own number of days)	Step 2
□	Run once; confirm every input check passes before reading a list	Step 4, How to read
□	Check one Snowflake role-in-role row and one Databricks nested group against your own account	What you will need
□	Give the reviewer each platform's packet; record its SHA-256 on the review's row	Step 1
□	Keep each packet file in the packets folder next to your exports	Step 1
□	Record the review, who acknowledged it for management and when	Step 1
□	Read list 1: every NO OWNER account has an owner or a reason	How to read
□	Read list 2: every GAP, OVERDUE and NO ACKNOWLEDGMENT has a documented explanation	How to read
□	Read list 3: every NOT REVIEWED, REVIEWED OWN ACCESS and NO REVIEWER row is explained or redone	How to read
□	Read list 4: every STILL HELD and REMOVED LATE row is carried out or explained	How to read
□	Keep every run's lists, packets and hashes	When this is not enough

Method

- **Input:** generated records written by the script as CSV with column names taken from each platform's documentation (the links in step 1), with fixed rows and no randomness. The HR list, the login map, the third-party and system account lists and the review record are generated too. The people, roles and counts are invented.
- **Dates:** the demo export day is 2026-10-30, UTC. The clean run's reviews are on 2026-04-14 and 2026-09-28; the planted run moves some of them, as the step says.
- **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 demo three times and got the same list hashes.

- **Our assumptions, to check against one of your own rows:** a Snowflake role-in-role grant is a `GRANTS_TO_ROLES` row with `GRANTED_ON` ROLE and `PRIVILEGE` USAGE; a Snowflake grant with no `DELETED_ON` to a dropped user does not stay attached to a new user of the same name, unless it was created after that user (the page does not say); a Databricks group member's `value` is the member's `id`, and a member that is another group's id is a nested group; a successful Databricks row has the status 200 in `response`; the Databricks audit rows carry `targetUserName` and `targetGroupName` in `request_params`, one user per row (the reference does not give the format for `removePrincipalsFromGroup`, which can remove several users at once: if it lists them together, that removal shows as GONE, DATE NOT RECORDED); and the platforms' column names are as the linked pages list them on 2026-11-01.

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.
- **PCI sets the questions, not your numbers.** The deadline for addressing wrong access is yours. The script cannot tell you whether your numbers are right, whether a late review was handled by a documented process, or whether the review as a whole was enough. The assessor decides.
- **The review itself is not in the platforms.** The script checks the review's own record against what the platforms hold. It cannot tell whether a reviewer thought about a row, or whether management really confirmed it.
- **The records may not hold what you need.** Redshift's role revokes have no date in the views we read, and its user log may keep only about seven days, or fewer (the AWS reference gives seven days for STL views and none for this one), so keep copies. Snowflake's history is kept for 365 days. Databricks keeps its audit table for 365 days, and workspace-level events stay in each workspace's region. Databricks group members must be collected group by group, and a member the script cannot place is only counted. Last sign-in is read for Snowflake only. A Snowflake `ALTER USER ... RENAME TO` is not in these exports, so a renamed Snowflake user is followed only if the login map names both logins.
- **Access outside the holdings read here is not in the packet:** object privileges granted straight to a user, Snowflake database roles, Redshift user groups, the Databricks account admin role and workspace-local groups (each workspace's admins group included), a Redshift superuser's reach beyond the flag, and access in tools outside the three platforms. An account shown as (no role or group read) may still hold access through one of them.
- **Packet identity covers four columns.** The SHA-256 covers each row's platform, login, access and how it is held. The owner, department and job function printed beside them are not part

of it.

- **Read, not enforced.** The script reads exports. It never connects to a platform and never changes an account.
- **Evidence, not a verdict.** The script produces evidence. It never says a requirement is met, and it does not know what any assessor will accept.

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 (fixed rows) with column names taken from each platform's documentation; not a company's data

When this is not enough

- Accounts outside the three platforms: an identity provider, a BI tool or a cloud console. The four lists see only what the three platforms record.
- A review that has no record. If the review lives in email threads, there is no `reviews.csv` to check, and writing one is the first job.
- A policy that is silent. If your policy has no time for removal, the script has nothing to measure against, and choosing one is for you and your assessor.
- A single run. One export shows one day. Run it on a schedule; each run keeps its own folder of lists, packets and hashes.
- Platforms other than these three, and the rest of requirement 7 and the other requirements of the standard.

Sources

Every document linked in this guide, read on 2026-11-01. PCI DSS v4.0.1 is licensed text: we name requirement 7.2.4 by its id and describe it in our own words.

- PCI Security Standards Council: document library (read 2026-11-01)

- Snowflake: ACCOUNT_USAGE.USERS (read 2026-11-01)
- Snowflake: ACCOUNT_USAGE.GRANTS_TO_USERS (read 2026-11-01)
- Snowflake: user management (read 2026-11-01)
- Snowflake: Account Usage (read 2026-11-01)
- Snowflake: ACCOUNT_USAGE.GRANTS_TO_ROLES (read 2026-11-01)
- Snowflake: overview of access control (read 2026-11-01)
- Databricks: list account users (read 2026-11-01)
- Databricks: list account service principals (read 2026-11-01)
- Databricks: get account group (read 2026-11-01)
- Databricks: audit log system table (read 2026-11-01)
- Databricks: audit log reference (read 2026-11-01)
- Azure Databricks: audit log reference (read 2026-11-01)
- Databricks: system tables overview (read 2026-11-01)
- Amazon Redshift: SVV_USER_INFO (read 2026-11-01)
- Amazon Redshift: SVV_USER_GRANTS (read 2026-11-01)
- Amazon Redshift: SVV_ROLE_GRANTS (read 2026-11-01)
- Amazon Redshift: SYS_USERLOG (read 2026-11-01)
- Amazon Redshift: system tables and views reference (read 2026-11-01)

Get the free script and see your own review gaps

The script is free. It reads CSV exports from your platforms and your own review record. It only reads, and it never writes.

Seeing access changes every day is a different job. 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 lists every access change and who made it, including access someone gave by hand. No platform passwords stored.

See how Data Control Tower works