DATA CONTROL TOWER · GUIDE

NIST SP 800-53 AC-2 evidence for Snowflake, Databricks, Redshift

Four lists from Snowflake, Databricks and Redshift exports, against time periods you set.

Published 2026-11-01 · Online version and downloads: www.datameaning.com/guides/nist-800-53-ac-2-account-evidence-for-snowflake-databricks-redshift/

AC-2 sets the questions and leaves every time period to you. Our free script builds four lists from platform records: enabled accounts with no recent sign-in, temporary accounts past their end date, transfers that kept old access, and the account changes it could read in the records. We ran it on generated records. It produces evidence; your assessor decides.

What you will need

- Snowflake: `ACCOUNT_USAGE.USERS` and `GRANTS_TO_USERS` can lag by up to 120 minutes (Snowflake: ACCOUNT_USAGE.USERS (read 2026-11-01); Snowflake: ACCOUNT_USAGE.GRANTS_TO_USERS (read 2026-11-01)), and `QUERY_HISTORY` by up to 45 minutes and covers the last 365 days; its `start_time` is in the local time zone, so export from a session set to UTC (Snowflake: ACCOUNT_USAGE.QUERY_HISTORY (read 2026-11-01)). Query text longer than 100K characters is truncated. Snowflake's `EXPIRES_AT` is the date and time when the user's status is set to EXPIRED and the user can no longer log in (Snowflake: ACCOUNT_USAGE.USERS (read 2026-11-01)).
- Databricks: the audit table has the columns `event_time`, `event_date`, `user_identity`, `service_name`, `action_name`, `request_params` and `response`, and account-level events are recorded with `workspace_id` 0 (Databricks: audit log system table (read 2026-11-01)). The actions we read, such as `add`, `deactivateUser` and `addPrincipalToGroup`, are named on the audit log reference (Databricks: audit log reference (read 2026-11-01); Azure Databricks: audit log reference (read 2026-11-01), which adds the Microsoft Entra ID sign-ins `aadBrowserLogin` and `aadTokenLogin`). Most of them are logged twice over: at account level, where `add` means a user is added to the account, and at workspace level, where it

means a user is added to one workspace. Databricks keeps `system.access.audit` for 365 days, "Regional for workspace-level events. Global for account-level events." (Databricks: system tables overview (read 2026-11-01)). The SCIM users list gives `id`, `userName` and `active` but has no groups field (Databricks: list account users (read 2026-11-01)), and the groups list no longer returns members: you get them from each group's details, where each member has a `value` (Databricks: list account groups (read 2026-11-01)).

- Redshift: `SYS_USERLOG` and `SYS_CONNECTION_LOG` are visible only to superusers (Amazon Redshift: SYS_USERLOG (read 2026-11-01); Amazon Redshift: SYS_CONNECTION_LOG (read 2026-11-01)). `SVV_USER_INFO` is visible to all users, but only superusers see all rows (Amazon Redshift: SVV_USER_INFO (read 2026-11-01)), and its page lists no column for a disabled state, so the script treats every user it lists as enabled. `SVV_USER_GRANTS` lists roles explicitly granted to users, visible to superusers and users with the ACCESS SYSTEM TABLE permission (Amazon Redshift: SVV_USER_GRANTS (read 2026-11-01)). AWS's reference says "STL system views retain seven days of log history" (Amazon Redshift: system tables and views reference (read 2026-11-01)), and the `SYS_CONNECTION_LOG` and `SYS_USERLOG` pages state no period of their own, so plan for about seven days. For an `INACTIVE_DAYS` longer than that, build `redshift_sys_connection_log.csv`, with the three columns in step 1, from the connection log your audit logging writes to Amazon S3 or CloudWatch (Amazon Redshift: database audit logging (read 2026-11-01)), or from daily copies you keep. Otherwise the Redshift day-by-day check fails and every list is marked INCOMPLETE.
- Your own lists: HR people and transfers, a login map, the temporary and emergency accounts with end dates, a map from each role or group to a department, and your change tickets. 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).

The control

NIST SP 800-53 Rev. 5 is a catalog of security and privacy controls. This page quotes control AC-2, Account Management, from the NIST OSCAL catalog of SP 800-53 Rev. 5, release 5.2.0 (read 2026-11-01), word for word except AC-2 h.2, which is summarized around a quoted fragment. Each item in square brackets is a parameter, shown with the catalog's own label: a value the organization defines. NIST does not fill it in, and neither does this page. The enhancements, such as AC-2(2) and AC-2(3), apply only where your selected baseline includes them; this page does not know which baseline you have.

- **AC-2 f:** "Create, enable, modify, disable, and remove accounts in accordance with [policy, procedures, prerequisites, and criteria];"
- **AC-2 g:** "Monitor the use of accounts;"

- **AC-2 h.2:** account managers are notified within a time period you set “when users are terminated or transferred”.
- **AC-2 l:** “Align account management processes with personnel termination and transfer processes.”
- **AC-2(2), Automated Temporary and Emergency Account Management:** “Automatically [remove; disable] temporary and emergency accounts after [time period].”
- **AC-2(3), Disable Accounts:** “Disable accounts within [time period] when the accounts: (a) Have expired; (b) Are no longer associated with a user or individual; (c) Are in violation of organizational policy; or (d) Have been inactive for [time period].”
- **AC-2(4), Automated Audit Actions:** “Automatically audit account creation, modification, enabling, disabling, and removal actions.”

So the evidence has two halves. The platforms' own records show what happened: who is enabled, who signed in, what changed. Your policy says what should have happened: how many days of inactivity, how long a temporary account may outlive its end date, how quickly a transfer loses the old team's access. The script reads the first half and asks you for the second. It never picks a number for you.

Two neighbors are 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-in methods are in the SOC 2 user access guide. This page covers what AC-2 adds: inactivity, temporary accounts, transfers, and the trail of account actions.

What you will have at the end

A Python script, ac2_account_evidence.py, that reads CSV exports of each platform's own account records, with the platform's published column names, plus lists you keep. It runs 17 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: the PDF you are reading.

The four lists

List	AC-2 text it gives evidence on	What it flags
1. Inactive but enabled	AC-2(3)(d)	An enabled account with no sign-in for INACTIVE_DAYS, counted from its last sign-in, or from its creation if the records hold no sign-in.

List	AC-2 text it gives evidence on	What it flags
2. Temporary and emergency accounts past their end	AC-2(2), AC-2(3)(a)	A listed account that is still enabled after its end date plus TEMP_GRACE_DAYS (STILL ENABLED PAST END), was used after it (USED AFTER END), or was ended later (ENDED LATE).
3. Transfers that kept the old access	AC-2 h.2 and l	For each HR transfer, a role or group your map ties to the old department that the person still holds TRANSFER_DAYS after the transfer (KEPT OLD ACCESS).
4. Account actions in the period	AC-2(4)	Each successful create, modify, enable, disable and remove it reads from the platform records, with who did it where the record says. Also each change ticket with no recorded action (NOT RECORDED). The first is the full set of recorded actions, not a finding.

The records we ran it on are **generated**: a seeded stand-in, not a company's data and not a measurement of anything. We have not run the script on a real Snowflake, Databricks or Redshift account. The script produces evidence; it does not say a control is met or that anyone will accept it.

Step 1. Export each platform's records into one folder

The script reads these sixteen files from `EXPORT_DIR`. 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. Timestamps are read 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 sixteen files

File	Columns the script reads	Defined in
`snowflake_users.csv`	name, disabled, created_on, deleted_on, last_success_login, expires_at	Snowflake: ACCOUNT_USAGE.USERS (read 2026-11-01)
`snowflake_grants_to_users.csv`	created_on, deleted_on, role, grantee_name, granted_by	Snowflake: ACCOUNT_USAGE.GRANTS_TO_USERS (read 2026-11-01)
`snowflake_query_history.csv`	query_id, query_text, query_type, user_name, role_name, start_time, execution_status	Snowflake: ACCOUNT_USAGE.QUERY_HISTORY (read 2026-11-01)

File	Columns the script reads	Defined in
`databricks_users.csv`	id, username, active (the SCIM id, userName and active)	Databricks: list account users (read 2026-11-01)
`databricks_group_members.csv`	group_name, member_id (one row per member of each group: its displayName and the member's value)	Databricks: list account groups (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	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_sys_userlog.csv`	user_id, user_name, original_user_name, action, record_time	Amazon Redshift: SYS_USERLOG (read 2026-11-01)
`redshift_sys_connection_log.csv`	event, record_time, user_name	Amazon Redshift: SYS_CONNECTION_LOG (read 2026-11-01)
`hr_people.csv`	employee_email, department	You keep it: your HR file
`hr_transfers.csv`	employee_email, transfer_date, old_department, new_department	You keep it: your HR file
`login_map.csv`	platform, login, employee_email	You keep it: which login belongs to which person
`temp_accounts.csv`	platform, login, end_date, kind (temporary or emergency)	You keep it: your list with end dates

File	Columns the script reads	Defined in
`role_departments.csv`	platform, role_name, department	You keep it: which role or group belongs to which department
`change_tickets.csv`	ticket, platform, login, action (create, modify, enable, disable or remove), change_date	You keep it: your change tickets

Databricks keeps workspace-level events in each workspace's region. Run the audit export in a workspace in each region you use and put all the rows in one file; one region's export lacks the others' sign-ins, and the script cannot tell that any are missing.

Run `python3 ac2_account_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 each as a role that can read SNOWFLAKE.ACCOUNT_USAGE; set the session to UTC
first)
ALTER SESSION SET TIMEZONE = 'UTC';
SELECT name, disabled, created_on, deleted_on, last_success_login, expires_at FROM
SNOWFLAKE.ACCOUNT_USAGE.USERS;
SELECT created_on, deleted_on, role, grantee_name, granted_by FROM
SNOWFLAKE.ACCOUNT_USAGE.GRANTS_TO_USERS;
-- 120 days is a demo window: go back at least as far as your INACTIVE_DAYS
SELECT query_id, query_text, query_type, user_name, role_name, start_time, execution_status
FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY WHERE start_time >= DATEADD(day, -120, CURRENT_DATE);

-- Databricks: system.access.audit (maps and structs written as JSON text); go back at least
INACTIVE_DAYS.
-- Workspace-level rows are kept in the workspace's region: run this in a workspace in each region
you use
-- and put all the rows in one file.
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(),
120);
-- databricks_users.csv (id, username, active): GET /api/2.0/accounts/{account_id}/scim/v2/Users
--   -> id, userName, active, one row per user
-- databricks_group_members.csv (group_name, member_id): GET .../scim/v2/Groups, then GET
.../scim/v2/Groups/{id}
--   for each group (the list no longer returns members) -> 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_id, user_name, original_user_name, action, record_time FROM sys_userlog;
SELECT event, record_time, user_name FROM sys_connection_log;
SELECT user_name, user_id FROM svv_user_info;
SELECT user_name, role_name FROM svv_user_grants;
```

Step 2. Set your time periods

AC-2 leaves each time period to the organization, so the script has no default for `INACTIVE_DAYS`, `TEMP_GRACE_DAYS` or `TRANSFER_DAYS`. A list whose setting is blank does not run: it prints NOT RUN, and an input check fails. Here is the settings block at the top of the download. The demo fills in 90, 1 and 14 only so there is something to measure; those are not a recommendation, and nothing on this page says what your number should be.

```
# ---- CHANGE THESE FOR YOUR OWN DATA -------------------------------------------------------
PERIOD_START = "2026-09-01"          # first day of the review period (UTC, inclusive)
PERIOD_END = "2026-09-30"            # last day of the review period (UTC, inclusive)
AS_OF = "2026-10-05"                 # the day you took the exports (UTC); records must run to here
EXPORT_DIR = None                    # folder holding the export files named in FILES below
PLATFORMS_EXPORTED = ["snowflake", "databricks", "redshift"]   # a platform not listed is NOT
CHECKED
INACTIVE_DAYS = None                 # no default: days without a sign-in before an enabled account
is flagged
TEMP_GRACE_DAYS = None               # no default: days after a temporary account's end date it may
still be live
TRANSFER_DAYS = None                 # no default: days after a transfer by which old-department
access must go
KNOWN_GAP_DAYS = {}                  # days you can explain, e.g. {"redshift": ["2026-10-03"]}
# Databricks `accounts` audit actions (AWS and Azure audit log references): which ones are a sign-
in, and which
# ones change an account. The kinds apply to account-level rows (workspace_id 0); a workspace-level
row of any of
# these actions changes the account's place in one workspace, so it is listed as a modify of that
workspace.
DATABRICKS_SIGNIN_ACTIONS = ["aadBrowserLogin", "aadTokenLogin",
"accountInHouseOAuthClientAuthentication",
                             "accountLoginCodeAuthentication",
"accountlessToAccountLoginAuthentication", "certLogin",
                             "jwtLogin", "login", "mfaLogin", "multiFactorAuthenticationLogin",
"oidcBrowserLogin",
                             "oidcTokenAuthorization", "samlLogin", "tokenLogin",
"workspaceLoginCodeAuthentication"]
DATABRICKS_ACTION_KIND = {"add": "create", "activateUser": "enable", "deactivateUser": "disable",
"delete": "remove",
                          "updateUser": "modify", "setAdmin": "modify", "removeAdmin": "modify",
                          "setAccountAdmin": "modify", "removeAccountAdmin": "modify",
                          "addPrincipalToGroup": "modify", "addPrincipalsToGroup": "modify",
                          "removePrincipalFromGroup": "modify", "removePrincipalsFromGroup":
"modify"}
# documented actions whose names look like user management or a sign-in but are not an account
change we list
DATABRICKS_OTHER_ACTIONS = ["createGroup", "removeGroup", "updateGroup", "changePassword",
"resetPassword",
                            "usernameDomainDenied", "validateEmail", "deleteUser",
"changeServicePrincipalAcls",
                            "multiFactorAuthenticationUpdateUserAuthPolicy"]
REDSHIFT_SIGNIN_EVENTS = ["authenticated"]     # SYS_CONNECTION_LOG events that count as a sign-in
# -------------------------------------------------------------------------------------------
```

The two lists of Databricks action names are settings too. A documented account action that looks like user management but is not on either list fails an input check, so a new action name never slips by unread.

Step 3. How each list is decided

- **Enabled.** Snowflake: `DISABLED` is not true and `EXPIRES_AT` is empty or still in the future on the export day. Databricks: SCIM `active` is true. Redshift: every user in `SVV_USER_INFO`.
- **Last sign-in.** Snowflake: the latest of `LAST_SUCCESS_LOGIN` and any successful query by that user. Databricks: the latest successful (status 200) sign-in action in the `accounts` service, at account or workspace level. Redshift: the latest `authenticated` event in `SYS_CONNECTION_LOG`. An account with no sign-in in the records is counted from its creation (a `create` action, or `CREATED_ON` on Snowflake); with neither, it is flagged, because an input check has already shown the records cover every day back to `INACTIVE_DAYS` before the export. A Databricks user with no sign-in whose only record is a workspace-level `add` is flagged too, with that basis shown: the records look the same for a new user added from a workspace and for an old, idle user added to a new workspace, so the script does not treat it as the account's creation.
- **Temporary and emergency accounts.** Your list, plus any Snowflake user with an `EXPIRES_AT` that your list does not name. Status, in this order: USED AFTER END (any sign-in or successful query after the end date plus `TEMP_GRACE_DAYS`), STILL ENABLED PAST END, ENDED LATE (the first disable or remove action, the Snowflake `EXPIRES_AT` or the Snowflake `DELETED_ON` came after that date). A disable or remove counts only from the start of the login's current account (its first create after the last removal that comes before its latest create), and `DELETED_ON` only when no live user has that name, because Snowflake keeps dropped users in `USERS` and a name can be used again. An account that is gone from the platform with no recorded removal is NOT FOUND ON PLATFORM, and one that is disabled with no recorded date is ENDED, DATE NOT RECORDED. Both count as follow-up, never as clean.
- **Transfers.** For each transfer dated in the period, every login the login map gives that person on each platform, and every role or group that login holds now. A held role counts when your map ties it to the old department and not to the new one. Who holds a role comes from the current views (`GRANTS_TO_USERS` with no `DELETED_ON`, the group members export, `SVV_USER_GRANTS`), not from replaying logs. Redshift groups are not read: `SVV_USER_GRANTS` lists roles. A Databricks group member is matched to a user by the member's `value` and the user's `id`; a member that is not a user (a nested group or a service principal) is counted and not expanded.
- **Account actions.** Snowflake: the text of each successful `CREATE USER`, `ALTER USER` and `DROP USER` statement, and each `GRANT ROLE ... TO USER` and `REVOKE ROLE ... FROM`

`USER`, read with comments (`--`, `//` and `/* */`), double quotes and string literals removed. A successful statement that starts as one of these but that the script cannot read fails an input check; it is never skipped. `ALTER USER ... SET DISABLED = TRUE` is a disable and `= FALSE` an enable. Snowflake does not list USER among its `CREATE OR ALTER` objects today (Snowflake: CREATE OR ALTER (read 2026-11-01)); if that form appears, the script reads it as it reads `ALTER USER`; a create ticket may match it, and it can start a login's current account. “If the identifier is omitted, the statement modifies the active (i.e. logged in) user” (Snowflake: ALTER USER (read 2026-11-01)), so an `ALTER USER` that names no user is read as a change to the user who ran it. `ALTER USER ... ABORT ALL QUERIES` changes no account and is not listed. A name in double quotes that holds a space or another character an unquoted name cannot fails an input check rather than being read as a different user. The actor is the statement's `user_name`. Databricks: the `accounts` actions in `DATABRICKS_ACTION_KIND`, with the actor from `user_identity`. An account-level row (`workspace_id` 0) is read as its kind. A workspace-level row, such as a user added to or removed from one workspace, is listed as a modify with the workspace id, and is never taken as the account's creation or end. Redshift: each `SYS_USERLOG` row. AWS describes the `user_id` column as the identifier of the user who submitted the unload, which is not an obvious fit for this view, so the script prints `user_id N` in the actor column instead of a name; check it against your own cluster before you report it as the actor. A failed Snowflake statement is counted and not listed.

- **Tickets.** A ticket is matched to a recorded action on the same platform, login, kind of action and UTC day. A Databricks create ticket also matches a workspace-level `add` for a login with no account-level `add` in the records. A ticket with no match in the period is NOT RECORDED. Recorded actions with no ticket are counted, not flagged.

Step 4. Run it on the clean generated records

`python3 ac2_account_evidence.py --demo` writes generated exports (seed 232) in each platform's column names and reads them back. In this run every enabled account has signed in recently, every temporary account ended on time, the person who transferred holds only the new department's access, and every ticket matches a recorded action, so nothing needs follow-up. That says only that these generated records agree with these settings. The 17 input checks all passed (they are shown in step 6, where some fail on purpose).

```
17 input checks, 0 failed.

== 1. Inactive but enabled (AC-2(3)(d))
   period: state on 2026-10-05; filters: enabled accounts on snowflake, databricks, redshift;
INACTIVE_DAYS=90; run at 2026-11-01 17:48:48 UTC; rows: 17; sha256:
d048000b100d5da3038827b24c8797b537b486c5e0e9ef999591267466bc814c
   platform   | login                | basis_time          | basis        | days_idle | status
   databricks | ana@example.test     | 2026-10-02 11:31:00 | last sign-in | 3         | ok
   databricks | ben@example.test     | 2026-10-02 12:19:00 | last sign-in | 3         | ok
   databricks | chen@example.test    | 2026-10-02 13:34:00 | last sign-in | 3         | ok
```

```
   databricks | dara@example.test    | 2026-10-02 10:55:00 | last sign-in | 3         | ok
   databricks | sam@example.test     | 2026-10-02 09:10:00 | last sign-in | 3         | ok
   databricks | svc-etl@example.test | 2026-10-05 08:09:00 | last sign-in | 0         | ok
   redshift   | ana                  | 2026-10-02 15:36:00 | last sign-in | 3         | ok
   redshift   | chen                 | 2026-10-02 08:20:00 | last sign-in | 3         | ok
   redshift   | kim                  | 2026-10-02 10:55:00 | last sign-in | 3         | ok
   redshift   | sam                  | 2026-10-02 16:48:00 | last sign-in | 3         | ok
   redshift   | svc_bi               | 2026-10-05 08:01:00 | last sign-in | 0         | ok
   snowflake  | admin_sam            | 2026-10-02 16:18:00 | last sign-in | 3         | ok
   snowflake  | ana                  | 2026-10-02 12:20:00 | last sign-in | 3         | ok
   snowflake  | ben                  | 2026-10-02 10:43:00 | last sign-in | 3         | ok
   snowflake  | chen                 | 2026-10-02 11:39:00 | last sign-in | 3         | ok
   snowflake  | dara                 | 2026-10-02 14:39:00 | last sign-in | 3         | ok
   snowflake  | svc_load             | 2026-10-05 12:01:00 | last sign-in | 0         | ok

== 2. Temporary and emergency accounts past their end (AC-2(2), AC-2(3)(a))
   period: end dates up to 2026-09-30; state on 2026-10-05; filters: your temp_accounts list plus
Snowflake EXPIRES_AT; TEMP_GRACE_DAYS=1; run at 2026-11-01 17:48:48 UTC; rows: 4; sha256:
d2392a8479d654921d3876b2bbd2bab68dbd2ead72b309f8e7f5fdb9bad235a6
   platform   | login                       | kind      | end_date   | source    | enabled_now
| last_use            | ended_on   | status
   databricks | eli.contractor@example.test | temporary | 2026-09-15 | your list | false
| 2026-09-14 10:32:00 | 2026-09-15 | ended on time
   redshift   | eli_temp                    | temporary | 2026-09-15 | your list | not on platform
| 2026-09-14 09:23:00 | 2026-09-16 | ended on time
   redshift   | fire_call                   | emergency | 2026-09-12 | your list | not on platform
| 2026-09-11 08:40:00 | 2026-09-13 | ended on time
   snowflake  | eli_temp                    | temporary | 2026-09-20 | your list | false
| 2026-09-18 09:17:00 | 2026-09-20 | ended on time

== 3. Transfers that kept the old access (AC-2 h.2 and l)
   period: transfers dated 2026-09-01 to 2026-09-30; holdings on 2026-10-05; filters: roles and
groups your department map ties to the old department; TRANSFER_DAYS=14; run at 2026-11-01 17:48:48
UTC; rows: 3; sha256: 6c9f6c2990a277c2e6dac8eb84984269fa378df0caa06ec8959c900fd6e33a7d
   employee          | transfer_date | platform   | logins            | old_access_held | status
   dara@example.test | 2026-09-10    | databricks | dara@example.test |                 | clean
   dara@example.test | 2026-09-10    | redshift   |                   |                 | no login
on this platform
   dara@example.test | 2026-09-10    | snowflake  | dara              |                 | clean
   held roles and groups of transferred people that your department map does not list: 0

== 4a. Account actions the platforms recorded (AC-2(4))
   period: 2026-09-01 to 2026-09-30 (UTC); filters: every create, modify, enable, disable and
remove; successful only; run at 2026-11-01 17:48:48 UTC; rows: 15; sha256:
bd7ce2d7edd371e730232fce6accba0ac41f3425a8348ac8b7f4e622b92c5c78
   platform   | at_utc              | kind    | target                      | actor           |
source                              | ticket
   databricks | 2026-09-01 10:00:00 | create  | eli.contractor@example.test | sam@example.test |
audit add                           | no ticket
   databricks | 2026-09-01 10:00:00 | modify  | eli.contractor@example.test | sam@example.test |
audit add (workspace 1234567890123456) | no ticket
   databricks | 2026-09-10 10:00:00 | modify  | dara@example.test           | sam@example.test |
audit addPrincipalToGroup           | no ticket
   databricks | 2026-09-10 10:00:00 | modify  | dara@example.test           | sam@example.test |
audit removePrincipalFromGroup      | no ticket
   databricks | 2026-09-15 10:00:00 | disable | eli.contractor@example.test | sam@example.test |
audit deactivateUser                | T-103
   redshift   | 2026-09-01 10:00:00 | create  | eli_temp                    | user_id 100     |
```

```
sys_userlog create                     | no ticket
   redshift   | 2026-09-02 10:00:00 | remove  | jon                          | user_id 100      |
sys_userlog drop                       | T-102
   redshift   | 2026-09-03 11:00:00 | modify  | ana                          | user_id 100      |
sys_userlog alter                      | no ticket
   redshift   | 2026-09-08 10:00:00 | create  | kim                          | user_id 100      |
sys_userlog create                     | T-101
   redshift   | 2026-09-11 10:00:00 | create  | fire_call                    | user_id 100      |
sys_userlog create                     | no ticket
   redshift   | 2026-09-13 10:00:00 | remove  | fire_call                    | user_id 100      |
sys_userlog drop                       | T-106
   redshift   | 2026-09-16 10:00:00 | remove  | eli_temp                     | user_id 100      |
sys_userlog drop                       | T-107
   snowflake  | 2026-09-01 10:00:00 | create  | eli_temp                     | admin_sam        |
query q00361                           | T-104
   snowflake  | 2026-09-10 10:00:00 | modify  | dara                         | admin_sam        |
query q00362                           | T-105
   snowflake  | 2026-09-10 10:00:00 | modify  | dara                         | admin_sam        |
query q00365                           | T-105
   counts: databricks create 1; databricks disable 1; databricks modify 3; redshift create 3;
redshift modify 1; redshift remove 3; snowflake create 1; snowflake modify 2

== 4b. Change tickets with no recorded action (AC-2(4))
   period: 2026-09-01 to 2026-09-30 (UTC); filters: your change_tickets, matched on platform,
login, action and day; run at 2026-11-01 17:48:48 UTC; rows: 0; sha256:
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
   recorded actions with no ticket: 7 of 15 (a count to look at, not a finding)

Needs follow-up: inactive 0, temporary 0, transfers 0, not recorded 0; input checks failed: 0. The
script produces evidence; it does not say a control is met.
```

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

`python3 ac2_account_evidence.py --demo --plant` changes the generated records so that some things went wrong. An enabled Snowflake account has not signed in since 2026-05-14, a Databricks user was never deactivated and has no sign-in in the records, and a Redshift user has no sign-in either (list 1). The contractor's Snowflake and Databricks accounts were used after their end dates, and so was the emergency Redshift account. The Databricks one was removed from its workspace on 2026-09-15 but never deactivated, so the script lists that removal as a modify of the workspace, not as the account's end; another Redshift account was dropped on 2026-09-30, after an end date of 2026-09-15 (list 2). A person who moved from Finance to Data on 2026-09-10 still holds the Finance role on Snowflake and the Finance group on Databricks (list 3). Five tickets have no recorded action (list 4b). The script reports each one. The Snowflake `EXPIRES_AT` was left empty in this run, which is why that account shows as still enabled.

```
17 input checks, 0 failed.

== 1. Inactive but enabled (AC-2(3)(d))
   period: state on 2026-10-05; filters: enabled accounts on snowflake, databricks, redshift;
INACTIVE_DAYS=90; run at 2026-11-01 17:48:49 UTC; rows: 23; sha256:
```

```
4a82cd1d22d0806e352830659a443c97ed9a86ec15a7f6a9413e2bf699de7939
   platform   | login                       | basis_time          | basis                     |
days_idle | status
   databricks | ana@example.test            | 2026-10-02 14:31:00 | last sign-in              | 3
| ok
   databricks | ben@example.test            | 2026-10-02 13:05:00 | last sign-in              | 3
| ok
   databricks | chen@example.test           | 2026-10-02 10:16:00 | last sign-in              | 3
| ok
   databricks | dara@example.test           | 2026-10-02 09:16:00 | last sign-in              | 3
| ok
   databricks | eli.contractor@example.test | 2026-09-22 11:24:00 | last sign-in              | 13
| ok
   databricks | ivy@example.test            |                     | no sign-in in the records |
| INACTIVE BUT ENABLED
   databricks | sam@example.test            | 2026-10-02 11:45:00 | last sign-in              | 3
| ok
   databricks | svc-etl@example.test        | 2026-10-05 11:04:00 | last sign-in              | 0
| ok
   redshift   | ana                         | 2026-10-02 11:22:00 | last sign-in              | 3
| ok
   redshift   | chen                        | 2026-10-02 12:59:00 | last sign-in              | 3
| ok
   redshift   | fire_call                   | 2026-09-18 15:17:00 | last sign-in              | 17
| ok
   redshift   | jon                         |                     | no sign-in in the records |
| INACTIVE BUT ENABLED
   redshift   | kim                         | 2026-10-02 16:02:00 | last sign-in              | 3
| ok
   redshift   | sam                         | 2026-10-02 08:29:00 | last sign-in              | 3
| ok
   redshift   | svc_bi                      | 2026-10-05 11:50:00 | last sign-in              | 0
| ok
   snowflake  | admin_sam                   | 2026-10-02 16:18:00 | last sign-in              | 3
| ok
   snowflake  | ana                         | 2026-10-02 12:20:00 | last sign-in              | 3
| ok
   snowflake  | ben                         | 2026-10-02 10:43:00 | last sign-in              | 3
| ok
   snowflake  | chen                        | 2026-10-02 11:39:00 | last sign-in              | 3
| ok
   snowflake  | dara                        | 2026-10-02 14:39:00 | last sign-in              | 3
| ok
   snowflake  | eli_temp                    | 2026-09-28 10:49:00 | last sign-in              | 7
| ok
   snowflake  | hal                         | 2026-05-14 08:00:00 | last sign-in              | 144
| INACTIVE BUT ENABLED
   snowflake  | svc_load                    | 2026-10-05 10:53:00 | last sign-in              | 0
| ok

== 2. Temporary and emergency accounts past their end (AC-2(2), AC-2(3)(a))
   period: end dates up to 2026-09-30; state on 2026-10-05; filters: your temp_accounts list plus
Snowflake EXPIRES_AT; TEMP_GRACE_DAYS=1; run at 2026-11-01 17:48:49 UTC; rows: 4; sha256:
1f476bb91389c9f78007006a2f6cac4e40faa8b1f9d7ea3627913b0b16faf6c1
   platform   | login                       | kind      | end_date   | source    | enabled_now
| last_use            | ended_on   | status
   databricks | eli.contractor@example.test | temporary | 2026-09-15 | your list | true
| 2026-09-22 11:24:00 |            | USED AFTER END
```

```
   redshift   | eli_temp                    | temporary | 2026-09-15 | your list | not on platform
| 2026-09-14 14:46:00 | 2026-09-30 | ENDED LATE
   redshift   | fire_call                   | emergency | 2026-09-12 | your list | true
| 2026-09-18 15:17:00 |            | USED AFTER END
   snowflake  | eli_temp                    | temporary | 2026-09-20 | your list | true
| 2026-09-28 10:49:00 |            | USED AFTER END

== 3. Transfers that kept the old access (AC-2 h.2 and l)
   period: transfers dated 2026-09-01 to 2026-09-30; holdings on 2026-10-05; filters: roles and
groups your department map ties to the old department; TRANSFER_DAYS=14; run at 2026-11-01 17:48:49
UTC; rows: 3; sha256: 2b66125c4797dc17a23c83217957633511e74961d3bc16dd2ea5ec11280743da
   employee          | transfer_date | platform   | logins            | old_access_held | status
   dara@example.test | 2026-09-10    | databricks | dara@example.test | finance-readers | KEPT OLD
ACCESS
   dara@example.test | 2026-09-10    | redshift   |                   |                 | no login
on this platform
   dara@example.test | 2026-09-10    | snowflake  | dara              | fin_analyst     | KEPT OLD
ACCESS
   held roles and groups of transferred people that your department map does not list: 0

== 4a. Account actions the platforms recorded (AC-2(4))
   period: 2026-09-01 to 2026-09-30 (UTC); filters: every create, modify, enable, disable and
remove; successful only; run at 2026-11-01 17:48:49 UTC; rows: 11; sha256:
75a87feba37a65fe2561df234b08dcc1e178822e80f91766e870a7cd88ca7ced
   platform   | at_utc              | kind   | target                      | actor           |
source                                | ticket
   databricks | 2026-09-01 10:00:00 | create | eli.contractor@example.test | sam@example.test |
audit add                             | no ticket
   databricks | 2026-09-01 10:00:00 | modify | eli.contractor@example.test | sam@example.test |
audit add (workspace 1234567890123456)    | no ticket
   databricks | 2026-09-10 10:00:00 | modify | dara@example.test           | sam@example.test |
audit addPrincipalToGroup             | no ticket
   databricks | 2026-09-15 10:00:00 | modify | eli.contractor@example.test | sam@example.test |
audit delete (workspace 1234567890123456) | no ticket
   redshift   | 2026-09-01 10:00:00 | create | eli_temp                    | user_id 100     |
sys_userlog create                    | no ticket
   redshift   | 2026-09-03 11:00:00 | modify | ana                         | user_id 100     |
sys_userlog alter                     | no ticket
   redshift   | 2026-09-08 10:00:00 | create | kim                         | user_id 100     |
sys_userlog create                    | T-101
   redshift   | 2026-09-11 10:00:00 | create | fire_call                   | user_id 100     |
sys_userlog create                    | no ticket
   redshift   | 2026-09-30 10:00:00 | remove | eli_temp                    | user_id 100     |
sys_userlog drop                      | no ticket
   snowflake  | 2026-09-01 10:00:00 | create | eli_temp                    | admin_sam       |
query q00364                          | T-104
   snowflake  | 2026-09-10 10:00:00 | modify | dara                        | admin_sam       |
query q00365                          | T-105
   counts: databricks create 1; databricks modify 3; redshift create 3; redshift modify 1; redshift
remove 1; snowflake create 1; snowflake modify 1

== 4b. Change tickets with no recorded action (AC-2(4))
   period: 2026-09-01 to 2026-09-30 (UTC); filters: your change_tickets, matched on platform,
login, action and day; run at 2026-11-01 17:48:49 UTC; rows: 5; sha256:
469e8c35302158eb351f8716cb8ee1a5c844efb21069856cb79111a3cb264988
   ticket | platform   | login                       | action  | change_date | status
   T-102  | redshift   | jon                         | remove  | 2026-09-02  | NOT RECORDED
   T-103  | databricks | eli.contractor@example.test | disable | 2026-09-15  | NOT RECORDED
```

```
   T-106  | redshift   | fire_call                  | remove  | 2026-09-13  | NOT RECORDED
   T-107  | redshift   | eli_temp                   | remove  | 2026-09-16  | NOT RECORDED
   T-108  | redshift   | lee                        | create  | 2026-09-20  | NOT RECORDED
   recorded actions with no ticket: 8 of 11 (a count to look at, not a finding)

Needs follow-up: inactive 3, temporary 4, transfers 2, not recorded 5; input checks failed: 0. The
script produces evidence; it does not say a control is met.
```

Step 6. Run it with faults planted in the inputs

A list is only as good as its inputs, so the script checks them first and marks every list INCOMPLETE if one fails. `python3 ac2_account_evidence.py --demo --plant-input` plants one fault for 14 of the 17 checks, and each one fails and names what it found. Three checks do not fail in this run: the check on the dates and platforms in the settings (it would stop the script before any list), and the Snowflake and Databricks day-by-day record checks, which use the same query as the Redshift one that does fail.

```
Input checks
  [ok] dates and platforms in the settings
  [FAIL] every time period is set (AC-2 leaves each one to you): not set to a whole number of days:
TRANSFER_DAYS
  [FAIL] every file is there with every column the script reads: redshift_svv_user_grants.csv has
no column role_name
      read 9 rows into snowflake_users
      read 7 rows into snowflake_grants_to_users
      read 366 rows into snowflake_query_history
      read 8 rows into databricks_users
      read 5 rows into databricks_group_members
      read 357 rows into databricks_audit
      read 5 rows into redshift_svv_user_info
      read 0 rows into redshift_svv_user_grants
      read 8 rows into redshift_sys_userlog
      read 259 rows into redshift_sys_connection_log
      read 10 rows into hr_people
      read 2 rows into hr_transfers
      read 22 rows into login_map
      read 4 rows into temp_accounts
      read 10 rows into role_departments
      read 8 rows into change_tickets
  [FAIL] every date and time is written as YYYY-MM-DD [HH:MM:SS] (UTC): not readable, so left out:
snowflake_users.created_on: 1, temp_accounts.end_date: 1
  [FAIL] every value is one the script knows: snowflake_users.disabled not true or false: 1;
role_departments.platform not one of snowflake, databricks, redshift: 1; temp_accounts.kind not
temporary or emergency: 1; change_tickets.action not one of create, modify, enable, disable,
remove: 1
  [FAIL] every successful CREATE, ALTER and DROP USER statement and role grant to a user could be
read: 1 successful statement(s) look like one but name no user the script can read
  [FAIL] every Databricks action that looks like user management is one the script lists: unknown:
updateUserRole
  [FAIL] every Databricks action that looks like a sign-in is on the sign-in list: unknown:
passkeyLogin
  [FAIL] every Databricks sign-in and account action has a readable response status and
workspace_id: 1 row(s) whose response is not JSON with a status code
```

```
  [FAIL] every SYS_USERLOG action is alter, create, drop or rename: unknown: suspend
  [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] every role or group in the department map is held by someone: matches no held role:
redshift/data_rw, redshift/fin_ro, snowflake/ghost_role
  [FAIL] every transfer names someone in the HR people list: not in hr_people: zed@example.test
      Databricks group members that are not account users (groups or service principals; not
expanded): 0
  [ok] snowflake: records exist for every day from 2026-07-07 to 2026-10-05
  [ok] databricks: records exist for every day from 2026-07-07 to 2026-10-05
  [FAIL] redshift: records exist for every day from 2026-07-07 to 2026-10-05: 1 day(s) with none,
first 2026-09-14, last 2026-09-14

17 input checks, 14 failed.  ** INCOMPLETE: 14 input check(s) failed; read the lists with that in
mind **
```

Which planted fault trips which check

Check	Planted fault
every time period is set	TRANSFER_DAYS left unset (list 3 prints NOT RUN)
every file is there with every column the script reads	the Redshift SVV_USER_GRANTS export with role_name renamed to role
every date and time is written as YYYY-MM-DD [HH:MM:SS] (UTC)	a Snowflake created_on of 2026-13-45, and a temporary account end date written 9/20/26
every value is one the script knows	a DISABLED value of maybe, a platform called oracle, a kind called tmp, a ticket action called rename
every successful CREATE, ALTER and DROP USER statement and role grant to a user could be read	a successful statement that is just ALTER USER
every Databricks action that looks like user management is one the script lists	an action called updateUserRole
every Databricks action that looks like a sign-in is on the sign-in list	an action called passkeyLogin (a name we invented for the test)
every Databricks sign-in and account action has a readable response status and workspace_id	a login whose response is not JSON
every SYS_USERLOG action is alter, create, drop or rename	an action called suspend
no login appears twice in one platform's account list	the same Snowflake user twice
no login is mapped to two people	one Snowflake login mapped to two emails

Check	Planted fault
every role or group in the department map is held by someone	a role called GHOST_ROLE that nobody holds (the renamed Redshift column also leaves two Redshift roles unmatched)
every transfer names someone in the HR people list	a transfer for an email that is not in hr_people
redshift: records exist for every day	every Redshift connection record of 2026-09-14 removed

The code

The complete download, `ac2_account_evidence.py`, also at www.datameaning.com/guides/nist-800-53-ac-2-account-evidence-for-snowflake-databricks-redshift/ac2_account_evidence.py.

```
#!/usr/bin/env python3
"""NIST SP 800-53 AC-2 account evidence from Snowflake, Databricks and Redshift exports, in DuckDB.

    python3 ac2_account_evidence.py --demo                # clean generated records
    python3 ac2_account_evidence.py --demo --plant        # generated records with planted findings
    python3 ac2_account_evidence.py --demo --plant-input  # generated records with planted input
faults
    python3 ac2_account_evidence.py --export-sql          # print the export queries for each
platform
    python3 ac2_account_evidence.py                       # your exports: set the settings below
first
    python3 ac2_account_evidence.py --strict              # exit 1 unless nothing needs follow-up

AC-2 leaves every time period to the organization (each one is an organization-defined parameter),
so
INACTIVE_DAYS, TEMP_GRACE_DAYS and TRANSFER_DAYS have no default: a list whose setting is unset
does not run.
The demo uses 90, 1 and 14 so the demo has something to measure; they are not a recommendation.

The script reads CSV exports of each platform's own records, with the platform's published column
names, plus
lists you keep (HR people and transfers, a login map, temporary accounts, a role-to-department map,
change
tickets). It prints, each with its period, filters, run time, row count and a SHA-256 of its rows:
  1. inactive but enabled (AC-2(3)(d)): an enabled account with no sign-in for INACTIVE_DAYS,
counted from the
     last sign-in the records hold, or from creation if there is none
  2. temporary and emergency accounts past their end (AC-2(2), AC-2(3)(a)): accounts on your list,
or with a
     Snowflake EXPIRES_AT, still enabled, used, or ended later than the end date plus
TEMP_GRACE_DAYS
  3. transfers that kept the old access (AC-2 h.2 and l): roles or groups your map ties to the old
department
     that the person still holds TRANSFER_DAYS after the transfer
  4. account actions in the period (AC-2(4)): each successful create, modify, enable, disable and
remove the
     script reads from the platforms' records, and every change ticket with no recorded action (NOT
RECORDED).
```

```
     The first is the full set of recorded actions, not a finding.
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.

The demo records are GENERATED (seed 232): 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/query_history
  Databricks https://docs.databricks.com/aws/en/admin/system-tables/audit-logs
             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/list
  Redshift   https://docs.aws.amazon.com/redshift/latest/dg/SYS_USERLOG.html
             https://docs.aws.amazon.com/redshift/latest/dg/SYS_CONNECTION_LOG.html
             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
"""
import csv
import datetime as dt
import hashlib
import pathlib
import random
import re
import sys

import duckdb

# ---- CHANGE THESE FOR YOUR OWN DATA -------------------------------------------------------
PERIOD_START = "2026-09-01"          # first day of the review period (UTC, inclusive)
PERIOD_END = "2026-09-30"            # last day of the review period (UTC, inclusive)
AS_OF = "2026-10-05"                 # the day you took the exports (UTC); records must run to here
EXPORT_DIR = None                    # folder holding the export files named in FILES below
PLATFORMS_EXPORTED = ["snowflake", "databricks", "redshift"]   # a platform not listed is NOT
CHECKED
INACTIVE_DAYS = None                 # no default: days without a sign-in before an enabled account
is flagged
TEMP_GRACE_DAYS = None               # no default: days after a temporary account's end date it may
still be live
TRANSFER_DAYS = None                 # no default: days after a transfer by which old-department
access must go
KNOWN_GAP_DAYS = {}                  # days you can explain, e.g. {"redshift": ["2026-10-03"]}
# Databricks `accounts` audit actions (AWS and Azure audit log references): which ones are a sign-
in, and which
# ones change an account. The kinds apply to account-level rows (workspace_id 0); a workspace-level
row of any of
# these actions changes the account's place in one workspace, so it is listed as a modify of that
workspace.
DATABRICKS_SIGNIN_ACTIONS = ["aadBrowserLogin", "aadTokenLogin",
"accountInHouseOAuthClientAuthentication",
                             "accountLoginCodeAuthentication",
"accountlessToAccountLoginAuthentication", "certLogin",
                             "jwtLogin", "login", "mfaLogin", "multiFactorAuthenticationLogin",
"oidcBrowserLogin",
                             "oidcTokenAuthorization", "samlLogin", "tokenLogin",
"workspaceLoginCodeAuthentication"]
DATABRICKS_ACTION_KIND = {"add": "create", "activateUser": "enable", "deactivateUser": "disable",
"delete": "remove",
```

```
                           "updateUser": "modify", "setAdmin": "modify", "removeAdmin": "modify",
                           "setAccountAdmin": "modify", "removeAccountAdmin": "modify",
                           "addPrincipalToGroup": "modify", "addPrincipalsToGroup": "modify",
                           "removePrincipalFromGroup": "modify", "removePrincipalsFromGroup":
"modify"}
# documented actions whose names look like user management or a sign-in but are not an account
change we list
DATABRICKS_OTHER_ACTIONS = ["createGroup", "removeGroup", "updateGroup", "changePassword",
"resetPassword",
                            "usernameDomainDenied", "validateEmail", "deleteUser",
"changeServicePrincipalAcls",
                            "multiFactorAuthenticationUpdateUserAuthPolicy"]
REDSHIFT_SIGNIN_EVENTS = ["authenticated"]     # SYS_CONNECTION_LOG events that count as a sign-in
# ------------------------------------------------------------------------------------------

FILES = {
    "snowflake_users": ["name", "disabled", "created_on", "deleted_on", "last_success_login",
"expires_at"],
    "snowflake_grants_to_users": ["created_on", "deleted_on", "role", "grantee_name",
"granted_by"],
    "snowflake_query_history": ["query_id", "query_text", "query_type", "user_name", "role_name",
"start_time",
                                "execution_status"],
    "databricks_users": ["id", "username", "active"],
    "databricks_group_members": ["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"],
    "redshift_svv_user_grants": ["user_name", "role_name"],
    "redshift_sys_userlog": ["user_id", "user_name", "original_user_name", "action",
"record_time"],
    "redshift_sys_connection_log": ["event", "record_time", "user_name"],
    "hr_people": ["employee_email", "department"],
    "hr_transfers": ["employee_email", "transfer_date", "old_department", "new_department"],
    "login_map": ["platform", "login", "employee_email"],
    "temp_accounts": ["platform", "login", "end_date", "kind"],
    "role_departments": ["platform", "role_name", "department"],
    "change_tickets": ["ticket", "platform", "login", "action", "change_date"],
}
PLATFORM_FILES = {
    "snowflake": ["snowflake_users", "snowflake_grants_to_users", "snowflake_query_history"],
    "databricks": ["databricks_users", "databricks_group_members", "databricks_audit"],
    "redshift": ["redshift_svv_user_info", "redshift_svv_user_grants", "redshift_sys_userlog",
                 "redshift_sys_connection_log"],
}
PLATFORMS = ["snowflake", "databricks", "redshift"]
KINDS = ["create", "modify", "enable", "disable", "remove"]
OUT_DIR = pathlib.Path("evidence-out")
LIST_CSVS = ["1_inactive_but_enabled.csv", "2_temporary_accounts.csv", "3_transfers.csv",
"4a_account_actions.csv",
             "4b_tickets_not_recorded.csv"]
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}"
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)


# ---------------------------------------------------------------------------------------------
# Generated demo records (seed 232). Everything below this block to "ENGINE" builds files only.
# ---------------------------------------------------------------------------------------------
def write_csv(folder, name, rows):
    with open(folder / f"{name}.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(FILES[name])
        w.writerows(rows)


def generate(folder, plant, plant_input):
    rnd = random.Random(232)
    folder.mkdir(parents=True, exist_ok=True)
    d0, asof = dt.date(2026, 7, 1), dt.date(2026, 10, 5)
    days = [d0 + dt.timedelta(i) for i in range((asof - d0).days + 1)]
    D = lambda s: dt.date.fromisoformat(s)
    at = lambda d, h=9, m=0: f"{d} {h:02d}:{m:02d}:00"

    def schedule(first, last, always=False):
        out = []
        for d in days:
            if not (D(first) <= d <= D(last)):
                continue
            if always or (d.weekday() < 5 and (rnd.random() < 0.7 or d >= D(last) -
dt.timedelta(days=4))):
                out.append((d, 8 + rnd.randrange(9), rnd.randrange(60)))
        return out
```

```
    # ---- Snowflake
    sf_logins = {"ANA": ("2026-07-01", "2026-10-02"), "BEN": ("2026-07-01", "2026-10-02"),
                 "CHEN": ("2026-07-01", "2026-10-02"), "DARA": ("2026-07-01", "2026-10-02"),
                 "ADMIN_SAM": ("2026-07-01", "2026-10-02"), "ELI_TEMP": ("2026-09-01", "2026-09-
18"),
                 "SVC_LOAD": ("2026-07-01", "2026-10-05")}
    if plant:
        sf_logins["ELI_TEMP"] = ("2026-09-01", "2026-09-28")
    qh, qn, last_login = [], 0, {}
    for login, (a, b) in sf_logins.items():
        for d, h, m in schedule(a, b, always=(login == "SVC_LOAD")):
            qn += 1
            qh.append([f"q{qn:05d}", "SELECT 1", "SELECT", login, "ANALYST", at(d, h, m),
"success"])
            last_login[login] = at(d, h, m)

    def stmt(day_, text, who="ADMIN_SAM", status="success"):
        nonlocal qn
        qn += 1
        qh.append([f"q{qn:05d}", text, "UNKNOWN", who, "SECURITYADMIN", at(D(day_), 10), status])

    stmt("2026-09-01", "CREATE USER ELI_TEMP MUST_CHANGE_PASSWORD = TRUE COMMENT = 'contractor'")
    stmt("2026-09-10", "GRANT ROLE DATA_ENGINEER TO USER DARA")
    stmt("2026-09-12", "DROP USER GHOST", status="fail")
    if not plant:
        stmt("2026-08-03", "ALTER USER HAL SET DISABLED = TRUE")
        stmt("2026-09-10", '/* ticket T-105 */ REVOKE ROLE "FIN_ANALYST" FROM USER "DARA"')
    users = [[n, "false", "2026-01-15 09:00:00", "", last_login.get(n, ""), ""]
             for n in ["ANA", "BEN", "CHEN", "DARA", "ADMIN_SAM", "SVC_LOAD"]]
    users.append(["ELI_TEMP", "false", "2026-09-01 10:00:00", "", last_login["ELI_TEMP"],
                  "" if plant else "2026-09-20 00:00:00"])
    users.append(["HAL", "false" if plant else "true", "2025-11-03 09:00:00", "", "2026-05-14
08:00:00", ""])
    grants = [["2026-01-15 09:00:00", "", "FIN_ANALYST", "ANA", "SECURITYADMIN"],
              ["2026-01-15 09:00:00", "", "SALES_READER", "BEN", "SECURITYADMIN"],
              ["2026-01-15 09:00:00", "", "DATA_ENGINEER", "CHEN", "SECURITYADMIN"],
              ["2026-09-10 10:00:00", "", "DATA_ENGINEER", "DARA", "SECURITYADMIN"],
              ["2026-02-01 09:00:00", "" if plant else "2026-09-10 10:00:00", "FIN_ANALYST",
"DARA", "SECURITYADMIN"],
              ["2025-11-03 09:00:00", "", "FIN_ANALYST", "HAL", "SECURITYADMIN"],
              ["2026-01-15 09:00:00", "", "ACCOUNTADMIN", "ADMIN_SAM", "SECURITYADMIN"]]

    # ---- Databricks
    E = lambda n: f"{n}@example.test"
    db_logins = {"ana": ("2026-07-01", "2026-10-02"), "ben": ("2026-07-01", "2026-10-02"),
                 "chen": ("2026-07-01", "2026-10-02"), "dara": ("2026-07-01", "2026-10-02"),
                 "sam": ("2026-07-01", "2026-10-02"), "eli.contractor": ("2026-09-01", "2026-09-
14"),
                 "svc-etl": ("2026-07-01", "2026-10-05")}
    if plant:
        db_logins["eli.contractor"] = ("2026-09-01", "2026-09-22")
    audit, eid = [], 0

    WS = "1234567890123456"    # one workspace; account-level rows carry workspace_id 0

    def aud(d, h, who, action, params, code="200", ws="0"):
        nonlocal eid
```

```
        eid += 1
        import json
        audit.append([at(d, h, 0) if isinstance(d, dt.date) else d, str(d)[:10], ws,
                      json.dumps({"email": E(who)}), "accounts", action, json.dumps(params),
                      json.dumps({"status_code": int(code)}) if code.isdigit() else code])

    for login, (a, b) in db_logins.items():
        for d, h, m in schedule(a, b, always=(login == "svc-etl")):
            aud(f"{d} {h:02d}:{m:02d}:00", 0, login, "tokenLogin" if login == "svc-etl" else
"login",
                {"user": E(login), "authenticationMethod": "saml"}, ws=WS)
    adm = lambda d, action, ws="0", **p: aud(f"{d} 10:00:00", 0, "sam", action, {**p, "endpoint":
"accountsConsole"},
                                             ws=ws)
    adm("2026-09-01", "add", targetUserName=E("eli.contractor"))
    adm("2026-09-01", "add", ws=WS, targetUserName=E("eli.contractor"))    # added to the
workspace: not a creation
    adm("2026-09-10", "addPrincipalToGroup", targetUserName=E("dara"), targetGroupName="data-
engineers")
    if not plant:
        adm("2026-08-20", "deactivateUser", targetUserName=E("ivy"))
        adm("2026-09-10", "removePrincipalFromGroup", targetUserName=E("dara"),
targetGroupName="finance-readers")
        adm("2026-09-15", "deactivateUser", targetUserName=E("eli.contractor"))
    else:   # removed from the workspace only: the account itself stays active
        adm("2026-09-15", "delete", ws=WS, targetUserName=E("eli.contractor"))
    ids = {n: str(7001 + i) for i, n in enumerate(["ana", "ben", "chen", "dara", "sam", "svc-etl",
"eli.contractor",
                                                   "ivy"])}
    dbu = [[ids[n], E(n), "true"] for n in ["ana", "ben", "chen", "dara", "sam", "svc-etl"]]
    dbu += [[ids["eli.contractor"], E("eli.contractor"), "true" if plant else "false"],
            [ids["ivy"], E("ivy"), "true" if plant else "false"]]
    members = [["finance-readers", ids["ana"]], ["sales-readers", ids["ben"]], ["sales-readers",
ids["ivy"]],
               ["data-engineers", ids["chen"]], ["data-engineers", ids["dara"]]]
    if plant:
        members.append(["finance-readers", ids["dara"]])

    # ---- Redshift
    rs_logins = {"ana": ("2026-07-01", "2026-10-02"), "chen": ("2026-07-01", "2026-10-02"),
                 "sam": ("2026-07-01", "2026-10-02"), "kim": ("2026-09-09", "2026-10-02"),
                 "eli_temp": ("2026-09-01", "2026-09-14"), "fire_call": ("2026-09-11", "2026-09-
12"),
                 "svc_bi": ("2026-07-01", "2026-10-05")}
    if plant:
        rs_logins["fire_call"] = ("2026-09-11", "2026-09-20")
    conn = []
    for login, (a, b) in rs_logins.items():
        for d, h, m in schedule(a, b, always=(login == "svc_bi")):
            conn.append(["authenticated", f"{d} {h:02d}:{m:02d}:00", login])
    conn.append(["authentication failure", "2026-09-05 03:00:00", "unknown"])
    ulog = [[100, "eli_temp", "", "create", "2026-09-01 10:00:00"], [100, "ana", "", "alter",
"2026-09-03 11:00:00"],
            [100, "kim", "", "create", "2026-09-08 10:00:00"], [100, "fire_call", "", "create",
"2026-09-11 10:00:00"]]
    if plant:
        ulog.append([100, "eli_temp", "", "drop", "2026-09-30 10:00:00"])
    else:
```

```
        ulog += [[100, "jon", "", "drop", "2026-09-02 10:00:00"], [100, "fire_call", "", "drop",
"2026-09-13 10:00:00"],
                 [100, "eli_temp", "", "drop", "2026-09-16 10:00:00"]]
    rsu = [[n, i] for i, n in enumerate(["ana", "chen", "kim", "sam", "svc_bi"], 101)]
    rsg = [["ana", "fin_ro"], ["chen", "data_rw"], ["kim", "data_rw"]]
    if plant:
        rsu += [["jon", 110], ["fire_call", 111]]
        rsg.append(["jon", "data_rw"])

    # ---- lists you keep
    hr = [["ana@example.test", "Finance"], ["ben@example.test", "Sales"], ["chen@example.test",
"Data"],
          ["dara@example.test", "Data"], ["eli@example.test", "Contractors"], ["hal@example.test",
"Finance"],
          ["ivy@example.test", "Sales"], ["jon@example.test", "Data"], ["kim@example.test",
"Data"],
          ["sam@example.test", "IT"]]
    transfers = [["dara@example.test", "2026-09-10", "Finance", "Data"]]
    lm = [["snowflake", n, E(e)] for n, e in [("ANA", "ana"), ("BEN", "ben"), ("CHEN", "chen"),
("DARA", "dara"),
                                              ("ELI_TEMP", "eli"), ("HAL", "hal"), ("ADMIN_SAM",
"sam")]]
    lm += [["databricks", E(n), E(e)] for n, e in [("ana", "ana"), ("ben", "ben"), ("chen",
"chen"),
           ("dara", "dara"), ("eli.contractor", "eli"), ("ivy", "ivy"), ("sam", "sam")]]
    lm += [["redshift", n, E(e)] for n, e in [("ana", "ana"), ("chen", "chen"), ("kim", "kim"),
("jon", "jon"),
           ("eli_temp", "eli"), ("fire_call", "sam"), ("sam", "sam")]]
    temp = [["snowflake", "ELI_TEMP", "2026-09-20", "temporary"], ["databricks",
E("eli.contractor"), "2026-09-15",
            "temporary"], ["redshift", "eli_temp", "2026-09-15", "temporary"],
            ["redshift", "fire_call", "2026-09-12", "emergency"]]
    roles = [["snowflake", "FIN_ANALYST", "Finance"], ["snowflake", "DATA_ENGINEER", "Data"],
             ["snowflake", "SALES_READER", "Sales"], ["databricks", "finance-readers", "Finance"],
             ["databricks", "data-engineers", "Data"], ["databricks", "sales-readers", "Sales"],
             ["redshift", "fin_ro", "Finance"], ["redshift", "data_rw", "Data"]]
    tickets = [["T-101", "redshift", "kim", "create", "2026-09-08"], ["T-102", "redshift", "jon",
"remove", "2026-09-02"],
               ["T-103", "databricks", E("eli.contractor"), "disable", "2026-09-15"],
               ["T-104", "snowflake", "ELI_TEMP", "create", "2026-09-01"],
               ["T-105", "snowflake", "DARA", "modify", "2026-09-10"],
               ["T-106", "redshift", "fire_call", "remove", "2026-09-13"],
               ["T-107", "redshift", "eli_temp", "remove", "2026-09-16"]]
    if plant:
        tickets.append(["T-108", "redshift", "lee", "create", "2026-09-20"])

    if plant_input:
        users.append(["ANA", "false", "2026-01-15 09:00:00", "", last_login["ANA"], ""])
# duplicate login
        users[1][1] = "maybe"                                                                  #
BEN: DISABLED not true or false
        users[2][2] = "2026-13-45 09:00:00"                                                    #
CHEN: created_on not a timestamp
        grants_header_fault = True
        temp[0][2] = "9/20/26"                                                                 #
end date not YYYY-MM-DD
        temp[1][3] = "tmp"                                                                     #
kind not temporary or emergency
```

```
        roles.append(["snowflake", "GHOST_ROLE", "Finance"])                                    #
key matches no held role
        roles.append(["oracle", "DBA", "IT"])                                                    #
platform not known
        aud("2026-09-12 10:00:00", 0, "sam", "updateUserRole", {"targetUserName": E("ben")})    #
looks like user management
        aud("2026-09-12 11:00:00", 0, "ben", "passkeyLogin", {"user": E("ben")})                 #
looks like a sign-in
        aud("2026-09-12 12:00:00", 0, "ben", "login", {"user": E("ben")}, code="not json")       #
response unreadable
        conn[:] = [c for c in conn if not c[1].startswith("2026-09-14")]                         #
a day with no records
        qh.append(["q99999", "ALTER USER", "UNKNOWN", "ADMIN_SAM", "SECURITYADMIN", "2026-09-12
13:00:00", "success"])
        ulog.append([100, "kim", "", "suspend", "2026-09-12 14:00:00"])                          #
action not known
        transfers.append(["zed@example.test", "2026-09-12", "Finance", "Data"])                  #
not in hr_people
        lm.append(["snowflake", "ANA", "someone.else@example.test"])                             #
login mapped to two people
        tickets.append(["T-109", "snowflake", "ANA", "rename", "2026-09-12"])                    #
action not known
    else:
        grants_header_fault = False

    write_csv(folder, "snowflake_users", users)
    write_csv(folder, "snowflake_query_history", qh)
    write_csv(folder, "databricks_users", dbu)
    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_sys_userlog", ulog)
    write_csv(folder, "redshift_sys_connection_log", conn)
    write_csv(folder, "hr_people", hr)
    write_csv(folder, "hr_transfers", transfers)
    write_csv(folder, "login_map", lm)
    write_csv(folder, "temp_accounts", temp)
    write_csv(folder, "role_departments", roles)
    write_csv(folder, "change_tickets", tickets)
    write_csv(folder, "snowflake_grants_to_users", grants)
    if grants_header_fault:   # the export came with the column renamed: role_name -> role
        p = folder / "redshift_svv_user_grants.csv"
        p.write_text(p.read_text().replace("user_name,role_name", "user_name,role", 1))


EXPORT_SQL = """-- Snowflake (run each as a role that can read SNOWFLAKE.ACCOUNT_USAGE; set the
session to UTC first)
ALTER SESSION SET TIMEZONE = 'UTC';
SELECT name, disabled, created_on, deleted_on, last_success_login, expires_at FROM
SNOWFLAKE.ACCOUNT_USAGE.USERS;
SELECT created_on, deleted_on, role, grantee_name, granted_by FROM
SNOWFLAKE.ACCOUNT_USAGE.GRANTS_TO_USERS;
-- 120 days is a demo window: go back at least as far as your INACTIVE_DAYS
SELECT query_id, query_text, query_type, user_name, role_name, start_time, execution_status
FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY WHERE start_time >= DATEADD(day, -120, CURRENT_DATE);

-- Databricks: system.access.audit (maps and structs written as JSON text); go back at least
```

```
INACTIVE_DAYS.
-- Workspace-level rows are kept in the workspace's region: run this in a workspace in each region
you use
-- and put all the rows in one file.
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(),
120);
-- databricks_users.csv (id, username, active): GET /api/2.0/accounts/{account_id}/scim/v2/Users
--   -> id, userName, active, one row per user
-- databricks_group_members.csv (group_name, member_id): GET .../scim/v2/Groups, then GET
.../scim/v2/Groups/{id}
--   for each group (the list no longer returns members) -> 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_id, user_name, original_user_name, action, record_time FROM sys_userlog;
SELECT event, record_time, user_name FROM sys_connection_log;
SELECT user_name, user_id FROM svv_user_info;
SELECT user_name, role_name FROM svv_user_grants;
"""


# ---------------------------------------------------------------------------------------------
# ENGINE
# ---------------------------------------------------------------------------------------------
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()


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, INACTIVE_DAYS, TEMP_GRACE_DAYS, TRANSFER_DAYS
    global 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)
        INACTIVE_DAYS, TEMP_GRACE_DAYS, TRANSFER_DAYS = 90, 1, 14
        if plant_input:
            TRANSFER_DAYS = None   # an unset setting: list 3 must not run
        print(f"Generated records written to {GEN_DIR}/ (seed 232; not a company's data)."
              f" Demo settings: INACTIVE_DAYS=90, TEMP_GRACE_DAYS=1, TRANSFER_DAYS=14.")
    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)
    OUT_DIR.mkdir(parents=True, exist_ok=True)
    for name in LIST_CSVS:    # evidence-out/ holds this run's lists only, never an earlier run's
        (OUT_DIR / name).unlink(missing_ok=True)
    print(f"\nRun at {RUN_AT}. Period {PERIOD_START} to {PERIOD_END} (UTC); exports taken {AS_OF};
"
          f"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("\nInput checks")

    # --- settings
    problems = []
    for nm, v in [("PERIOD_START", PERIOD_START), ("PERIOD_END", PERIOD_END), ("AS_OF", AS_OF)]:
        if not (isinstance(v, str) and re.fullmatch(DATE_RE, v)):
            problems.append(f"{nm} is not written as YYYY-MM-DD")
    if not problems:
        try:
            ps, pe, ao = (dt.date.fromisoformat(x) for x in (PERIOD_START, PERIOD_END, AS_OF))
            if not ps <= pe <= ao:
                problems.append("need PERIOD_START <= PERIOD_END <= AS_OF")
        except ValueError:
            problems.append("a date is not a real calendar date")
    for p in PLATFORMS_EXPORTED:
        if p not in PLATFORMS:
            problems.append(f"PLATFORMS_EXPORTED has '{p}', not one of {PLATFORMS}")
    rep.check("dates and platforms in the settings", not problems, "; ".join(problems))
    if problems:
        return 2
    unset = [n for n, v in [("INACTIVE_DAYS", INACTIVE_DAYS), ("TEMP_GRACE_DAYS", TEMP_GRACE_DAYS),
                            ("TRANSFER_DAYS", TRANSFER_DAYS)] if not (isinstance(v, int) and v >=
0)]
    rep.check("every time period is set (AC-2 leaves each one to you)", not unset,
              ("not set to a whole number of days: " + ", ".join(unset)) if unset else "")

    # --- 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 = []
        if (plat is None or plat in PLATFORMS_EXPORTED) and path.is_file():
            have = [d[0] for d in con.execute("SELECT * FROM read_csv(?, header=true,
all_varchar=true) LIMIT 0",
                                              [str(path)]).description]
        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 missing:
            bad_files.append(f"{name}.csv has no column {', '.join(missing)}")
        if have and not missing:
            sel = ", ".join(f'"{h}" AS "{h.strip().lower()}"' for h in have)
            con.execute(f"CREATE TABLE t_{name} AS SELECT {sel} FROM read_csv(?, header=true,
all_varchar=true)",
                        [str(path)])
        else:
            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 = [("snowflake_query_history", "start_time"), ("databricks_audit", "event_time"),
               ("redshift_sys_userlog", "record_time"), ("redshift_sys_connection_log",
"record_time")]
    opt_ts = [("snowflake_users", c) for c in ("created_on", "deleted_on", "last_success_login",
"expires_at")] + \
             [("snowflake_grants_to_users", "created_on"), ("snowflake_grants_to_users",
"deleted_on")]
    must_day = [("hr_transfers", "transfer_date"), ("temp_accounts", "end_date"),
("change_tickets", "change_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}")
    for tbl, col in must_day:
        n = con.execute(f"SELECT count(*) FROM t_{tbl} WHERE ({day(col)}) IS 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 = []
    for tbl, col in [("snowflake_users", "disabled"), ("databricks_users", "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}")
    for tbl, col in [("login_map", "platform"), ("temp_accounts", "platform"), ("role_departments",
"platform"),
                     ("change_tickets", "platform")]:
        n = con.execute(f"SELECT count(*) FROM t_{tbl} WHERE lower(trim({col})) NOT IN
({quote_list(PLATFORMS)}) "
                        f"OR {col} IS NULL").fetchone()[0]
        if n:
```

```
            odd.append(f"{tbl}.{col} not one of {', '.join(PLATFORMS)}: {n}")
    n = con.execute(f"SELECT count(*) FROM t_temp_accounts WHERE lower(trim(kind)) NOT IN
('temporary','emergency') "
                    f"OR kind IS NULL").fetchone()[0]
    if n:
        odd.append(f"temp_accounts.kind not temporary or emergency: {n}")
    n = con.execute(f"SELECT count(*) FROM t_change_tickets WHERE lower(trim(action)) NOT IN
({quote_list(KINDS)}) "
                    f"OR action IS NULL").fetchone()[0]
    if n:
        odd.append(f"change_tickets.action not one of {', '.join(KINDS)}: {n}")
    for tbl, col in [("temp_accounts", "login"), ("hr_transfers", "employee_email"),
("change_tickets", "login"),
                     ("databricks_users", "id"), ("databricks_group_members", "member_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))

    # --- Snowflake statements: read CREATE, ALTER and DROP USER (and role grants to a user) from
the text
    def clean_sql(text):
        """One pass, left to right, so a quote inside a comment or a comment marker inside a
literal is read as
        what it is: literals and comments (--, // and /* */) become a space. A double-quoted name
keeps its text, unless it holds a
        character an unquoted name cannot (a space, say); then it becomes a mark no pattern reads,
and the
        statement fails the input check below instead of being read as a different user."""
        def one(m):
            if m.group(1) is None:
                return " "
            name = m.group(1).replace('""', '"')
            return name if re.fullmatch(r"[A-Za-z0-9_.@$\-]+", name) else "\x01"
        t = re.sub(r"'(?:[^']|'')*'|--[^\n]*|//[^\n]*|/\*.*?\*/|\"((?:[^\"]|\"\")*)\"", one, text
or "", flags=re.S)
        return re.sub(r"\s+", " ", t).strip().rstrip("; ")   # a statement may end with a semicolon
    NAMEP = r"([A-Za-z0-9_.@$\-]+)"
    pat = {"create": re.compile(r"^CREATE (OR REPLACE |OR ALTER )?USER (?:IF NOT EXISTS )?" + NAMEP
+ r"(?: (.*))?$", re.I),
           "alter": re.compile(r"^ALTER USER (?:IF EXISTS )?" + NAMEP + r"(?: (.*))?$", re.I),
           "drop": re.compile(r"^DROP USER (?:IF EXISTS )?" + NAMEP, re.I),
           "grant": re.compile(r"^GRANT ROLE \S+ TO USER " + NAMEP, re.I),      # the role name is
not kept
           "revoke": re.compile(r"^REVOKE ROLE \S+ FROM USER " + NAMEP, re.I)}
    # a statement that starts as a user change (after comments are stripped) but that the patterns
do not read is
    # counted, never skipped; "CREATE USER" inside a privilege name or a procedure body is not a
user change
    looks = re.compile(r"^((CREATE|ALTER|DROP)( OR (REPLACE|ALTER))? USER\b|(GRANT|REVOKE) ROLE\b.*
(TO|FROM) USER\b)", re.I)
    # an unreadable quoted name where the user's name goes; a quoted option value later in the
statement is fine
    name_unread = re.compile(r"^((CREATE( OR (REPLACE|ALTER))?|ALTER|DROP) USER (IF (NOT )?EXISTS
)?\x01"
                             r"|(GRANT|REVOKE) ROLE \S+ (TO|FROM) USER \x01)", re.I)
    # the words an ALTER USER can start with when it names no user (Snowflake's ALTER USER syntax)
    ALTER_NO_NAME = ("SET", "UNSET", "RESET", "RENAME", "ADD", "REMOVE", "MODIFY", "ROTATE",
```

```
"ABORT", "ENROLL")
    con.execute("CREATE TEMP TABLE sf_actions (platform VARCHAR, t TIMESTAMP, actor VARCHAR, target
VARCHAR, "
                "kind VARCHAR, source VARCHAR)")
    unread, failed_user_stmts, rows = 0, 0, []
    src = con.execute(f"SELECT lower(trim(user_name)), ({ts('start_time')}),
lower(trim(execution_status)), "
                      f"query_text, query_id FROM t_snowflake_query_history").fetchall()
    for who, t, status, text, qid in src:
        s = clean_sql(text)
        kind, target = None, None
        if name_unread.match(s):   # a quoted name no pattern can read: never let IF be taken as
the name
            unread += status == "success"
            continue
        if m := pat["create"].match(s):
            kind, target = "create", m.group(2)
            if m.group(1) and m.group(1).upper().startswith("OR ALTER"):   # creates the user or
changes it
                d = re.search(r"\bDISABLED\s*=\s*(TRUE|FALSE)\b", m.group(3) or "", re.I)
                kind = ("disable" if d.group(1).upper() == "TRUE" else "enable") if d else "modify"
        elif m := pat["drop"].match(s):
            kind, target = "remove", m.group(1)
        elif m := pat["alter"].match(s):
            target, rest = m.group(1), m.group(2) or ""
            if target.upper() in ALTER_NO_NAME:   # no name: Snowflake alters the logged-in user
                target, rest = who, (target + " " + rest).strip()
            if re.match(r"ABORT ALL QUERIES\b", rest, re.I):   # stops queries; the account is not
changed
                continue
            d = re.search(r"\bDISABLED\s*=\s*(TRUE|FALSE)\b", rest, re.I)
            kind = ("disable" if d.group(1).upper() == "TRUE" else "enable") if d else "modify"
        elif m := (pat["grant"].match(s) or pat["revoke"].match(s)):
            kind, target = "modify", m.group(1)
        elif looks.match(s):
            if status == "success":
                unread += 1
            continue
        else:
            continue
        if status != "success":
            failed_user_stmts += 1
            continue
        rows.append(("snowflake", t, who, target.lower(), kind,
                     "query " + qid + (" (create or alter)" if s.upper().startswith("CREATE OR
ALTER") else "")))
    con.executemany("INSERT INTO sf_actions VALUES (?,?,?,?,?,?)", rows)
    rep.check("every successful CREATE, ALTER and DROP USER statement and role grant to a user
could be read",
              unread == 0,
              f"{unread} successful statement(s) look like one but name no user the script can
read" if unread
              else f"{failed_user_stmts} failed user statement(s) not counted" if failed_user_stmts
else "")

    # --- Databricks audit: actions the script does not know
    known = DATABRICKS_SIGNIN_ACTIONS + list(DATABRICKS_ACTION_KIND) + DATABRICKS_OTHER_ACTIONS
    unk_mgmt = con.execute(f"SELECT DISTINCT trim(action_name) FROM t_databricks_audit WHERE
```

```
lower(trim(service_name))="
                           f"'accounts' AND trim(action_name) NOT IN ({quote_list(known)}) AND "
                           f"regexp_matches(action_name, '(?i)(user|principal|admin|group)') "
                           f"AND NOT regexp_matches(action_name, '(?i)(login|authenticat)') ORDER
BY 1").fetchall()
    unk_sign = con.execute(f"SELECT DISTINCT trim(action_name) FROM t_databricks_audit WHERE
lower(trim(service_name))="
                           f"'accounts' AND trim(action_name) NOT IN ({quote_list(known)}) AND "
                           f"regexp_matches(action_name, '(?i)(login|authenticat)') ORDER BY
1").fetchall()
    rep.check("every Databricks action that looks like user management is one the script lists",
not unk_mgmt,
              "unknown: " + ", ".join(r[0] for r in unk_mgmt) if unk_mgmt else "")
    rep.check("every Databricks action that looks like a sign-in is on the sign-in list", not
unk_sign,
              "unknown: " + ", ".join(r[0] for r in unk_sign) if unk_sign else "")
    inscope = quote_list(DATABRICKS_SIGNIN_ACTIONS + list(DATABRICKS_ACTION_KIND))
    nstat = con.execute(f"SELECT count(*) FROM t_databricks_audit WHERE
lower(trim(service_name))='accounts' AND "
                        f"trim(action_name) IN ({inscope}) 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 ({inscope}) AND {blank('workspace_id')} IS
NULL").fetchone()[0]
    rep.check("every Databricks sign-in and account action 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))
    ulog_unknown = con.execute("SELECT DISTINCT lower(trim(action)) FROM t_redshift_sys_userlog
WHERE "
                               "lower(trim(action)) NOT IN ('alter','create','drop','rename') OR
action IS NULL "
                               "ORDER BY 1").fetchall()
    rep.check("every SYS_USERLOG action is alter, create, drop or rename", not ulog_unknown,
              "unknown: " + ", ".join(str(r[0]) for r in ulog_unknown) if ulog_unknown else "")

    # --- normalize: accounts, sign-ins, holdings, actions
    asof_end = f"TIMESTAMP '{AS_OF} 00:00:00' + INTERVAL 1 DAY"
    con.execute(f"""CREATE TEMP TABLE signins AS
        SELECT 'snowflake' AS platform, lower(trim(user_name)) AS login, ({ts('start_time')}) AS t
          FROM t_snowflake_query_history WHERE lower(trim(execution_status)) = 'success' AND
{blank('user_name')} IS NOT NULL
        UNION ALL SELECT 'snowflake', lower(trim(name)), ({ts('last_success_login')}) FROM
t_snowflake_users
          WHERE {blank('last_success_login')} IS NOT NULL
        UNION ALL SELECT 'databricks', lower(trim(coalesce({blank(jv('request_params', 'user'))},
                 {jv('user_identity', 'email')}))), ({ts('event_time')}) FROM t_databricks_audit
          WHERE lower(trim(service_name)) = 'accounts' AND trim(action_name) IN
({quote_list(DATABRICKS_SIGNIN_ACTIONS)})
            AND {status_ok()}
        UNION ALL SELECT 'redshift', lower(trim(user_name)), ({ts('record_time')}) FROM
t_redshift_sys_connection_log
          WHERE trim(event) IN ({quote_list(REDSHIFT_SIGNIN_EVENTS)})""")
    con.execute("CREATE TEMP TABLE db_kind (action VARCHAR, kind VARCHAR)")
```

```
    con.executemany("INSERT INTO db_kind VALUES (?,?)", list(DATABRICKS_ACTION_KIND.items()))
    con.execute(f"""CREATE TEMP TABLE actions AS
        SELECT * FROM sf_actions
        UNION ALL SELECT 'databricks', ({ts('a.event_time')}), lower(trim({jv('a.user_identity',
'email')})),
                lower(trim({jv('a.request_params', 'targetUserName')})),
                CASE WHEN trim(a.workspace_id) = '0' THEN k.kind ELSE 'modify' END,
                'audit ' || trim(a.action_name) || CASE WHEN trim(a.workspace_id) = '0' THEN ''
                ELSE ' (workspace ' || trim(a.workspace_id) || ')' END
          FROM t_databricks_audit a JOIN db_kind k ON k.action = trim(a.action_name)
          WHERE lower(trim(a.service_name)) = 'accounts' AND {status_ok('a.response')} AND
{blank('a.workspace_id')} IS NOT NULL
        UNION ALL SELECT 'redshift', ({ts('record_time')}), 'user_id ' || trim(user_id),
lower(trim(user_name)),
                CASE lower(trim(action)) WHEN 'create' THEN 'create' WHEN 'drop' THEN 'remove'
ELSE 'modify' END,
                'sys_userlog ' || lower(trim(action)) || CASE WHEN {blank('original_user_name')}
IS NOT NULL
                THEN ' (was ' || lower(trim(original_user_name)) || ')' ELSE '' END
          FROM t_redshift_sys_userlog WHERE lower(trim(action)) IN
('alter','create','drop','rename')""")
    # A user added from a workspace may have no account-level add in the records. That workspace
add stays a modify
    # (it is also what adding an existing, idle user to a new workspace looks like), but a create
ticket may match it.
    con.execute("ALTER TABLE actions ADD COLUMN ticket_kind VARCHAR")
    con.execute("""UPDATE actions SET ticket_kind = CASE WHEN platform = 'databricks' AND source
LIKE 'audit add (workspace %'
        AND NOT EXISTS (SELECT 1 FROM actions y WHERE y.platform = 'databricks' AND y.target =
actions.target
                        AND y.kind = 'create') THEN 'create'
        WHEN platform = 'snowflake' AND source LIKE '% (create or alter)' THEN 'create' ELSE kind
END""")
    # Each login's current incarnation starts at the first create after the last remove that
precedes its latest
    # create: a later CREATE USER IF NOT EXISTS (a no-op on an existing user) never moves it, and
an earlier
    # holder of a reused name never counts.
    con.execute("""CREATE TEMP TABLE since AS
        WITH lc AS (SELECT platform, target, max(t) AS last_create FROM actions
                    WHERE kind = 'create' OR (platform = 'snowflake' AND ticket_kind = 'create')
GROUP BY 1, 2),
        lr AS (SELECT lc.platform, lc.target, lc.last_create, (SELECT max(r.t) FROM actions r WHERE
r.platform = lc.platform
               AND r.target = lc.target AND r.kind = 'remove' AND r.t < lc.last_create) AS
prior_remove FROM lc)
        SELECT lr.platform, lr.target, (SELECT min(c.t) FROM actions c WHERE c.platform =
lr.platform AND c.target = lr.target
               AND (c.kind = 'create' OR (c.platform = 'snowflake' AND c.ticket_kind = 'create'))
               AND (lr.prior_remove IS NULL OR c.t > lr.prior_remove)) AS since FROM lr""")
    con.execute(f"""CREATE TEMP TABLE accounts AS
        SELECT 'snowflake' AS platform, lower(trim(name)) AS login,
               (lower(trim(disabled)) <> 'true' AND (({ts('expires_at')}) IS NULL OR
({ts('expires_at')}) > {asof_end})) AS enabled,
               ({ts('created_on')}) AS created_on, ({ts('expires_at')}) AS expires_at
          FROM t_snowflake_users WHERE {blank('deleted_on')} IS NULL
        UNION ALL SELECT 'databricks', lower(trim(username)), lower(trim(active)) = 'true', NULL,
NULL
```

```
          FROM t_databricks_users
        UNION ALL SELECT 'redshift', lower(trim(user_name)), TRUE, NULL, NULL FROM
t_redshift_svv_user_info""")
    con.execute("""CREATE TEMP TABLE accounts2 AS
        SELECT a.platform, a.login, a.enabled, a.expires_at,
               coalesce(a.created_on, (SELECT n.since FROM since n WHERE n.platform = a.platform
                        AND n.target = a.login)) AS created_on,
               (SELECT max(s.t) FROM signins s WHERE s.platform = a.platform AND s.login = a.login)
AS last_signin,
               (SELECT min(x.t) FROM actions x WHERE x.platform = 'databricks' AND a.platform =
'databricks'
                        AND x.target = a.login AND x.kind = 'modify' AND x.ticket_kind = 'create')
AS ws_added
        FROM accounts a""")
    con.execute(f"""CREATE TEMP TABLE holdings AS
        SELECT 'snowflake' AS platform, lower(trim(grantee_name)) AS login, lower(trim(role)) AS
role_name
          FROM t_snowflake_grants_to_users WHERE {blank('deleted_on')} IS NULL
        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 'redshift', lower(trim(user_name)), lower(trim(role_name)) FROM
t_redshift_svv_user_grants""")
    con.execute(f"""CREATE TEMP TABLE ended_sf AS SELECT lower(trim(name)) AS login,
max(({ts('deleted_on')})::DATE) AS d
        FROM t_snowflake_users WHERE {blank('deleted_on')} IS NOT NULL GROUP BY 1""")

    # --- 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)), count(DISTINCT
lower(trim(employee_email))) "
                            "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))
    nokey = con.execute(f"""SELECT lower(trim(platform)), lower(trim(role_name)) FROM
t_role_departments r
        WHERE lower(trim(platform)) IN ({quote_list(PLATFORMS_EXPORTED)}) AND NOT EXISTS (SELECT 1
FROM holdings h
          WHERE h.platform = lower(trim(r.platform)) AND h.role_name = lower(trim(r.role_name)))
ORDER BY 1, 2""").fetchall()
    rep.check("every role or group in the department map is held by someone", not nokey,
              "matches no held role: " + ", ".join(f"{p}/{r}" for p, r in nokey) if nokey else "")
    nohr = con.execute("SELECT DISTINCT coalesce(lower(trim(t.employee_email)), '(blank)') FROM
t_hr_transfers t WHERE "
                       "NOT EXISTS (SELECT 1 FROM t_hr_people p WHERE lower(trim(p.employee_email))
= "
                       "lower(trim(t.employee_email))) ORDER BY 1").fetchall()
    rep.check("every transfer names someone in the HR people list", not nohr,
              "not in hr_people: " + ", ".join(r[0] for r in nohr) if nohr 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))").fetchone()
[0]
        print(f"      Databricks group members that are not account users (groups or service
principals; "
              f"not expanded): {nm}")

    # --- completeness: a record for every day the lists depend on, per platform
    span_start = PERIOD_START
    if isinstance(INACTIVE_DAYS, int) and INACTIVE_DAYS >= 0:
        span_start = min(PERIOD_START, str(dt.date.fromisoformat(AS_OF) -
dt.timedelta(days=INACTIVE_DAYS)))
    day_src = {"snowflake": f"SELECT ({ts('start_time')})::DATE AS d FROM
t_snowflake_query_history",
               "databricks": f"SELECT ({ts('event_time')})::DATE AS d FROM t_databricks_audit "
                             f"WHERE lower(trim(service_name)) = 'accounts'",
               "redshift": f"SELECT ({ts('record_time')})::DATE AS d FROM
t_redshift_sys_connection_log"}
    for p in PLATFORMS_EXPORTED:
        gaps = [str(r[0]) for r in con.execute(f"""SELECT g.d::DATE FROM generate_series(DATE
'{span_start}',
            DATE '{AS_OF}', INTERVAL 1 DAY) g(d) WHERE g.d::DATE NOT IN (SELECT d FROM
({day_src[p]}) WHERE d IS NOT NULL)
            ORDER BY 1""").fetchall()]
        gaps = [g for g in gaps if g not in KNOWN_GAP_DAYS.get(p, [])]
        rep.check(f"{p}: records exist for every day from {span_start} to {AS_OF}", not gaps,
                  f"{len(gaps)} day(s) with none, first {gaps[0]}, last {gaps[-1]}" if gaps else
"")

    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):
        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 ""))
        if rows:
            w = [max(len(str(c)), *(len(str(r[i])) for r in rows)) for i, c in enumerate(cols)]
            print("   " + " | ".join(str(c).ljust(w[i]) for i, c in enumerate(cols)))
            for r in rows:
                print("   " + " | ".join(("" if v is None else str(v)).ljust(w[i]) for i, v in
enumerate(r)))
        with open(OUT_DIR / csv_name, "w", newline="") 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)
    need = lambda n: f"NOT RUN: {n} is not set"

    def not_run(title, setting, csv_name):
        print(f"\n== {title}\n   " + need(setting))
        with open(OUT_DIR / csv_name, "w", newline="") as f:
            csv.writer(f).writerows([["status"], [need(setting) + f"; run at {RUN_AT}"]])

    # --- 1. inactive but enabled
    if isinstance(INACTIVE_DAYS, int) and INACTIVE_DAYS >= 0:
```

```
        rows = con.execute(f"""SELECT platform, login,
            coalesce(last_signin, created_on, ws_added)::VARCHAR AS basis_time,
            CASE WHEN last_signin IS NOT NULL THEN 'last sign-in' WHEN created_on IS NOT NULL THEN
'created'
                 WHEN ws_added IS NOT NULL THEN 'no sign-in; added to a workspace, no account-level
add'
                 ELSE 'no sign-in in the records' END AS basis,
            date_diff('day', coalesce(last_signin, created_on, ws_added)::DATE, DATE '{AS_OF}') AS
days_idle,
            CASE WHEN coalesce(last_signin, created_on) IS NULL
                   OR date_diff('day', coalesce(last_signin, created_on)::DATE, DATE '{AS_OF}') >=
{INACTIVE_DAYS}
                 THEN 'INACTIVE BUT ENABLED' ELSE 'ok' END AS status
            FROM accounts2 WHERE enabled AND platform IN ({pl}) ORDER BY platform,
login""").fetchall()
        show("1. Inactive but enabled (AC-2(3)(d))", ["platform", "login", "basis_time", "basis",
"days_idle", "status"],
             rows, f"state on {AS_OF}", f"enabled accounts on {', '.join(PLATFORMS_EXPORTED)};
INACTIVE_DAYS={INACTIVE_DAYS}",
             "1_inactive_but_enabled.csv")
        rep.follow_up["inactive"] = sum(r[5] != "ok" for r in rows)
    else:
        not_run("1. Inactive but enabled (AC-2(3)(d))", "INACTIVE_DAYS",
"1_inactive_but_enabled.csv")

    # --- 2. temporary and emergency accounts
    if isinstance(TEMP_GRACE_DAYS, int) and TEMP_GRACE_DAYS >= 0:
        rows = con.execute(f"""WITH pop AS (
            SELECT lower(trim(platform)) AS platform, lower(trim(login)) AS login,
({day('end_date')}) AS end_date,
                   lower(trim(kind)) AS kind, 'your list' AS source FROM t_temp_accounts
            UNION ALL SELECT platform, login, expires_at::DATE, 'temporary', 'Snowflake EXPIRES_AT'
FROM accounts2 x
              WHERE platform = 'snowflake' AND expires_at IS NOT NULL
                AND NOT EXISTS (SELECT 1 FROM t_temp_accounts l WHERE lower(trim(l.platform)) =
'snowflake'
                                AND lower(trim(l.login)) = x.login))
          SELECT p.platform, p.login, p.kind, p.end_date::VARCHAR AS end_date, p.source,
            coalesce(a.enabled::VARCHAR, 'not on platform') AS enabled_now,
            (SELECT max(s.t)::VARCHAR FROM signins s WHERE s.platform = p.platform AND s.login =
p.login) AS last_use,
            least(e.d, a.expires_at::DATE, CASE WHEN a.login IS NULL THEN sd.d END)::VARCHAR AS
ended_on,
            CASE
              WHEN EXISTS (SELECT 1 FROM signins s WHERE s.platform = p.platform AND s.login =
p.login
                           AND s.t::DATE > p.end_date + {TEMP_GRACE_DAYS}) THEN 'USED AFTER END'
              WHEN a.enabled THEN CASE WHEN DATE '{AS_OF}' > p.end_date + {TEMP_GRACE_DAYS}
                                       THEN 'STILL ENABLED PAST END' ELSE 'NOT YET DUE' END
              WHEN least(e.d, a.expires_at::DATE, CASE WHEN a.login IS NULL THEN sd.d END) >
p.end_date + {TEMP_GRACE_DAYS} THEN 'ENDED LATE'
              WHEN least(e.d, a.expires_at::DATE, CASE WHEN a.login IS NULL THEN sd.d END) IS NOT
NULL THEN 'ended on time'
              WHEN a.login IS NULL THEN 'NOT FOUND ON PLATFORM' ELSE 'ENDED, DATE NOT RECORDED' END
AS status
          FROM pop p LEFT JOIN accounts2 a ON a.platform = p.platform AND a.login = p.login
          LEFT JOIN (SELECT x.platform, x.target, min(x.t)::DATE AS d FROM actions x
                     LEFT JOIN accounts2 c ON c.platform = x.platform AND c.login = x.target
```

```
                    WHERE x.kind IN ('remove', 'disable')
                      AND x.t >= coalesce(c.created_on, (SELECT n.since FROM since n WHERE
n.platform = x.platform
                                          AND n.target = x.target), TIMESTAMP '1900-01-01')
                    GROUP BY 1, 2) e ON e.platform = p.platform AND e.target = p.login
          LEFT JOIN ended_sf sd ON p.platform = 'snowflake' AND sd.login = p.login
          WHERE p.platform IN ({pl}) AND p.end_date <= DATE '{PERIOD_END}' ORDER BY 1,
2""").fetchall()
        show("2. Temporary and emergency accounts past their end (AC-2(2), AC-2(3)(a))",
             ["platform", "login", "kind", "end_date", "source", "enabled_now", "last_use",
"ended_on", "status"], rows,
             f"end dates up to {PERIOD_END}; state on {AS_OF}",
             f"your temp_accounts list plus Snowflake EXPIRES_AT; TEMP_GRACE_DAYS=
{TEMP_GRACE_DAYS}",
             "2_temporary_accounts.csv")
        rep.follow_up["temporary"] = sum(r[8] in ("USED AFTER END", "STILL ENABLED PAST END",
"ENDED LATE",
                                                  "NOT FOUND ON PLATFORM", "ENDED, DATE NOT
RECORDED") for r in rows)
    else:
        not_run("2. Temporary and emergency accounts past their end (AC-2(2), AC-2(3)(a))",
"TEMP_GRACE_DAYS",
                "2_temporary_accounts.csv")

    # --- 3. transfers that kept the old access
    if isinstance(TRANSFER_DAYS, int) and TRANSFER_DAYS >= 0:
        rows = con.execute(f"""WITH t AS (SELECT lower(trim(employee_email)) AS email,
({day('transfer_date')}) AS tdate,
             lower(trim(old_department)) AS old_dept, lower(trim(new_department)) AS new_dept,
trim(old_department) AS old_name
             FROM t_hr_transfers),
           tp AS (SELECT t.*, p.platform FROM t, (SELECT unnest([{pl}]) AS platform) p
                  WHERE t.tdate BETWEEN DATE '{PERIOD_START}' AND DATE '{PERIOD_END}')
         SELECT tp.email, tp.tdate::VARCHAR AS transfer_date, tp.platform,
           coalesce((SELECT string_agg(DISTINCT lower(trim(m.login)), ', ' ORDER BY
lower(trim(m.login))) FROM t_login_map m
                     WHERE lower(trim(m.platform)) = tp.platform AND lower(trim(m.employee_email))
= tp.email), '') AS logins,
           coalesce((SELECT string_agg(DISTINCT h.role_name, ', ' ORDER BY h.role_name) FROM
t_login_map m
                     JOIN holdings h ON h.platform = tp.platform AND h.login =
lower(trim(m.login))
                     JOIN t_role_departments r ON lower(trim(r.platform)) = tp.platform AND
lower(trim(r.role_name)) = h.role_name
                     WHERE lower(trim(m.platform)) = tp.platform AND lower(trim(m.employee_email))
= tp.email
                       AND lower(trim(r.department)) = tp.old_dept
                       AND NOT EXISTS (SELECT 1 FROM t_role_departments r2 WHERE
lower(trim(r2.platform)) = tp.platform
                            AND lower(trim(r2.role_name)) = h.role_name AND
lower(trim(r2.department)) = tp.new_dept)), '')
                    AS old_access_held,
           CASE WHEN NOT EXISTS (SELECT 1 FROM t_login_map m WHERE lower(trim(m.platform)) =
tp.platform
                                 AND lower(trim(m.employee_email)) = tp.email) THEN 'no login on
this platform'
                WHEN tp.tdate + {TRANSFER_DAYS} > DATE '{AS_OF}' THEN 'NOT YET DUE'
                WHEN EXISTS (SELECT 1 FROM t_login_map m JOIN holdings h ON h.platform =
```

```
tp.platform
                              AND h.login = lower(trim(m.login))
                              JOIN t_role_departments r ON lower(trim(r.platform)) = tp.platform
                              AND lower(trim(r.role_name)) = h.role_name
                              WHERE lower(trim(m.platform)) = tp.platform AND
lower(trim(m.employee_email)) = tp.email
                              AND lower(trim(r.department)) = tp.old_dept
                              AND NOT EXISTS (SELECT 1 FROM t_role_departments r2 WHERE
lower(trim(r2.platform)) = tp.platform
                                  AND lower(trim(r2.role_name)) = h.role_name AND
lower(trim(r2.department)) = tp.new_dept))
                 THEN 'KEPT OLD ACCESS' ELSE 'clean' END AS status
          FROM tp ORDER BY 1, 2, 3""").fetchall()
        show("3. Transfers that kept the old access (AC-2 h.2 and l)",
             ["employee", "transfer_date", "platform", "logins", "old_access_held", "status"],
rows,
             f"transfers dated {PERIOD_START} to {PERIOD_END}; holdings on {AS_OF}",
             f"roles and groups your department map ties to the old department; TRANSFER_DAYS=
{TRANSFER_DAYS}",
             "3_transfers.csv")
        rep.follow_up["transfers"] = sum(r[5] == "KEPT OLD ACCESS" for r in rows)
        un = con.execute(f"""SELECT count(*) FROM (SELECT DISTINCT h.platform, h.login, h.role_name
FROM
            (SELECT lower(trim(employee_email)) AS email FROM t_hr_transfers WHERE
({day('transfer_date')})
              BETWEEN DATE '{PERIOD_START}' AND DATE '{PERIOD_END}') t
            JOIN t_login_map m ON lower(trim(m.employee_email)) = t.email
            JOIN holdings h ON h.platform = lower(trim(m.platform)) AND h.login =
lower(trim(m.login))
            WHERE h.platform IN ({pl}) AND NOT EXISTS (SELECT 1 FROM t_role_departments r
                 WHERE lower(trim(r.platform)) = h.platform AND lower(trim(r.role_name)) =
h.role_name))""").fetchone()[0]
        print(f"   held roles and groups of transferred people that your department map does not
list: {un}")
    else:
        not_run("3. Transfers that kept the old access (AC-2 h.2 and l)", "TRANSFER_DAYS",
"3_transfers.csv")

    # --- 4. account actions
    acts = con.execute(f"""SELECT a.platform, a.t::VARCHAR AS at_utc, a.kind, a.target,
            coalesce({blank('a.actor')}, '(not in the record)') AS actor, a.source,
            coalesce((SELECT string_agg(k.ticket, ', ' ORDER BY k.ticket) FROM t_change_tickets k
                      WHERE lower(trim(k.platform)) = a.platform AND lower(trim(k.login)) =
a.target
                        AND lower(trim(k.action)) IN (a.kind, a.ticket_kind) AND
({day('k.change_date')}) = a.t::DATE),
                     'no ticket') AS ticket
        FROM actions a WHERE a.t::DATE BETWEEN DATE '{PERIOD_START}' AND DATE '{PERIOD_END}'
          AND a.platform IN ({pl}) ORDER BY a.platform, a.t, a.kind, a.target,
a.source""").fetchall()
    show("4a. Account actions the platforms recorded (AC-2(4))",
         ["platform", "at_utc", "kind", "target", "actor", "source", "ticket"], acts,
         f"{PERIOD_START} to {PERIOD_END} (UTC)", "every create, modify, enable, disable and
remove; successful only",
         "4a_account_actions.csv")
    by = con.execute("SELECT platform, kind, count(*) FROM (SELECT * FROM actions WHERE t::DATE
BETWEEN DATE '" +
                     PERIOD_START + "' AND DATE '" + PERIOD_END + "' AND platform IN (" + pl +
```

```
                    )) GROUP BY 1, 2 ORDER BY 1, 2").fetchall()
    print("   counts: " + "; ".join(f"{p} {k} {n}" for p, k, n in by))
    miss = con.execute(f"""SELECT k.ticket, lower(trim(k.platform)) AS platform,
lower(trim(k.login)) AS login,
           lower(trim(k.action)) AS action, ({day('k.change_date')})::VARCHAR AS change_date, 'NOT
RECORDED' AS status
        FROM t_change_tickets k WHERE ({day('k.change_date')}) BETWEEN DATE '{PERIOD_START}' AND
DATE '{PERIOD_END}'
          AND lower(trim(k.platform)) IN ({pl}) AND lower(trim(k.action)) IN ({quote_list(KINDS)})
          AND NOT EXISTS (SELECT 1 FROM actions a WHERE a.platform = lower(trim(k.platform))
              AND a.target = lower(trim(k.login)) AND lower(trim(k.action)) IN (a.kind,
a.ticket_kind)
              AND a.t::DATE = ({day('k.change_date')}))
        ORDER BY 1""").fetchall()
    show("4b. Change tickets with no recorded action (AC-2(4))",
         ["ticket", "platform", "login", "action", "change_date", "status"], miss,
         f"{PERIOD_START} to {PERIOD_END} (UTC)", "your change_tickets, matched on platform, login,
action and day",
         "4b_tickets_not_recorded.csv")
    rep.follow_up["not_recorded"] = len(miss)
    nt = sum(1 for r in acts if r[6] == "no ticket")
    print(f"   recorded actions with no ticket: {nt} of {len(acts)} (a count to look at, not a
finding)")

    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 ["inactive", "temporary", "transfers", "not_recorded"]]
    print(f"\nNeeds follow-up: {', '.join(parts)}; input checks failed: {len(failed)}. "
          "The script produces evidence; it does not say a control is met.")
    if "--strict" in args and (total or failed):
        return 1
    return 0


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

The runs

Run 1: clean generated records (--demo)

```
Generated records written to generated/ (seed 232; not a company's data). Demo settings:
INACTIVE_DAYS=90, TEMP_GRACE_DAYS=1, TRANSFER_DAYS=14.

Run at 2026-11-01 17:48:48 UTC. Period 2026-09-01 to 2026-09-30 (UTC); exports taken 2026-10-05;
platforms exported: snowflake, databricks, redshift; not checked: none.

Input checks
  [ok] dates and platforms in the settings
  [ok] every time period is set (AC-2 leaves each one to you)
  [ok] every file is there with every column the script reads
      read 8 rows into snowflake_users
      read 7 rows into snowflake_grants_to_users
```

```
      read 365 rows into snowflake_query_history
      read 8 rows into databricks_users
      read 5 rows into databricks_group_members
      read 354 rows into databricks_audit
      read 5 rows into redshift_svv_user_info
      read 3 rows into redshift_svv_user_grants
      read 7 rows into redshift_sys_userlog
      read 265 rows into redshift_sys_connection_log
      read 10 rows into hr_people
      read 1 rows into hr_transfers
      read 21 rows into login_map
      read 4 rows into temp_accounts
      read 8 rows into role_departments
      read 7 rows into change_tickets
  [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 successful CREATE, ALTER and DROP USER statement and role grant to a user could be
read: 1 failed user statement(s) not counted
  [ok] every Databricks action that looks like user management is one the script lists
  [ok] every Databricks action that looks like a sign-in is on the sign-in list
  [ok] every Databricks sign-in and account action has a readable response status and workspace_id
  [ok] every SYS_USERLOG action is alter, create, drop or rename
  [ok] no login appears twice in one platform's account list
  [ok] no login is mapped to two people
  [ok] every role or group in the department map is held by someone
  [ok] every transfer names someone in the HR people list
      Databricks group members that are not account users (groups or service principals; not
expanded): 0
  [ok] snowflake: records exist for every day from 2026-07-07 to 2026-10-05
  [ok] databricks: records exist for every day from 2026-07-07 to 2026-10-05
  [ok] redshift: records exist for every day from 2026-07-07 to 2026-10-05

17 input checks, 0 failed.

== 1. Inactive but enabled (AC-2(3)(d))
   period: state on 2026-10-05; filters: enabled accounts on snowflake, databricks, redshift;
INACTIVE_DAYS=90; run at 2026-11-01 17:48:48 UTC; rows: 17; sha256:
d048000b100d5da3038827b24c8797b537b486c5e0e9ef999591267466bc814c
   platform   | login                | basis_time          | basis        | days_idle | status
   databricks | ana@example.test     | 2026-10-02 11:31:00 | last sign-in | 3         | ok
   databricks | ben@example.test     | 2026-10-02 12:19:00 | last sign-in | 3         | ok
   databricks | chen@example.test    | 2026-10-02 13:34:00 | last sign-in | 3         | ok
   databricks | dara@example.test    | 2026-10-02 10:55:00 | last sign-in | 3         | ok
   databricks | sam@example.test     | 2026-10-02 09:10:00 | last sign-in | 3         | ok
   databricks | svc-etl@example.test | 2026-10-05 08:09:00 | last sign-in | 0         | ok
   redshift   | ana                  | 2026-10-02 15:36:00 | last sign-in | 3         | ok
   redshift   | chen                 | 2026-10-02 08:20:00 | last sign-in | 3         | ok
   redshift   | kim                  | 2026-10-02 10:55:00 | last sign-in | 3         | ok
   redshift   | sam                  | 2026-10-02 16:48:00 | last sign-in | 3         | ok
   redshift   | svc_bi               | 2026-10-05 08:01:00 | last sign-in | 0         | ok
   snowflake  | admin_sam            | 2026-10-02 16:18:00 | last sign-in | 3         | ok
   snowflake  | ana                  | 2026-10-02 12:20:00 | last sign-in | 3         | ok
   snowflake  | ben                  | 2026-10-02 10:43:00 | last sign-in | 3         | ok
   snowflake  | chen                 | 2026-10-02 11:39:00 | last sign-in | 3         | ok
   snowflake  | dara                 | 2026-10-02 14:39:00 | last sign-in | 3         | ok
   snowflake  | svc_load             | 2026-10-05 12:01:00 | last sign-in | 0         | ok

== 2. Temporary and emergency accounts past their end (AC-2(2), AC-2(3)(a))
```

```
   period: end dates up to 2026-09-30; state on 2026-10-05; filters: your temp_accounts list plus
Snowflake EXPIRES_AT; TEMP_GRACE_DAYS=1; run at 2026-11-01 17:48:48 UTC; rows: 4; sha256:
d2392a8479d654921d3876b2bbd2bab68dbd2ead72b309f8e7f5fdb9bad235a6
   platform   | login                      | kind      | end_date   | source    | enabled_now
| last_use            | ended_on   | status
   databricks | eli.contractor@example.test | temporary | 2026-09-15 | your list | false
| 2026-09-14 10:32:00 | 2026-09-15 | ended on time
   redshift   | eli_temp                   | temporary | 2026-09-15 | your list | not on platform
| 2026-09-14 09:23:00 | 2026-09-16 | ended on time
   redshift   | fire_call                  | emergency | 2026-09-12 | your list | not on platform
| 2026-09-11 08:40:00 | 2026-09-13 | ended on time
   snowflake  | eli_temp                   | temporary | 2026-09-20 | your list | false
| 2026-09-18 09:17:00 | 2026-09-20 | ended on time

== 3. Transfers that kept the old access (AC-2 h.2 and l)
   period: transfers dated 2026-09-01 to 2026-09-30; holdings on 2026-10-05; filters: roles and
groups your department map ties to the old department; TRANSFER_DAYS=14; run at 2026-11-01 17:48:48
UTC; rows: 3; sha256: 6c9f6c2990a277c2e6dac8eb84984269fa378df0caa06ec8959c900fd6e33a7d
   employee          | transfer_date | platform   | logins            | old_access_held | status
   dara@example.test | 2026-09-10    | databricks | dara@example.test |                 | clean
   dara@example.test | 2026-09-10    | redshift   |                   |                 | no login
on this platform
   dara@example.test | 2026-09-10    | snowflake  | dara              |                 | clean
   held roles and groups of transferred people that your department map does not list: 0

== 4a. Account actions the platforms recorded (AC-2(4))
   period: 2026-09-01 to 2026-09-30 (UTC); filters: every create, modify, enable, disable and
remove; successful only; run at 2026-11-01 17:48:48 UTC; rows: 15; sha256:
bd7ce2d7edd371e730232fce6accba0ac41f3425a8348ac8b7f4e622b92c5c78
   platform   | at_utc              | kind    | target                     | actor            |
source                              | ticket
   databricks | 2026-09-01 10:00:00 | create  | eli.contractor@example.test | sam@example.test |
audit add                           | no ticket
   databricks | 2026-09-01 10:00:00 | modify  | eli.contractor@example.test | sam@example.test |
audit add (workspace 1234567890123456) | no ticket
   databricks | 2026-09-10 10:00:00 | modify  | dara@example.test           | sam@example.test |
audit addPrincipalToGroup           | no ticket
   databricks | 2026-09-10 10:00:00 | modify  | dara@example.test           | sam@example.test |
audit removePrincipalFromGroup      | no ticket
   databricks | 2026-09-15 10:00:00 | disable | eli.contractor@example.test | sam@example.test |
audit deactivateUser                | T-103
   redshift   | 2026-09-01 10:00:00 | create  | eli_temp                   | user_id 100      |
sys_userlog create                  | no ticket
   redshift   | 2026-09-02 10:00:00 | remove  | jon                        | user_id 100      |
sys_userlog drop                    | T-102
   redshift   | 2026-09-03 11:00:00 | modify  | ana                        | user_id 100      |
sys_userlog alter                   | no ticket
   redshift   | 2026-09-08 10:00:00 | create  | kim                        | user_id 100      |
sys_userlog create                  | T-101
   redshift   | 2026-09-11 10:00:00 | create  | fire_call                  | user_id 100      |
sys_userlog create                  | no ticket
   redshift   | 2026-09-13 10:00:00 | remove  | fire_call                  | user_id 100      |
sys_userlog drop                    | T-106
   redshift   | 2026-09-16 10:00:00 | remove  | eli_temp                   | user_id 100      |
sys_userlog drop                    | T-107
   snowflake  | 2026-09-01 10:00:00 | create  | eli_temp                   | admin_sam        |
query q00361                        | T-104
   snowflake  | 2026-09-10 10:00:00 | modify  | dara                       | admin_sam        |
```

```
query q00362                            | T-105
   snowflake  | 2026-09-10 10:00:00 | modify  | dara                          | admin_sam       |
query q00365                            | T-105
   counts: databricks create 1; databricks disable 1; databricks modify 3; redshift create 3;
redshift modify 1; redshift remove 3; snowflake create 1; snowflake modify 2

== 4b. Change tickets with no recorded action (AC-2(4))
   period: 2026-09-01 to 2026-09-30 (UTC); filters: your change_tickets, matched on platform,
login, action and day; run at 2026-11-01 17:48:48 UTC; rows: 0; sha256:
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
   recorded actions with no ticket: 7 of 15 (a count to look at, not a finding)

Needs follow-up: inactive 0, temporary 0, transfers 0, not recorded 0; input checks failed: 0. The
script produces evidence; it does not say a control is met.
```

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

```
Generated records written to generated/ (seed 232; not a company's data). Demo settings:
INACTIVE_DAYS=90, TEMP_GRACE_DAYS=1, TRANSFER_DAYS=14.

Run at 2026-11-01 17:48:49 UTC. Period 2026-09-01 to 2026-09-30 (UTC); exports taken 2026-10-05;
platforms exported: snowflake, databricks, redshift; not checked: none.

Input checks
  [ok] dates and platforms in the settings
  [ok] every time period is set (AC-2 leaves each one to you)
  [ok] every file is there with every column the script reads
      read 8 rows into snowflake_users
      read 7 rows into snowflake_grants_to_users
      read 366 rows into snowflake_query_history
      read 8 rows into databricks_users
      read 6 rows into databricks_group_members
      read 357 rows into databricks_audit
      read 7 rows into redshift_svv_user_info
      read 4 rows into redshift_svv_user_grants
      read 5 rows into redshift_sys_userlog
      read 266 rows into redshift_sys_connection_log
      read 10 rows into hr_people
      read 1 rows into hr_transfers
      read 21 rows into login_map
      read 4 rows into temp_accounts
      read 8 rows into role_departments
      read 8 rows into change_tickets
  [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 successful CREATE, ALTER and DROP USER statement and role grant to a user could be
read: 1 failed user statement(s) not counted
  [ok] every Databricks action that looks like user management is one the script lists
  [ok] every Databricks action that looks like a sign-in is on the sign-in list
  [ok] every Databricks sign-in and account action has a readable response status and workspace_id
  [ok] every SYS_USERLOG action is alter, create, drop or rename
  [ok] no login appears twice in one platform's account list
  [ok] no login is mapped to two people
  [ok] every role or group in the department map is held by someone
  [ok] every transfer names someone in the HR people list
      Databricks group members that are not account users (groups or service principals; not
```

```
expanded): 0
  [ok] snowflake: records exist for every day from 2026-07-07 to 2026-10-05
  [ok] databricks: records exist for every day from 2026-07-07 to 2026-10-05
  [ok] redshift: records exist for every day from 2026-07-07 to 2026-10-05

17 input checks, 0 failed.

== 1. Inactive but enabled (AC-2(3)(d))
   period: state on 2026-10-05; filters: enabled accounts on snowflake, databricks, redshift;
INACTIVE_DAYS=90; run at 2026-11-01 17:48:49 UTC; rows: 23; sha256:
4a82cd1d22d0806e352830659a443c97ed9a86ec15a7f6a9413e2bf699de7939
   platform   | login                       | basis_time          | basis                     |
days_idle | status
   databricks | ana@example.test            | 2026-10-02 14:31:00 | last sign-in              | 3
| ok
   databricks | ben@example.test            | 2026-10-02 13:05:00 | last sign-in              | 3
| ok
   databricks | chen@example.test           | 2026-10-02 10:16:00 | last sign-in              | 3
| ok
   databricks | dara@example.test           | 2026-10-02 09:16:00 | last sign-in              | 3
| ok
   databricks | eli.contractor@example.test | 2026-09-22 11:24:00 | last sign-in              | 13
| ok
   databricks | ivy@example.test            |                     | no sign-in in the records |
| INACTIVE BUT ENABLED
   databricks | sam@example.test            | 2026-10-02 11:45:00 | last sign-in              | 3
| ok
   databricks | svc-etl@example.test        | 2026-10-05 11:04:00 | last sign-in              | 0
| ok
   redshift   | ana                         | 2026-10-02 11:22:00 | last sign-in              | 3
| ok
   redshift   | chen                        | 2026-10-02 12:59:00 | last sign-in              | 3
| ok
   redshift   | fire_call                   | 2026-09-18 15:17:00 | last sign-in              | 17
| ok
   redshift   | jon                         |                     | no sign-in in the records |
| INACTIVE BUT ENABLED
   redshift   | kim                         | 2026-10-02 16:02:00 | last sign-in              | 3
| ok
   redshift   | sam                         | 2026-10-02 08:29:00 | last sign-in              | 3
| ok
   redshift   | svc_bi                      | 2026-10-05 11:50:00 | last sign-in              | 0
| ok
   snowflake  | admin_sam                   | 2026-10-02 16:18:00 | last sign-in              | 3
| ok
   snowflake  | ana                         | 2026-10-02 12:20:00 | last sign-in              | 3
| ok
   snowflake  | ben                         | 2026-10-02 10:43:00 | last sign-in              | 3
| ok
   snowflake  | chen                        | 2026-10-02 11:39:00 | last sign-in              | 3
| ok
   snowflake  | dara                        | 2026-10-02 14:39:00 | last sign-in              | 3
| ok
   snowflake  | eli_temp                    | 2026-09-28 10:49:00 | last sign-in              | 7
| ok
   snowflake  | hal                         | 2026-05-14 08:00:00 | last sign-in              | 144
| INACTIVE BUT ENABLED
   snowflake  | svc_load                    | 2026-10-05 10:53:00 | last sign-in              | 0
```

```
| ok

== 2. Temporary and emergency accounts past their end (AC-2(2), AC-2(3)(a))
   period: end dates up to 2026-09-30; state on 2026-10-05; filters: your temp_accounts list plus
Snowflake EXPIRES_AT; TEMP_GRACE_DAYS=1; run at 2026-11-01 17:48:49 UTC; rows: 4; sha256:
1f476bb91389c9f78007006a2f6cac4e40faa8b1f9d7ea3627913b0b16faf6c1
   platform   | login                      | kind      | end_date   | source    | enabled_now
| last_use            | ended_on   | status
   databricks | eli.contractor@example.test | temporary | 2026-09-15 | your list | true
| 2026-09-22 11:24:00 |            | USED AFTER END
   redshift   | eli_temp                   | temporary | 2026-09-15 | your list | not on platform
| 2026-09-14 14:46:00 | 2026-09-30 | ENDED LATE
   redshift   | fire_call                  | emergency | 2026-09-12 | your list | true
| 2026-09-18 15:17:00 |            | USED AFTER END
   snowflake  | eli_temp                   | temporary | 2026-09-20 | your list | true
| 2026-09-28 10:49:00 |            | USED AFTER END

== 3. Transfers that kept the old access (AC-2 h.2 and l)
   period: transfers dated 2026-09-01 to 2026-09-30; holdings on 2026-10-05; filters: roles and
groups your department map ties to the old department; TRANSFER_DAYS=14; run at 2026-11-01 17:48:49
UTC; rows: 3; sha256: 2b66125c4797dc17a23c83217957633511e74961d3bc16dd2ea5ec11280743da
   employee          | transfer_date | platform   | logins            | old_access_held | status
   dara@example.test | 2026-09-10    | databricks | dara@example.test | finance-readers | KEPT OLD
ACCESS
   dara@example.test | 2026-09-10    | redshift   |                   |                 | no login
on this platform
   dara@example.test | 2026-09-10    | snowflake  | dara              | fin_analyst     | KEPT OLD
ACCESS
   held roles and groups of transferred people that your department map does not list: 0

== 4a. Account actions the platforms recorded (AC-2(4))
   period: 2026-09-01 to 2026-09-30 (UTC); filters: every create, modify, enable, disable and
remove; successful only; run at 2026-11-01 17:48:49 UTC; rows: 11; sha256:
75a87feba37a65fe2561df234b08dcc1e178822e80f91766e870a7cd88ca7ced
   platform   | at_utc              | kind   | target                     | actor           |
source                                | ticket
   databricks | 2026-09-01 10:00:00 | create | eli.contractor@example.test | sam@example.test |
audit add                             | no ticket
   databricks | 2026-09-01 10:00:00 | modify | eli.contractor@example.test | sam@example.test |
audit add (workspace 1234567890123456)    | no ticket
   databricks | 2026-09-10 10:00:00 | modify | dara@example.test          | sam@example.test |
audit addPrincipalToGroup             | no ticket
   databricks | 2026-09-15 10:00:00 | modify | eli.contractor@example.test | sam@example.test |
audit delete (workspace 1234567890123456) | no ticket
   redshift   | 2026-09-01 10:00:00 | create | eli_temp                   | user_id 100     |
sys_userlog create                    | no ticket
   redshift   | 2026-09-03 11:00:00 | modify | ana                        | user_id 100     |
sys_userlog alter                     | no ticket
   redshift   | 2026-09-08 10:00:00 | create | kim                        | user_id 100     |
sys_userlog create                    | T-101
   redshift   | 2026-09-11 10:00:00 | create | fire_call                  | user_id 100     |
sys_userlog create                    | no ticket
   redshift   | 2026-09-30 10:00:00 | remove | eli_temp                   | user_id 100     |
sys_userlog drop                      | no ticket
   snowflake  | 2026-09-01 10:00:00 | create | eli_temp                   | admin_sam       |
query q00364                          | T-104
   snowflake  | 2026-09-10 10:00:00 | modify | dara                       | admin_sam       |
query q00365                          | T-105
```

```
   counts: databricks create 1; databricks modify 3; redshift create 3; redshift modify 1; redshift
remove 1; snowflake create 1; snowflake modify 1

== 4b. Change tickets with no recorded action (AC-2(4))
   period: 2026-09-01 to 2026-09-30 (UTC); filters: your change_tickets, matched on platform,
login, action and day; run at 2026-11-01 17:48:49 UTC; rows: 5; sha256:
469e8c35302158eb351f8716cb8ee1a5c844efb21069856cb79111a3cb264988
   ticket | platform   | login                       | action  | change_date | status
   T-102  | redshift   | jon                         | remove  | 2026-09-02  | NOT RECORDED
   T-103  | databricks | eli.contractor@example.test | disable | 2026-09-15  | NOT RECORDED
   T-106  | redshift   | fire_call                   | remove  | 2026-09-13  | NOT RECORDED
   T-107  | redshift   | eli_temp                    | remove  | 2026-09-16  | NOT RECORDED
   T-108  | redshift   | lee                         | create  | 2026-09-20  | NOT RECORDED
   recorded actions with no ticket: 8 of 11 (a count to look at, not a finding)

Needs follow-up: inactive 3, temporary 4, transfers 2, not recorded 5; input checks failed: 0. The
script produces evidence; it does not say a control is met.
```

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

```
Generated records written to generated/ (seed 232; not a company's data). Demo settings:
INACTIVE_DAYS=90, TEMP_GRACE_DAYS=1, TRANSFER_DAYS=14.

Run at 2026-11-01 17:48:50 UTC. Period 2026-09-01 to 2026-09-30 (UTC); exports taken 2026-10-05;
platforms exported: snowflake, databricks, redshift; not checked: none.

Input checks
  [ok] dates and platforms in the settings
  [FAIL] every time period is set (AC-2 leaves each one to you): not set to a whole number of days:
TRANSFER_DAYS
  [FAIL] every file is there with every column the script reads: redshift_svv_user_grants.csv has
no column role_name
      read 9 rows into snowflake_users
      read 7 rows into snowflake_grants_to_users
      read 366 rows into snowflake_query_history
      read 8 rows into databricks_users
      read 5 rows into databricks_group_members
      read 357 rows into databricks_audit
      read 5 rows into redshift_svv_user_info
      read 0 rows into redshift_svv_user_grants
      read 8 rows into redshift_sys_userlog
      read 259 rows into redshift_sys_connection_log
      read 10 rows into hr_people
      read 2 rows into hr_transfers
      read 22 rows into login_map
      read 4 rows into temp_accounts
      read 10 rows into role_departments
      read 8 rows into change_tickets
  [FAIL] every date and time is written as YYYY-MM-DD [HH:MM:SS] (UTC): not readable, so left out:
snowflake_users.created_on: 1, temp_accounts.end_date: 1
  [FAIL] every value is one the script knows: snowflake_users.disabled not true or false: 1;
role_departments.platform not one of snowflake, databricks, redshift: 1; temp_accounts.kind not
temporary or emergency: 1; change_tickets.action not one of create, modify, enable, disable,
remove: 1
  [FAIL] every successful CREATE, ALTER and DROP USER statement and role grant to a user could be
read: 1 successful statement(s) look like one but name no user the script can read
```

```
  [FAIL] every Databricks action that looks like user management is one the script lists: unknown:
updateUserRole
  [FAIL] every Databricks action that looks like a sign-in is on the sign-in list: unknown:
passkeyLogin
  [FAIL] every Databricks sign-in and account action has a readable response status and
workspace_id: 1 row(s) whose response is not JSON with a status code
  [FAIL] every SYS_USERLOG action is alter, create, drop or rename: unknown: suspend
  [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] every role or group in the department map is held by someone: matches no held role:
redshift/data_rw, redshift/fin_ro, snowflake/ghost_role
  [FAIL] every transfer names someone in the HR people list: not in hr_people: zed@example.test
      Databricks group members that are not account users (groups or service principals; not
expanded): 0
  [ok] snowflake: records exist for every day from 2026-07-07 to 2026-10-05
  [ok] databricks: records exist for every day from 2026-07-07 to 2026-10-05
  [FAIL] redshift: records exist for every day from 2026-07-07 to 2026-10-05: 1 day(s) with none,
first 2026-09-14, last 2026-09-14

17 input checks, 14 failed.  ** INCOMPLETE: 14 input check(s) failed; read the lists with that in
mind **

== 1. Inactive but enabled (AC-2(3)(d))
   period: state on 2026-10-05; filters: enabled accounts on snowflake, databricks, redshift;
INACTIVE_DAYS=90; run at 2026-11-01 17:48:50 UTC; rows: 18; sha256:
570999d06f1c0162643bdd6d9f8f5f22d3a9de47a93ef6b7ff4f7a02b60ed694; INCOMPLETE
   platform   | login                | basis_time          | basis        | days_idle | status
   databricks | ana@example.test     | 2026-10-02 11:31:00 | last sign-in | 3         | ok
   databricks | ben@example.test     | 2026-10-02 12:19:00 | last sign-in | 3         | ok
   databricks | chen@example.test    | 2026-10-02 13:34:00 | last sign-in | 3         | ok
   databricks | dara@example.test    | 2026-10-02 10:55:00 | last sign-in | 3         | ok
   databricks | sam@example.test     | 2026-10-02 09:10:00 | last sign-in | 3         | ok
   databricks | svc-etl@example.test | 2026-10-05 08:09:00 | last sign-in | 0         | ok
   redshift   | ana                  | 2026-10-02 15:36:00 | last sign-in | 3         | ok
   redshift   | chen                 | 2026-10-02 08:20:00 | last sign-in | 3         | ok
   redshift   | kim                  | 2026-10-02 10:55:00 | last sign-in | 3         | ok
   redshift   | sam                  | 2026-10-02 16:48:00 | last sign-in | 3         | ok
   redshift   | svc_bi               | 2026-10-05 08:01:00 | last sign-in | 0         | ok
   snowflake  | admin_sam            | 2026-10-02 16:18:00 | last sign-in | 3         | ok
   snowflake  | ana                  | 2026-10-02 12:20:00 | last sign-in | 3         | ok
   snowflake  | ana                  | 2026-10-02 12:20:00 | last sign-in | 3         | ok
   snowflake  | ben                  | 2026-10-02 10:43:00 | last sign-in | 3         | ok
   snowflake  | chen                 | 2026-10-02 11:39:00 | last sign-in | 3         | ok
   snowflake  | dara                 | 2026-10-02 14:39:00 | last sign-in | 3         | ok
   snowflake  | svc_load             | 2026-10-05 12:01:00 | last sign-in | 0         | ok

== 2. Temporary and emergency accounts past their end (AC-2(2), AC-2(3)(a))
   period: end dates up to 2026-09-30; state on 2026-10-05; filters: your temp_accounts list plus
Snowflake EXPIRES_AT; TEMP_GRACE_DAYS=1; run at 2026-11-01 17:48:50 UTC; rows: 3; sha256:
7f75c8790fba9e64a8ccbe158c4f45186afd17a75b013f3fa97e4bb94124667a; INCOMPLETE
   platform   | login                       | kind      | end_date   | source    | enabled_now
| last_use            | ended_on   | status
   databricks | eli.contractor@example.test | tmp       | 2026-09-15 | your list | false
| 2026-09-14 10:32:00 | 2026-09-15 | ended on time
   redshift   | eli_temp                    | temporary | 2026-09-15 | your list | not on platform
| 2026-09-11 16:02:00 | 2026-09-16 | ended on time
   redshift   | fire_call                   | emergency | 2026-09-12 | your list | not on platform
| 2026-09-11 08:40:00 | 2026-09-13 | ended on time
```

```
== 3. Transfers that kept the old access (AC-2 h.2 and l)
   NOT RUN: TRANSFER_DAYS is not set

== 4a. Account actions the platforms recorded (AC-2(4))
   period: 2026-09-01 to 2026-09-30 (UTC); filters: every create, modify, enable, disable and
remove; successful only; run at 2026-11-01 17:48:50 UTC; rows: 15; sha256:
bd7ce2d7edd371e730232fce6accba0ac41f3425a8348ac8b7f4e622b92c5c78; INCOMPLETE
   platform   | at_utc              | kind    | target                       | actor            |
source                                | ticket
   databricks | 2026-09-01 10:00:00 | create  | eli.contractor@example.test | sam@example.test |
audit add                             | no ticket
   databricks | 2026-09-01 10:00:00 | modify  | eli.contractor@example.test | sam@example.test |
audit add (workspace 1234567890123456) | no ticket
   databricks | 2026-09-10 10:00:00 | modify  | dara@example.test           | sam@example.test |
audit addPrincipalToGroup             | no ticket
   databricks | 2026-09-10 10:00:00 | modify  | dara@example.test           | sam@example.test |
audit removePrincipalFromGroup        | no ticket
   databricks | 2026-09-15 10:00:00 | disable | eli.contractor@example.test | sam@example.test |
audit deactivateUser                  | T-103
   redshift   | 2026-09-01 10:00:00 | create  | eli_temp                    | user_id 100      |
sys_userlog create                    | no ticket
   redshift   | 2026-09-02 10:00:00 | remove  | jon                         | user_id 100      |
sys_userlog drop                      | T-102
   redshift   | 2026-09-03 11:00:00 | modify  | ana                         | user_id 100      |
sys_userlog alter                     | no ticket
   redshift   | 2026-09-08 10:00:00 | create  | kim                         | user_id 100      |
sys_userlog create                    | T-101
   redshift   | 2026-09-11 10:00:00 | create  | fire_call                   | user_id 100      |
sys_userlog create                    | no ticket
   redshift   | 2026-09-13 10:00:00 | remove  | fire_call                   | user_id 100      |
sys_userlog drop                      | T-106
   redshift   | 2026-09-16 10:00:00 | remove  | eli_temp                    | user_id 100      |
sys_userlog drop                      | T-107
   snowflake  | 2026-09-01 10:00:00 | create  | eli_temp                    | admin_sam        |
query q00361                          | T-104
   snowflake  | 2026-09-10 10:00:00 | modify  | dara                        | admin_sam        |
query q00362                          | T-105
   snowflake  | 2026-09-10 10:00:00 | modify  | dara                        | admin_sam        |
query q00365                          | T-105
   counts: databricks create 1; databricks disable 1; databricks modify 3; redshift create 3;
redshift modify 1; redshift remove 3; snowflake create 1; snowflake modify 2

== 4b. Change tickets with no recorded action (AC-2(4))
   period: 2026-09-01 to 2026-09-30 (UTC); filters: your change_tickets, matched on platform,
login, action and day; run at 2026-11-01 17:48:50 UTC; rows: 0; sha256:
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855; INCOMPLETE
   recorded actions with no ticket: 7 of 15 (a count to look at, not a finding)

Needs follow-up: inactive 0, temporary 0, transfers not run, not recorded 0; input checks failed:
14. The script produces evidence; it does not say a control is met.
```

How to read the results

- **Input checks first.** If any check failed, the lists say INCOMPLETE. A list built on records with a missing day or an unreadable value is not a population you can hand over.
- **Lists 1 to 3 are findings to look at.** A flagged row is a question for the account owner, not a verdict. An account can be inactive for a reason you can document; the script cannot know. Snowflake creates an internal `WORKSHEETS_APP_USER` user the first time Snowsight is used (Snowflake: ACCOUNT_USAGE.USERS (read 2026-11-01)), and it may show in list 1.
- **List 4a is a population, not a finding.** It is every account action the script reads in the period (successful ones only), sorted and fingerprinted. If you keep the row count and SHA-256 somewhere you cannot edit, you can check later that a list matches the one you produced.
- **List 4b is where records and tickets disagree.** NOT RECORDED means no action with that login, kind and day appears in the platform's records. The change may have happened on another day, under another login, or not at all, or the platform's records no longer reach back that far.
- **Not checked is not clean.** A platform left out of `PLATFORMS_EXPORTED` is printed as not checked, and a list whose setting is blank prints NOT RUN and writes a CSV that says so.

Run it on your own data

To run it on your own data, open the download and change only the settings between the lines marked CHANGE THESE FOR YOUR OWN DATA: set EXPORT_DIR to the folder that holds your exports, PERIOD_START, PERIOD_END and AS_OF to your dates, PLATFORMS_EXPORTED to the platforms you export, and INACTIVE_DAYS, TEMP_GRACE_DAYS and TRANSFER_DAYS to the number of days your own policy sets. Save each export under the file name the table gives, with the platform's own column names. Change nothing else: names are matched without regard to case, and a blank value is read as missing. Then run it without `--demo`.

We tested those steps on a second, separate set of generated exports: other logins in mixed case, upper-case Snowflake headers, fractional seconds, a blank department, a blank email in the login map, a CREATE USER statement with a comment and quotes, 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 16 input checks with none failing, and reported one inactive account, two temporary accounts to follow up (one used after its end date, and one whose Snowflake `EXPIRES_AT` came after the end date on the list), a third found only through its `EXPIRES_AT` that ended on time, one transfer that kept the old access despite a difference in case, and one ticket with no recorded action. That test is also generated data.

Checklist

Tick each one

Done	Step	Where
☐	Set INACTIVE_DAYS, TEMP_GRACE_DAYS and TRANSFER_DAYS from your own policy	Settings block
☐	Export the ten platform files with a user who sees every row	Step 1
☐	Write the six lists you keep: HR people, HR transfers, login map, temporary accounts, department map, change tickets	Step 1
☐	Run the script without --demo and read the input checks first	Step 4
☐	If any input check failed, fix the input and run again before you read the lists	Step 6
☐	Look at every flagged row in lists 1 to 3 and record the owner's answer	Step 5
☐	Keep list 4a with its period, filters, run time, row count and SHA-256	How to read the results
☐	Follow up every NOT RECORDED ticket in list 4b	How to read the results
☐	Note every platform printed as not checked and every list that printed NOT RUN	How to read the results
☐	Schedule the run and keep every run's lists and hashes	When this is not enough

Method

- **Input:** generated records written by the script as CSV in each platform's published column names (the links in step 1), with a fixed seed (232). The HR lists, the login map, the temporary account list, the department map and the tickets are generated too. The people, roles and counts are invented.
- **Period:** 2026-09-01 to 2026-09-30, UTC, inclusive. Records are read as of the end of the export day, 2026-10-05, and must cover every day from INACTIVE_DAYS before it (90 in the demo), which is why the day-by-day checks start on 2026-07-07.
- **Fingerprints:** each list is sorted on every column before it is hashed, so the same input gives the same SHA-256 on every run.
- **Our assumptions, to check against one of your own rows:** a Databricks sign-in is one of the actions on `DATABRICKS_SIGNIN_ACTIONS`, taken from the authentication events on the AWS

and Azure audit log reference pages; a Databricks group member's `value` is the member's `id` (the groups page names the field but does not say what it holds); a successful Databricks row has the status 200 in `response`; a Redshift sign-in is the event `authenticated` in `SYS_CONNECTION_LOG` (the page's sample shows that value, but does not list every value); Snowflake and Databricks names are lower-cased before they are compared; 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.
- **NIST sets the questions, not the answers.** Every bracketed time period is yours, and so is the choice of which enhancements apply. The script cannot tell you whether your numbers are the right ones, and the assessor decides what evidence is enough.
- **"Automatically" cannot be seen.** AC-2(2), AC-2(3) and AC-2(4) say the action is automatic. The script shows that an account ended, and when, and that actions were recorded. It cannot show that a machine did it. Snowflake's `EXPIRES_AT` is the only field we read that ends an account by itself.
- **The records may not hold what you need.** Redshift's `SYS_USERLOG` says what changed to which user and not, as far as the page tells us, who did it. `SVV_USER_INFO` lists no disabled state, so the script can see a Redshift account end only when it is dropped. Snowflake's `QUERY_HISTORY` keeps 365 days, and statements longer than 100K characters are truncated. Snowflake records SCIM provisioning requests in `REST_EVENT_HISTORY` (Snowflake: REST_EVENT_HISTORY (read 2026-11-01)), which the script does not read. Databricks keeps its audit table for 365 days, and its workspace-level events stay in each workspace's region: an export from one region misses the others, and the script cannot tell. Redshift's system log views keep about seven days unless you keep copies or turn on audit logging. Databricks group members must be collected group by group, and nested groups are not expanded.
- **Read, not enforced.** The script reads exports. It never connects to a platform and never changes an account.
- **Parsing has edges.** A Snowflake statement is read from its text. One that starts as a user change but names the user in a form the script does not read fails an input check when it is successful; it is never skipped. A user change written inside a procedure, a task or `EXECUTE IMMEDIATE` is not read from that text itself. Shared and service accounts have no HR record, so they show only in lists 1, 2 and 4.
- **Evidence, not a verdict.** The script produces evidence. It never says a control 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; Python 3

Date
: 2026-11-01

Dataset
: Generated records (seed 232) in each platform's published column names; 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.
- Access that is not an account role: object privileges granted straight to a user, and Redshift groups. List 3 reads roles and groups only.
- A policy that is silent. If your policy has no inactivity period, 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 and keep every run's lists and hashes.
- Platforms other than these three, and the other controls in the AC family.

Sources

Every document linked in this guide, with the day it was read:

- NIST OSCAL catalog of SP 800-53 Rev. 5, release 5.2.0 (read 2026-11-01): AC-2, AC-2(2), AC-2(3), AC-2(4) text, parameters
- Snowflake: ACCOUNT_USAGE.USERS (read 2026-11-01): DISABLED, LAST_SUCCESS_LOGIN, EXPIRES_AT, latency
- Snowflake: ACCOUNT_USAGE.GRANTS_TO_USERS (read 2026-11-01): role grants to users
- Snowflake: ACCOUNT_USAGE.QUERY_HISTORY (read 2026-11-01): query text, start time, status, retention
- Databricks: audit log system table (read 2026-11-01): audit table columns, account-level events

- Databricks: audit log reference (read 2026-11-01): accounts service actions
- Azure Databricks: audit log reference (read 2026-11-01): accounts service actions on Azure, Entra ID sign-ins
- Databricks: system tables overview (read 2026-11-01): audit table retention and regions
- Databricks: list account users (read 2026-11-01): SCIM users list
- Databricks: list account groups (read 2026-11-01): SCIM groups list
- Amazon Redshift: SYS_USERLOG (read 2026-11-01): user create, alter, rename, drop
- Amazon Redshift: SYS_CONNECTION_LOG (read 2026-11-01): connection events
- Amazon Redshift: SVV_USER_INFO (read 2026-11-01): users, no disabled state
- Amazon Redshift: SVV_USER_GRANTS (read 2026-11-01): roles granted to users
- Amazon Redshift: system tables and views reference (read 2026-11-01): system log retention
- Amazon Redshift: database audit logging (read 2026-11-01): audit logging to S3 and CloudWatch
- Snowflake: ALTER USER (read 2026-11-01): ALTER USER with no name
- Snowflake: CREATE OR ALTER (read 2026-11-01): objects CREATE OR ALTER supports
- Snowflake: REST_EVENT_HISTORY (read 2026-11-01): SCIM requests

The script is free. Seeing every access change each day is a different job.

The script in this guide is free to run on your own exports. Data Control Tower deploys into your own Azure subscription from a template your IT team can read before it runs. Data Control Tower lists every access change and who made it, including access someone gave by hand.

See Data Control Tower: https://www.datameaning.com/solutions/data-control-tower.html