DATA ACCESS CONCIERGE · GUIDE

# Keep Microsoft Entra access request history past 30 days

Archive the history, then check the copy for missing days, short files, and decisions with no approver or reason.

Published 2026-11-04 · Online version and downloads: www.datameaning.com/guides/keep-entra-access-request-history-past-30-days/

Microsoft Entra keeps audit logs for 30 days on P1 and P2. Archive them before they expire. Separately, keep your own daily export of access requests, with approver and reason. The five checks below run on that export. They catch a missing day, rows that did not load, repeats, and decisions with no approver or reason.

## What you will need

- A Microsoft Entra tenant, and an account with the Security Administrator role to create a diagnostic setting (Microsoft Learn).
- A destination that already exists: a storage account, a Log Analytics workspace or an event hub.
- A daily job of your own that writes one file of access requests per day and records how many rows it wrote.
- Python 3 and the `duckdb` module (`pip install duckdb`).

## Step 1. Send the logs somewhere that keeps them

Entra keeps audit logs for 30 days on Microsoft Entra ID P1 and P2 and for seven days on Free (Microsoft Learn: Microsoft Entra data retention). Retention changes are not retroactive: data that has already expired can't be recovered unless it was archived first, so the copy has to start before you need it.

A diagnostic setting sends the logs to a Log Analytics workspace, an event hub or a storage account. The destination must exist first, and logs can take up to three days to start appearing there (Microsoft Learn: Configure Microsoft Entra diagnostic settings). The retention page also notes that tenants with certain Microsoft 365 E5 licenses can use Microsoft Purview Audit (Premium) as an alternative. That archive covers the audit log. The checks below run on a second export, your own daily file of access requests (Step 2). The 30 days is Microsoft's audit-log retention. Microsoft's resource page for access requests states no retention period, and we did not test one.

## Step 2. Shape what you keep into two tables

The request table follows the Graph fields of accessPackageAssignmentRequest and approvalStage.

| Column | Graph field it follows |
|---|---|
| `request_id` | accessPackageAssignmentRequest id |
| `created_at` | createdDateTime |
| `state` | state |
| `reviewed_by` | approvalStage reviewedBy, its id only (all zeros if nobody reviewed) |
| `reviewed_at` | reviewedDateTime |
| `approver_justification` | justification on the approval stage |

The manifest has `export_day`, the day one export file covers, and `record_count`, the rows that export says it holds. Take the count from the source side (the query that built the file), not by counting the file afterwards, or a short file will match a short manifest.

The approval fields come from the approval stage, so your `REQUESTS_SQL` must return one row per request. We did not test requests with more than one approval stage; a join that repeats a request id will trip the `duplicates` check.

The audit log copy from Step 1 is a second source with its own fields, such as directoryAudit `activityDateTime`, `result` and `correlationId`. The script as shipped checks the request table only. The Graph resource page for the requests gives no retention period for them, and we did not test how far back a live tenant returns them.

## Step 3. Run five checks, each one plain SQL

| Check | It fails when | Planted fault that trips it |
|---|---|---|
| `depth` | the oldest export is newer than REQUIRED_DAYS before AS_OF | exports before 2026-08-20 removed |
| `gap` | a day from the oldest export to yesterday has no export, or a manifest row has a NULL | the export for 2026-09-15 removed |
| `reconcile` | rows loaded for a day differ from the count that day's export says it holds | two rows missing from the 2026-10-02 export |
| `duplicates` | a request id is repeated or missing | one request exported twice |
| `evidence` | a delivered or denied request has no approver, no reason, or a decision dated before the request | one all-zero approver, one blank reason, one decision dated before its request |

The gap check uses `NOT EXISTS` and not `NOT IN`, because a single NULL day in the manifest would make `NOT IN` match nothing and hide every gap.

```
SELECT CAST(d AS DATE)
FROM generate_series(CAST(? AS DATE), CAST(? AS DATE), INTERVAL 1 DAY) t(d)
WHERE NOT EXISTS (SELECT 1 FROM manifest m WHERE m.export_day = CAST(d AS DATE))
```

The evidence check answers “who approved, and why”. A NULL approver or reason counts as missing, and so does an approver that is not a GUID. A state written `Delivered` or `DENIED` still counts as decided, and if no request is in a decided state the check fails rather than passing on nothing.

```
SELECT count(*),
  sum(CASE WHEN reviewed_by IS NULL OR lower(trim(reviewed_by)) = '00000000-0000-0000-0000-
000000000000'
           OR NOT regexp_full_match(lower(trim(reviewed_by)), '[0-9a-f]{8}(-[0-9a-f]{4}){3}-[0-9a-
f]{12}')
           THEN 1 ELSE 0 END) AS no_approver,
  sum(CASE WHEN trim(coalesce(approver_justification, '')) = '' THEN 1 ELSE 0 END) AS no_reason,
  sum(CASE WHEN reviewed_at IS NULL OR created_at IS NULL OR reviewed_at < created_at
           THEN 1 ELSE 0 END) AS bad_date
FROM requests WHERE lower(trim(state)) IN ('delivered', 'denied')
```

## The code

The complete download, `entra_request_history_check.py`, also at https://www.datameaning.com/guides/keep-entra-access-request-history-past-30-days/entra_request_history_check.py.

```
#!/usr/bin/env python3
"""Check that exported Microsoft Entra access request history leaves no gap, and that every
decision still says who made it and why.

    python3 entra_request_history_check.py                  # clean run on generated demo records
    python3 entra_request_history_check.py --plant-error    # the same records with five faults
planted
    python3 entra_request_history_check.py --transcript run.txt   # also save what is printed
    python3 entra_request_history_check.py --exit-zero      # report failures but do not stop the
job

The demo records are GENERATED here with a fixed seed. Their column names follow Microsoft Graph's
accessPackageAssignmentRequest and approvalStage. They are not a measurement and not a run on a
tenant. To use your own export, change only the CONFIG block (see the page).

Needs: pip install duckdb
"""
import argparse
import datetime as dt
import random
import sys
import uuid

import duckdb

# ---------------------------------------------------------------------------------------------
# CONFIG: the only part a reader changes.
# DATABASE is ":memory:" for the generated demo, or the path of your own DuckDB file (opened read-
only).
# REQUESTS_SQL returns one row per exported request with exactly these columns:
#   request_id, created_at, state, reviewed_by, reviewed_at, approver_justification
# MANIFEST_SQL returns one row per export you received, with exactly these columns:
#   export_day (the calendar day the file covers), record_count (rows the export says it holds)
DATABASE = ":memory:"
REQUESTS_SQL = "SELECT * FROM demo_requests"
MANIFEST_SQL = "SELECT * FROM demo_manifest"
AS_OF = "2026-11-04"        # the day you run the check; use str(datetime.date.today()) for live
use
REQUIRED_DAYS = 90          # how far back the history must reach
DECIDED_STATES = ("delivered", "denied")   # states that went through an approver
# END CONFIG
# ---------------------------------------------------------------------------------------------

ZERO_GUID = "00000000-0000-0000-0000-000000000000"   # reviewedBy when nobody reviewed (Graph
approvalStage)
GUID = "[0-9a-f]{8}(-[0-9a-f]{4}){3}-[0-9a-f]{12}"   # reviewedBy.id; anything else names no
approver


def make_demo(con, plant):
    """Generated records, seed 134: about 95 daily exports, two to five requests a day."""
    rng = random.Random(134)
    first, last = dt.date(2026, 8, 1), dt.date(2026, 11, 3)
    approvers = [str(uuid.UUID(int=rng.getrandbits(128), version=4)) for _ in range(4)]
    reasons = ["Needed for the quarterly close", "Covers the analytics project", "Role change
approved by manager",
               "Contractor onboarding", "Temporary access for an audit sample"]
```

```
    reqs, manifest = [], []
    day = first
    while day <= last:
        n = rng.randint(2, 5)
        manifest.append([day, n])
        for _ in range(n):
            created = dt.datetime.combine(day, dt.time()) + dt.timedelta(seconds=rng.randint(0,
86399))
            reviewed = created + dt.timedelta(minutes=rng.randint(5, 2880))
            state = "delivered" if rng.random() < 0.85 else "denied"
            reqs.append([str(uuid.UUID(int=rng.getrandbits(128), version=4)), created, state,
                         rng.choice(approvers), reviewed, rng.choice(reasons)])
        day += dt.timedelta(days=1)
    planted = []
    if plant:
        def on(d):
            return [r for r in reqs if r[1].date() == d]
        # 1 depth: drop the oldest 19 days (export and requests), so history stops short of 90 days
        cut = dt.date(2026, 8, 20)
        reqs = [r for r in reqs if r[1].date() >= cut]
        manifest = [m for m in manifest if m[0] >= cut]
        planted.append(f"depth: exports before {cut} removed")
        # 2 gap: one whole day's export missing
        gap = dt.date(2026, 9, 15)
        reqs = [r for r in reqs if r[1].date() != gap]
        manifest = [m for m in manifest if m[0] != gap]
        planted.append(f"gap: the export for {gap} removed")
        # 3 reconcile: a file that holds fewer rows than its manifest says
        short = dt.date(2026, 10, 2)
        victims = on(short)[:2]
        if len(victims) < 2:
            raise SystemExit("could not plant the truncated export")
        reqs = [r for r in reqs if r not in victims]
        planted.append(f"reconcile: two rows missing from the export for {short}")
        # 4 duplicates: one request exported twice
        twin = on(dt.date(2026, 10, 12))[:1]
        if not twin:
            raise SystemExit("could not plant the duplicate")
        reqs.append(list(twin[0]))
        planted.append("duplicates: one request id appears twice")
        # 5 evidence: three decisions that lost their approver, their reason or their order in time
        target = [next((r for r in on(dt.date(2026, 10, d)) if r[2] == "delivered"), None) for d in
(20, 21, 22)]
        if None in target:
            raise SystemExit("could not plant the decision faults")
        target[0][3] = ZERO_GUID
        target[1][5] = "   "
        target[2][4] = target[2][1] - dt.timedelta(hours=1)
        planted.append("evidence: one approver is the all-zero id, one reason is blank, one
decision is dated before its request")
    con.execute("CREATE TABLE demo_requests(request_id VARCHAR, created_at TIMESTAMP, state
VARCHAR, "
                "reviewed_by VARCHAR, reviewed_at TIMESTAMP, approver_justification VARCHAR)")
    con.executemany("INSERT INTO demo_requests VALUES (?,?,?,?,?,?)", reqs)
    con.execute("CREATE TABLE demo_manifest(export_day DATE, record_count INTEGER)")
    con.executemany("INSERT INTO demo_manifest VALUES (?,?)", manifest)
    return planted
```

```
def run_checks(con):
    """Five checks, each plain SQL over the two tables. Returns [(name, ok, message)]."""
    con.execute(f"CREATE TEMP TABLE requests AS SELECT * FROM ({REQUESTS_SQL})")
    con.execute(f"CREATE TEMP TABLE manifest AS SELECT * FROM ({MANIFEST_SQL})")
    as_of = dt.date.fromisoformat(AS_OF)
    decided = (DECIDED_STATES,) if isinstance(DECIDED_STATES, str) else DECIDED_STATES
    states = ", ".join("'" + s.strip().lower().replace("'", "''") + "'" for s in decided)
    out = []

    # 1 depth: the oldest export reaches back far enough
    first = con.execute("SELECT CAST(min(export_day) AS DATE) FROM manifest").fetchone()[0]
    need = as_of - dt.timedelta(days=REQUIRED_DAYS)
    ok = first is not None and first <= need
    out.append(("depth", ok, f"oldest export {first}, needed {need} or earlier ({REQUIRED_DAYS}
days before {AS_OF})"))

    # 2 gap: every day from the oldest export to yesterday has one (NOT EXISTS, so a NULL day hides
nothing)
    nulls = con.execute("SELECT count(*) FROM manifest WHERE export_day IS NULL OR record_count IS
NULL").fetchone()[0]
    missing = []
    if first is not None:
        missing = [r[0] for r in con.execute(
            "SELECT CAST(d AS DATE) FROM generate_series(CAST(? AS DATE), CAST(? AS DATE), INTERVAL
1 DAY) t(d) "
            "WHERE NOT EXISTS (SELECT 1 FROM manifest m WHERE m.export_day = CAST(d AS DATE)) ORDER
BY 1",
            [first, as_of - dt.timedelta(days=1)]).fetchall()]
    out.append(("gap", not missing and not nulls,
                f"{len(missing)} day(s) with no export" + (f": {', '.join(map(str, missing[:5]))}"
if missing else "")
                + f"; {nulls} manifest row(s) with a NULL"))

    # 3 reconcile: rows loaded per day equal the count each export says it holds
    bad = con.execute(
        "WITH l AS (SELECT CAST(created_at AS DATE) d, count(DISTINCT request_id) n FROM requests
GROUP BY 1) "
        "SELECT coalesce(m.export_day, l.d), m.record_count, coalesce(l.n, 0) FROM manifest m "
        "FULL OUTER JOIN l ON l.d = m.export_day "
        "WHERE m.record_count IS DISTINCT FROM coalesce(l.n, 0) ORDER BY 1 NULLS FIRST").fetchall()
    out.append(("reconcile", not bad, f"{len(bad)} day(s) where loaded rows differ from the
export's count"
                + ("".join(f"; {r[0]}: export says {r[1]}, loaded {r[2]}" for r in bad[:3]))))

    # 4 duplicates: every request id present and used once
    dup = con.execute("SELECT count(*) FROM (SELECT request_id FROM requests WHERE request_id IS
NOT NULL "
                      "GROUP BY 1 HAVING count(*) > 1)").fetchone()[0]
    nokey = con.execute("SELECT count(*) FROM requests WHERE request_id IS NULL").fetchone()[0]
    out.append(("duplicates", not dup and not nokey, f"{dup} request id(s) repeated; {nokey} row(s)
with no id"))

    # 5 evidence: each decided request names its approver, gives a reason, and is dated after the
request
    r = con.execute(
        f"SELECT count(*), "
        f"sum(CASE WHEN reviewed_by IS NULL OR lower(trim(reviewed_by)) = '{ZERO_GUID}' "
```

```
        f"OR NOT regexp_full_match(lower(trim(reviewed_by)), '{GUID}') THEN 1 ELSE 0 END), "
        f"sum(CASE WHEN trim(coalesce(approver_justification, '')) = '' THEN 1 ELSE 0 END), "
        f"sum(CASE WHEN reviewed_at IS NULL OR created_at IS NULL OR reviewed_at < created_at THEN
1 ELSE 0 END) "
        f"FROM requests WHERE lower(trim(state)) IN ({states})").fetchone()
    total, no_who, no_why, no_when = r[0], r[1] or 0, r[2] or 0, r[3] or 0
    out.append(("evidence", total > 0 and not (no_who or no_why or no_when),
                f"{total} decided request(s): {no_who} without an approver, {no_why} without a
reason, "
                f"{no_when} dated before the request or not dated" + ("; none to check, see
DECIDED_STATES" if not total else "")))
    return out


def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--plant-error", action="store_true", help="plant five faults in the demo
records")
    ap.add_argument("--transcript", help="also write what is printed to this file")
    ap.add_argument("--exit-zero", action="store_true", help="report failed checks but exit 0 (for
a job that must not stop)")
    a = ap.parse_args()
    lines = []

    def say(s=""):
        print(s)
        lines.append(s)

    demo = DATABASE == ":memory:"
    if a.plant_error and not demo:
        raise SystemExit("--plant-error plants faults in the generated demo records only")
    con = duckdb.connect(DATABASE, read_only=not demo)
    con.execute("SET TimeZone = 'UTC'")   # a TIMESTAMPTZ created_at falls on its UTC day, the day
Graph's export covers
    planted = make_demo(con, a.plant_error) if demo else []
    say(f"duckdb {duckdb.__version__}; python {sys.version.split()[0]}; as of {AS_OF}; history
required: {REQUIRED_DAYS} days")
    if demo:
        n_req = con.execute("SELECT count(*) FROM demo_requests").fetchone()[0]
        n_man = con.execute("SELECT count(*) FROM demo_manifest").fetchone()[0]
        say(f"generated records (seed 134): {n_req} requests, {n_man} daily exports")
    else:
        say(f"your records: database {DATABASE}")
    for p in planted:
        say(f"planted -> {p}")
    results = run_checks(con)
    say()
    for name, ok, msg in results:
        say(f"{'PASS' if ok else 'FAIL'}  {name:<10} {msg}")
    failed = [n for n, ok, _ in results if not ok]
    say()
    if a.plant_error:
        caught = len(failed) == len(planted) == 5
        say(f"caught {len(failed)} of {len(planted)} planted errors: {', '.join(failed) or
'none'}")
        code = 0 if caught else 1
    else:
        say("every check passed" if not failed else f"{len(failed)} check(s) failed: {',
```

```
'.join(failed)}")
        code = 1 if failed and not a.exit_zero else 0
    if a.transcript:
        with open(a.transcript, "w") as f:
            f.write("\n".join(lines) + "\n")
    return code


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

## The runs

### Run 1: clean records

`python3 guides/keep-entra-access-request-history-past-30-days/entra_request_history_check.py`

```
duckdb 1.5.6; python 3.12.15; as of 2026-11-04; history required: 90 days
generated records (seed 134): 332 requests, 95 daily exports

PASS  depth      oldest export 2026-08-01, needed 2026-08-06 or earlier (90 days before 2026-11-04)
PASS  gap        0 day(s) with no export; 0 manifest row(s) with a NULL
PASS  reconcile  0 day(s) where loaded rows differ from the export's count
PASS  duplicates 0 request id(s) repeated; 0 row(s) with no id
PASS  evidence   332 decided request(s): 0 without an approver, 0 without a reason, 0 dated before
the request or not dated

every check passed
```

### Run 2: five faults planted

`python3 guides/keep-entra-access-request-history-past-30-days/entra_request_history_check.py --plant-error`

```
duckdb 1.5.6; python 3.12.15; as of 2026-11-04; history required: 90 days
generated records (seed 134): 270 requests, 75 daily exports
planted -> depth: exports before 2026-08-20 removed
planted -> gap: the export for 2026-09-15 removed
planted -> reconcile: two rows missing from the export for 2026-10-02
planted -> duplicates: one request id appears twice
planted -> evidence: one approver is the all-zero id, one reason is blank, one decision is dated
before its request

FAIL  depth      oldest export 2026-08-20, needed 2026-08-06 or earlier (90 days before 2026-11-04)
FAIL  gap        1 day(s) with no export: 2026-09-15; 0 manifest row(s) with a NULL
FAIL  reconcile  1 day(s) where loaded rows differ from the export's count; 2026-10-02: export says
5, loaded 3
FAIL  duplicates 1 request id(s) repeated; 0 row(s) with no id
FAIL  evidence   270 decided request(s): 1 without an approver, 1 without a reason, 1 dated before
the request or not dated
```

```
caught 5 of 5 planted errors: depth, gap, reconcile, duplicates, evidence
```

## How to read the results

- **gap:** the line names each day with no export. Re-pull that day from the source, reload it and run the check again.
- **reconcile:** the line names the day and gives both counts, what the export says and what loaded. A file cut short shows here.
- **duplicates:** a repeated request id is exported twice; a row with no id cannot be matched to anything.
- **evidence:** the three counts are decisions without an approver, without a reason, and dated before the request. These cannot be repaired from the copy; they go back to whoever owns that access.
- **depth:** the history does not reach back as far as your auditor needs. Audit-log days that expired before the archive started can't be recovered from Entra. Check whether your request source can still return older requests before you treat the shortfall as permanent.

To run it on your own data: load your request rows and your list of export files into a DuckDB file. In the CONFIG block, set DATABASE to the path of that file, set REQUESTS_SQL to a SELECT that returns request_id, created_at, state, reviewed_by, reviewed_at and approver_justification under those exact names, with one row per request, set MANIFEST_SQL to a SELECT that returns export_day and record_count, set AS_OF to the date you run it, set REQUIRED_DAYS to the number of days your auditor needs, and check DECIDED_STATES (lowercase) against the states in your export. Change nothing else in the file.

Write `AS_OF` like `"2026-11-04"`. The file opens your database read-only. To stop a scheduled job from failing when a check does, run the file with `--exit-zero`. If an access package policy needs no approval, its delivered requests may have no approver and the evidence check will flag them. Check how your tenant reports them, then leave them out with a `WHERE` clause in `REQUESTS_SQL`. We did not test how a tenant reports them. The script reads `created_at` in UTC, the time zone Graph uses for `createdDateTime`, so the reconcile check counts each request on its UTC day. Set `export_day` to the UTC day each export file covers.

## Checklist

| Done | Item |
|---|---|
| ☐ | A diagnostic setting sends the logs to a destination you control. |

| Done | Item |
|---|---|
| ☐ | A daily job writes one request file per day and records the rows it wrote, counted at the source. |
| ☐ | `REQUESTS_SQL`, `MANIFEST_SQL`, `DATABASE`, `AS_OF` and `REQUIRED_DAYS` are set. |
| ☐ | The depth check passes for the number of days your auditor needs. |
| ☐ | The gap and reconcile checks pass, or each named day has been re-pulled. |
| ☐ | The duplicates check passes. |
| ☐ | The evidence check passes, or each decision it names is back with its owner. |
| ☐ | Someone owns who can change or delete the copy, and how long it is kept. |

## Limits

- **This is not a tenant result.** The records are generated, with a fixed seed, and use Microsoft's published column names. The runs show that the checks catch five kinds of fault; they do not show what any tenant's history looks like.
- **The checks test completeness and shape, not truth.** They cannot tell whether the approver was the right person or the reason was honest.
- **The manifest is only as good as the job that writes it.**
- **We did not run Entra for this guide.** Retention figures come from the linked Microsoft pages and can change.
- **Tested on:** DuckDB 1.5.6, Python 3.12.15, open-source DuckDB in memory, generated records.

## Sources


- Microsoft Entra data retention, read 2026-11-04
- Configure Microsoft Entra diagnostic settings, read 2026-11-04
- accessPackageAssignmentRequest resource type, read 2026-11-04
- approvalStage resource type, read 2026-11-04
- directoryAudit resource type, read 2026-11-04
- DuckDB documentation, read 2026-11-04

## Keep who agreed, and why, with each decision

This guide keeps a copy of your access request history and checks it for gaps. It can show that a reason is missing. It can't write one that nobody recorded when the request was decided.

In Data Access Concierge, the requester writes why they need access. The owner, or a stand-in, decides on one screen. The yes is on record with a name and a time. Access ends on the agreed date. We set it up in your systems and run it for you.

See Data Access Concierge