

CONTROL TOWER FOR CLAUDE · GUIDE

Colorado's AI law (SB 26-189) and Claude at work: what it means

What the signed act (Session Laws 2026, Ch. 131) asks of deployers of automated decision-making technology, and a script that checks the notices, requests and records you keep against its dates.

Published 2026-10-31 · Online version and downloads: www.datameaning.com/guides/colorado-sb-26-189-and-claude-at-work/

Colorado's SB 26-189, signed May 14, 2026, repeals and replaces the earlier Colorado AI Act. It takes effect January 1, 2027 and applies to consequential decisions made on or after that date, employment decisions among them. A deployer that uses automated decision-making technology to materially influence such a decision must give notice before it is used, a plain-language description within 30 days after an adverse decision, a way to request personal data, correction and human review, and keep records for at least three years. We read the signed act. Legal must say whether HR use of Claude falls inside.

What the act says

Colorado's legislature passed SB 26-189, "Automated Decision-Making Technology", and the Governor signed it on May 14, 2026 (Colorado Session Laws 2026, Chapter 131). It repeals and replaces Colorado's earlier AI Act, SB 24-205. The act sets duties for a deployer that uses a "covered ADMT": automated decision-making technology "used to materially influence a consequential decision" (6-1-1701(5)), where materially influence means an output that is "a non-de minimis factor" in the decision and "affects the outcome", and not "incidental, trivial, or clerical uses" (6-1-1701(13)). Whether an HR decision made with Claude is one is for Legal to say; this guide treats HR uses as the ones to look at first (SB 26-189, Session Laws 2026, Ch. 131 (read 2026-10-31); [the legislature's bill page](#)). The table quotes each deployer duty from the act, and the record the script reads to check it.

The deployer duties in the act, and what the script reads

What the act says	The record the script reads	What it prints
"Prior to" using the technology in the decision, "a clear and conspicuous notice" to the consumer (6-1-1704(1)); a "prominent public notice" at points of consumer interaction meets this (6-1-1704(2))	Decision log: the date a notice was sent; or the day you began keeping a public notice	A decision with no notice recorded and no public notice in place, or a notice dated after the decision
After an adverse outcome, "within thirty days after making the decision", a "plain language description of the consequential decision and the role the covered ADMT played" (6-1-1704(3))	Decision log: the outcome, and the date the description was sent	An adverse decision with no description after 30 days, or one sent later
After an adverse outcome, on request: instructions for requesting personal data and correcting "factually incorrect or materially inaccurate" data, "consistent with section 6-1-1306" (6-1-1705(1)(a)(I))	Request log: the kind, the date received, the date answered	A request not answered, or answered after a deadline you set
After an adverse outcome, on request: "an opportunity for meaningful human review and reconsideration", "to the extent commercially reasonable" (6-1-1705(1)(a)(II))	Request log, if you record review requests there: the same fields	The same list
Records "reasonably necessary to demonstrate compliance", kept "for not less than three years after the date of a consequential decision" (6-1-1703)	Decision log: dates and id numbers	A month with no record, and missing id numbers

The act says records "may include" version identifiers, changelogs and documentation of material mitigation changes (6-1-1703); it does not list them all. Counting decision records by month, as the script does, is our proxy and not the legal duty. 6-1-1705 states no deadline for answering a request; it points to 6-1-1306, which Legal should read for response timing, so the script has no default deadline: you set one, with your counsel.

Dates and enforcement, from the act. The signed act takes effect January 1, 2027, and applies to consequential decisions made on or after that date (Section 5). The attorney general's rules on the description and on consumer rights are due on or before the same day (6-1-1704(4)(b), 6-1-1705(3)), and may change what a description must say. The attorney general enforces the act exclusively, and a violation is a "deceptive trade practice" under the Colorado Consumer Protection Act (6-1-1706(1)(b), (2)(a)); the act creates no new private right of action (6-1-1706(4)). Before an enforcement action the attorney general issues a notice of violation if a cure is deemed possible, and may sue if it is not cured within sixty days; that subsection is repealed January 1, 2030 (6-1-1706(3)).

What this covers, and what it does not

This guide does not tell you whether SB 26-189 applies to any use of Claude at your company. That is a legal question for your lawyers. We read the signed act, but the script does not test whether a technology "materially influences" a decision, who counts as a consumer, who is a deployer, or the act's exceptions for entities that meet other legal obligations (6-1-1708). It sorts your own record of each use so Legal sees which to read first, and it checks the notices, requests and records you keep against the act's dates. If you are also looking at New York City's rule on hiring tools, our [guide on NYC Local Law 144 and Claude in hiring](#) does the equivalent there.

This is not legal advice and not a compliance test. A line in the output says what a log records against the 30 days or the deadline you set. It does not say the act was or was not followed.

What you will need

- A register of your Claude uses in HR as a CSV with a header row: an id, whether the use supports an employment decision (hiring, promotion, pay, discipline, termination, another employment decision) or neither, whether it touches people in Colorado as your counsel defines that, and the date the row was created.
- A log of employment decisions where Claude's output was used, as a CSV with a header row: a decision id, the register id of the use, the decision date, whether the outcome was adverse, the date a notice was sent, the date a plain-language description was sent, and the date the record was created. These are records about people: keep them under your own access controls. The script runs on your machine and sends nothing to Data Meaning or anyone else.
- A log of requests for personal data, correction or human review as a CSV with a header row: an id, the kind, the date received, the date answered and the date the record was created.
- Python 3 and the duckdb module. The script reads each CSV with DuckDB's `read_csv` using a header row and `all_varchar` ([DuckDB: CSV overview](#) (read 2026-10-31)).

Step 1. Set the settings that have no default

Three settings decide what the checks mean, and each has no default. `REQUEST_DAYS` is the number of days you allow for answering a request: 6-1-1705 states none, so counsel sets it. `FIRST_USE_ON` is the day you began using Claude in employment decisions. `IDS_ARE_SEQUENTIAL` says whether your ids end in a number that goes up by one: `True` checks for missing numbers, `False` (ids from a shared counter, or not numbered) skips that check. If one of the three is unset, its check fails: a check that could not run is not reported as ok. `AS_OF` is the day the check is for, and `None` means today. `PUBLIC_NOTICE_SINCE` is optional: the day you began keeping a prominent public notice under 6-1-1704(2); decisions on or after it need no notice of their own in the log. The

same block holds the answer lists, which are ours, and you can disagree with them. The demo runs pass values on the command line (`--request-days 45, --first-use-on 2026-01-01, --ids-sequential yes, --as-of 2026-10-31`); 45 days is a number we chose for the demo, not one from the act.

```
# SETTINGS WITH NO DEFAULT: an unset one fails its check, because a check that could not run is not a pass.
REQUEST_DAYS = None          # the number of days you will allow to answer a request. 6-1-1705 states none (it points to 6-1-1306); your counsel sets it.
FIRST_USE_ON = None          # YYYY-MM-DD: the day you began using Claude in employment decisions
AS_OF = None                  # YYYY-MM-DD: the day the check is for; None = today
KNOWN_GAP_MONTHS = []        # months (YYYY-MM) with no decision records that you have explained, for example ["2026-03"]
PUBLIC_NOTICE_SINCE = None    # YYYY-MM-DD: the day you began keeping a prominent public notice at points of consumer interaction.
                                # 6-1-1704(2) says that meets the notice duty; a decision on or after it with no notice of its own is
                                # then not flagged. None: you rely on notices sent per decision, and each must be in the log.
IDS_ARE_SEQUENTIAL = None     # True: ids end in a number that goes up by one (D-001, D-002); a missing number is reported. False: ids come
                                # from a shared counter or are not numbered, so gaps are not checked.
                                # None: not set, and that check fails.
```

Dates must be written YYYY-MM-DD and nothing else. DuckDB's own date cast reads 8/5/26 as the year 8 and accepts 2026/09/24, so the script matches the pattern first and only then parses the date. A date that is not in that form is counted and fails an input check. It is never dropped.

Step 2. Sort each use, as recorded

The sort takes the first status that applies: a decision-type answer that is empty or unknown goes to Legal; a use that supports no employment decision is set aside; a Colorado answer that is empty or unknown goes to Legal; a use recorded as touching no one in Colorado is set aside; any other use goes to Legal. The statuses say what the register records. They do not say what the act requires.

```
CASE WHEN d.meaning IS NULL THEN 'FOR LEGAL TO CHECK'
      WHEN d.meaning = 'neither' THEN 'NOT AN EMPLOYMENT USE AS RECORDED'
      WHEN c.meaning IS NULL THEN 'FOR LEGAL TO CHECK'
      WHEN c.meaning = 'no' THEN 'NO COLORADO PEOPLE AS RECORDED'
      ELSE 'FOR LEGAL TO CHECK' END AS status,
```

Step 3. Check the notice and the 30-day description

For each decision the script finds the due date, the decision date plus 30 days, and compares the log with it. The act counts the 30 days from the decision ("within thirty days after making the decision"); the script uses the decision date. An adverse decision with no description is a hard

finding once the as-of date is past the due date; before that it is shown as open, not as a fault. The act requires the notice before the technology is used in the decision (6-1-1704(1)), so a notice dated after the decision is a follow-up item, printed as NOTICE DATED n DAYS AFTER THE DECISION; no demo run plants this fault. A decision with no notice of its own is a finding unless you set PUBLIC_NOTICE_SINCE and the decision is on or after it. A blank description is a finding. An unreadable date is never guessed.

```
hard, soft = [], []
if outcome is None:
    hard.append("outcome not recorded or not recognized")
if decided is None:
    hard.append("decision date not read")
else:
    if not n_given and public_since is not None and decided >= public_since:
        pass # 6-1-1704(2): the public notice you keep
meets the duty
    elif not n_given:
        hard.append("NO NOTICE RECORDED")
    elif notice is None:
        hard.append("notice date not read")
    elif notice > decided:
        hard.append(f"NOTICE DATED {(notice - decided).days} DAYS AFTER THE DECISION")
    if outcome == "adverse":
        due = decided + datetime.timedelta(days=DESCRIPTION_DAYS)
        if not s_given:
            (hard if as_of > due else soft).append(
                f"NO DESCRIPTION, {DESCRIPTION_DAYS} DAYS PASSED (due {due})" if as_of >
due else f"description due {due}")
        elif sent is None:
            hard.append("description date not read")
        elif sent > due:
            hard.append(f"DESCRIPTION SENT {(sent - decided).days} DAYS AFTER THE
DECISION")
```

Step 4. Check each request against your deadline

A request with no answer is a finding once the as-of date passes its due date, and open before that. An answer dated after the due date is a finding that gives the days taken. An answer dated before the request was received is an input fault, because the log cannot be right.

Step 5. Look for a gap in the 3 years of records

The window starts at the later of three years before the as-of date and your FIRST_USE_ON, and ends at the as-of date. The script lists every month in it with its count of decision records. A finished month with none is flagged unless you name it in KNOWN_GAP_MONTHS: a month with no record may be real, or may be a log that lost rows, and the script cannot tell which. A month still running on the as-of date, and a part month at the window start, are shown but not counted as gaps. If

IDS_ARE_SEQUENTIAL is True, it also reads each id as a prefix and a number and counts every number missing between the lowest and highest, printing them the way your log writes them (D-025, not d-25). The screen shows the first twelve and the count; `records_kept.csv` lists every one. Both checks look at the whole window, not one period, so one explained old gap does not fail every later check.

```
def id_gaps(ids, limit):
    """ids: id strings as the log writes them. Groups by prefix ignoring case, and returns (count
    of missing numbers,
    up to `limit` missing ids written the way the log writes them: its own prefix and zero
    padding)."""
    by_prefix = {}
    for i in ids:
        m = re.fullmatch(r"(.?)(\d+)", i)
        if m:
            g = by_prefix.setdefault(m.group(1).lower(), {"shown": m.group(1), "width":
len(m.group(2)), "nums": set()})
            g["nums"].add(int(m.group(2)))
    count, listed = 0, []
    for _, g in sorted(by_prefix.items()):
        nums = g["nums"]
        count += max(nums) - min(nums) + 1 - len(nums)
        k = min(nums)
        while len(listed) < limit and k <= max(nums):
            if k not in nums:
                listed.append(f"{g['shown']}{k:0{g['width']}d}")
            k += 1
    return count, listed
```

Step 6. Run the input checks before trusting any list

- **Settings:** the as-of date, the request deadline, the first-use date and the explained months (trip on: a date such as 31/10/2026, no deadline set, no first-use date, or a month written 3/2026; run 3 shows the unset deadline).
- **Ids:** register, decision and request ids listed more than once, ignoring case, and rows with no id (trip on: a duplicated or unnumbered row).
- **Answers in no list:** a decision type or Colorado answer in the register, an outcome in the decision log, a kind in the request log (trip on: selection, denied, appeal; an empty answer counts).
- **A decision that names a use not on the register,** including a blank use id (trips on: a decision that names UC-999).
- **Dates missing or not written YYYY-MM-DD,** and **dates after the as-of date** (trip on: 6/12/26, or a record dated 2027-01-01).

- **A description dated before its decision, or a request answered before it was received** (trips on: a request answered two days before it arrived). Run 2 plants no description dated before its decision, so no run shows that half of the check tripping.

A word in your list that no row uses is printed as information. A failed input check marks every list INCOMPLETE.

The code

The complete download, [colorado_sb26189_claude_check.py](#). Run it as written: `python3 colorado_sb26189_claude_check.py --as-of 2026-10-31 --request-days 45 --first-use-on 2026-01-01 --ids-sequential yes`.

```
#!/usr/bin/env python3
"""Colorado SB 26-189 and Claude at work: the notices, requests and records a company would keep,
and the dates checked, in DuckDB.

    python3 colorado_sb26189_claude_check.py --as-of 2026-10-31 --request-days 45 --first-use-on
2026-01-01 --ids-sequential yes    # clean generated records
    python3 colorado_sb26189_claude_check.py --plant --as-of 2026-10-31 --request-days 45 --first-
use-on 2026-01-01 --ids-sequential yes    # planted faults
    python3 colorado_sb26189_claude_check.py --as-of 2026-10-31 --first-use-on 2026-01-01 --ids-
sequential yes    # no request deadline set: that check fails
    python3 colorado_sb26189_claude_check.py --strict ...    # exit 1 when an input check failed
or a decision, request, record month or                                # missing id number needs follow-up;
FOR LEGAL TO CHECK uses alone do not,                                # they are a prompt for Legal
    python3 colorado_sb26189_claude_check.py --work-dir some/folder    # where generated files and
results are written

Builds four lists, each printed with its filters, row count and a SHA-256 of its rows, and each
written as a CSV:
    use sort        every row of your Claude use-case register: is it an employment use, and does it
touch people in Colorado as
                    you record it. An employment use, or an empty or unknown answer, is FOR LEGAL TO
CHECK (reason: SB 26-189).
                    A use is never called covered and never called exempt.
    decisions       every employment decision where Claude's output was used: is a notice recorded
before the decision
                    (6-1-1704(1)), and for an adverse decision, was a plain-language description sent
within 30 days after
                    making the decision (6-1-1704(3))
    requests        every request for personal data, correction or human review: answered, and within
the deadline YOU set
    records         whether the decision log has a record in every finished month of the window (a
month still running on the
                    as-of date is not counted as a gap), and whether its id numbers have gaps
The window is the later of "3 years before the as-of date" and "the date you began using Claude in
employment decisions".
Input checks run first (unreadable dates, answers in no list, duplicate ids, unset settings); a
failed check marks the lists INCOMPLETE.
```

The act (SB 26-189, Colorado Session Laws 2026, Ch. 131) takes effect January 1, 2027 and applies to consequential decisions made on or after that date (Section 5); decisions dated earlier are counted on an [info] line. This is NOT legal advice and NOT a compliance test.

Legal decides whether the act reaches a use. The act's text is not fetched by this script. The generated records are NOT a real organization's data.

"""

```
import csv
import datetime
import hashlib
import pathlib
import random
import re
import sys
```

```
import duckdb
```

```
# ---- CHANGE THESE FOR YOUR OWN DATA -----
REGISTER_FILE = None      # your Claude use-case register (CSV with a header row); None = generate
DECISIONS_FILE = None     # your log of employment decisions where Claude's output was used (CSV
                           # with a header row); None = generate
REQUESTS_FILE = None      # your log of requests for data, correction or human review (CSV with a
                           # header row); None = generate
# the column of YOUR files that holds each field. Dates must be written YYYY-MM-DD.
REG_COLS = {"use_case_id": "use_case_id", "decision": "decision", "colorado": "colorado_people",
            "created_on": "created_on"}
DEC_COLS = {"decision_id": "decision_id", "use_case_id": "use_case_id", "decided_on": "decided_on",
            "outcome": "outcome",
            "notice_on": "notice_sent_on", "description_on": "description_sent_on", "created_on":
            "created_on"}
REQ_COLS = {"request_id": "request_id", "kind": "kind", "received_on": "received_on",
            "answered_on": "answered_on",
            "created_on": "created_on"}
# YOUR answers (lower case, no spaces at the ends) -> the meaning the script gives them. An answer
# in your files that is in
# none of these lists is counted and fails an input check; it is never guessed.
DECISION_WORDS = {"hiring": "employment", "promotion": "employment", "pay": "employment",
                  "discipline": "employment",
                  "termination": "employment", "other employment decision": "employment",
                  "neither": "neither"}
COLORADO_WORDS = {"yes": "yes", "no": "no"}      # does the use touch people in Colorado, as
your counsel defines it
OUTCOME_WORDS = {"adverse": "adverse", "not adverse": "not adverse"}
KIND_WORDS = {"personal data": "personal data", "correction": "correction", "human review": "human
review"}
# SETTINGS WITH NO DEFAULT: an unset one fails its check, because a check that could not run is not
a pass.
REQUEST_DAYS = None      # the number of days you will allow to answer a request. 6-1-1705 states
none (it points to 6-1-1306); your counsel sets it.
FIRST_USE_ON = None      # YYYY-MM-DD: the day you began using Claude in employment decisions
AS_OF = None             # YYYY-MM-DD: the day the check is for; None = today
KNOWN_GAP_MONTHS = []    # months (YYYY-MM) with no decision records that you have explained, for
example ["2026-03"]
PUBLIC_NOTICE_SINCE = None # YYYY-MM-DD: the day you began keeping a prominent public notice at
points of consumer interaction.
                           # 6-1-1704(2) says that meets the notice duty; a decision on or after
```

```

it with no notice of its own is
    # then not flagged. None: you rely on notices sent per decision, and
each must be in the log.
IDS_ARE_SEQUENTIAL = None # True: ids end in a number that goes up by one (D-001, D-002); a
missing number is reported. False: ids come
    # from a shared counter or are not numbered, so gaps are not checked.
None: not set, and that check fails.
# -----

DESCRIPTION_DAYS = 30 # 6-1-1704(3): "within thirty days after making the decision", a plain-
language description of the decision and the technology's role
RETENTION_YEARS = 3 # 6-1-1703: records kept "for not less than three years after the date of
a consequential decision"
APPLIES_FROM = "2027-01-01" # Section 5(3): the act applies to consequential decisions made on or
after January 1, 2027
MISSING_LISTED = 10000 # missing ids written to records_kept.csv (the console shows the first 12
and the count)
SOURCES = [("SB 26-189, session law (Ch. 131)", "https://leg.colorado.gov/laws/session-laws/SB26-
189/131/download"),
            ("SB 26-189, bill page", "https://leg.colorado.gov/bills/sb26-189")]

def qcol(name):
    return ''' + name.replace('\'', '""') + '''

def strict_date(col):
    """YYYY-MM-DD only: '8/5/26' and '2026/09/24' are not dates here, and neither is '2026-02-
30'."""
    c = f"trim(coalesce({qcol(col)}, ''))"
    return (f"CASE WHEN regexp_full_match({c}, '\\d{{4}}-\\d{{2}}-\\d{{2}}') THEN try_strptime({c},
'Y-m-d')::DATE END",
            f"{c} <> ''")

def years_before(day, years):
    try:
        return day.replace(year=day.year - years)
    except ValueError:
        # 29 February
        return day.replace(year=day.year - years, day=28)

def month_start(day):
    return day.replace(day=1)

def next_month(day):
    return (day.replace(day=28) + datetime.timedelta(days=4)).replace(day=1)

def generate(mode, gen_dir):
    """Seeded stand-in for an organization's records. mode: clean or plant."""
    plant = mode == "plant"
    rnd = random.Random(226)
    gen_dir.mkdir(parents=True, exist_ok=True)
    as_of = datetime.date(2026, 10, 31)
    iso = lambda d: d.isoformat() if d else ""
    reg = [{"UC-201", "Summarize resumes for recruiters", "hiring", "yes"},

```

```

["UC-202", "Rank resumes against a job description", "hiring", "yes"],
["UC-203", "Draft job postings", "neither", "yes"],
["UC-204", "Suggest a promotion shortlist", "promotion", "yes"],
["UC-205", "Answer staff questions about HR policy", "neither", "yes"],
["UC-206", "Draft performance review summaries", "pay", "yes"],
["UC-207", "Screen applicants for a role open only outside Colorado", "hiring", "no"]]
if plant:
    reg += [{"UC-208", "Score interview notes", "hiring", ""},
Colorado not recorded
        ["uc-209", " Flag attendance problems ", "Discipline", " Yes "],
        #
planted: other case and spaces; still read
        ["UC-210", "Pick finalists", "selection", "yes"],
        #
planted: an answer in no list
        ["UC-202", "Rank resumes against a job description", "hiring", "yes"],
        #
planted: duplicate id
        ["", "Chat helper for recruiters", "", ""]]
        #
planted: no id, no answers
    with open(gen_dir / "register.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(["use_case_id", "use_case", "decision", "colorado_people", "created_on"])
        for r in reg:
            w.writerow([r[0], r[1], r[2], r[3], "2026-01-02"])
# 12 decisions in each month from January to October 2026, numbered in date order
decisions, n = [], 0
for month in range(1, 11):
    for k in range(12):
        n += 1
        decided = datetime.date(2026, month, 1) + datetime.timedelta(days=min(27, k * 2 +
rnd.randint(0, 1)))
        adverse = rnd.random() < 0.3
        notice = decided - datetime.timedelta(days=rnd.randint(0, 3))
        desc = decided + datetime.timedelta(days=rnd.randint(3, 25)) if adverse else None
        if desc and desc > as_of:
            desc = None
            # not sent yet; still inside its 30 days in the clean
run
        decisions.append([f"D-{n:03d}", rnd.choice(["UC-201", "UC-202", "UC-204", "UC-206"]),
iso(decided),
            "adverse" if adverse else "not adverse", iso(notice), iso(desc),
iso(decided)])
    if plant:
        def row(did):
            return next(r for r in decisions if r[0] == did)
        row("D-050")[4] = ""
        # planted: no notice recorded
        r = row("D-051"); r[3], r[5] = "adverse", ""
        # planted: adverse, no
description, 30 days long gone
        r = row("D-052"); r[3], r[5] = "adverse", iso(datetime.date.fromisoformat(r[2]) +
datetime.timedelta(days=41))
        # planted: sent on day 41
        row("D-053")[2] = "6/12/26"
        # planted: a date that does not
read as YYYY-MM-DD
        row("D-054")[3] = "denied"
        # planted: an outcome in no list
        row("D-055")[6] = "2027-01-01"
        # planted: a record dated after
the as-of date
        decisions.append(list(row("D-056")))
        # planted: a duplicated decision
id
        decisions = [r for r in decisions if not r[2].startswith("2026-03") and r[0] != "D-090"]
# planted: all of March gone, and one more row
        row("D-101")[1] = "UC-999"
        # planted: a use that is not on
the register

```

```

        decisions.append(["", "UC-201", "2026-06-15", "not adverse", "2026-06-14", "", "2026-06-15"])
    # planted: a decision with no id
    with open(gen_dir / "decisions.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(["decision_id", "use_case_id", "decided_on", "outcome", "notice_sent_on", "description_sent_on", "created_on"])
        w.writerows(decisions)
    requests = []
    for i in range(1, 31):
        received = datetime.date(2026, 2, 1) + datetime.timedelta(days=int(i * 8.5))
        answered = received + datetime.timedelta(days=rnd.randint(5, 40))
        requests.append([f"R-{i:03d}", rnd.choice(["personal data", "correction", "human review"]), iso(received),
                                                                iso(answered) if answered <= as_of else "", iso(received)])
    if plant:
        requests[3][3] = "" #
    planted: R-004, never answered
        requests[5][3] = iso(datetime.date.fromisoformat(requests[5][2]) + datetime.timedelta(days=60)) # planted: answered after 60 days
        requests[7][3] = iso(datetime.date.fromisoformat(requests[7][2]) - datetime.timedelta(days=2)) # planted: answered before it was received
        requests[9][1] = "appeal" #
    planted: a kind in no list
        requests.append(list(requests[10])) #
    planted: a duplicated request id
        requests.append(["", "correction", "2026-05-01", "2026-05-10", "2026-05-01"]) #
    planted: a request with no id
    with open(gen_dir / "requests.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(["request_id", "kind", "received_on", "answered_on", "created_on"])
        w.writerows(requests)
    return str(gen_dir / "register.csv"), str(gen_dir / "decisions.csv"), str(gen_dir / "requests.csv")

def fingerprint(rows):
    body = "\n".join("|".join("" if v is None else str(v) for v in r) for r in rows)
    return hashlib.sha256(body.encode()).hexdigest()

def population(out_dir, name, control, filters, rows, header, complete):
    out_dir.mkdir(parents=True, exist_ok=True)
    with open(out_dir / f"{name}.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(header)
        w.writerows(rows)
    print(f"\n== {control}: {name}" + ("" if complete else " ** INCOMPLETE: see input checks **"))
    print(f"filters : {filters}")
    print(f"rows : {len(rows)}")
    print(f"sha256 : {fingerprint(rows)}")

def flag(argv, name):
    """The value after a --flag, or None; stops with a message if it is missing."""
    if name not in argv:
        return None
    at = argv.index(name) + 1

```

```

if at >= len(argv) or argv[at].startswith("--"):
    sys.exit(f"usage: {name} needs a value after it")
return argv[at]

def read_day(text):
    if text is None or not re.fullmatch(r"\d{4}-\d{2}-\d{2}", text.strip()):
        return None
    try:
        return datetime.date.fromisoformat(text.strip())
    except ValueError:
        return None

def id_gaps(ids, limit):
    """ids: id strings as the log writes them. Groups by prefix ignoring case, and returns (count
    of missing numbers,
    up to `limit` missing ids written the way the log writes them: its own prefix and zero
    padding)."""
    by_prefix = {}
    for i in ids:
        m = re.fullmatch(r"(.*)\d+", i)
        if m:
            g = by_prefix.setdefault(m.group(1).lower(), {"shown": m.group(1), "width":
len(m.group(2)), "nums": set()})
            g["nums"].add(int(m.group(2)))
    count, listed = 0, []
    for _, g in sorted(by_prefix.items()):
        nums = g["nums"]
        count += max(nums) - min(nums) + 1 - len(nums)
        k = min(nums)
        while len(listed) < limit and k <= max(nums):
            if k not in nums:
                listed.append(f"{g['shown']}{k:0{g['width']}d}")
            k += 1
    return count, listed

def main():
    argv = sys.argv
    strict = "--strict" in argv
    mode = "plant" if "--plant" in argv else "clean"
    base = pathlib.Path(flag(argv, "--work-dir") or ".")
    out_dir, gen_dir = base / "evidence-out", base / "generated"
    as_of_text = flag(argv, "--as-of") or AS_OF
    days_text = flag(argv, "--request-days") or REQUEST_DAYS
    first_text = flag(argv, "--first-use-on") or FIRST_USE_ON
    pub_text = flag(argv, "--public-notice-since") or PUBLIC_NOTICE_SINCE
    seq_text = flag(argv, "--ids-sequential")
    ids_seq = {"yes": True, "no": False}.get(seq_text.lower()) if seq_text else IDS_ARE_SEQUENTIAL
    if ids_seq not in (True, False, None):
        sys.exit("IDS_ARE_SEQUENTIAL must be True, False or None (not set)")
    for name, words in [(("DECISION_WORDS", DECISION_WORDS), ("COLORADO_WORDS", COLORADO_WORDS),
("OUTCOME_WORDS", OUTCOME_WORDS),
("KIND_WORDS", KIND_WORDS))]:
        for k in words:
            if k != k.strip().lower():
                sys.exit(f"{name}: the answer {k!r} must be lower case with no spaces at the ends")

```

```

for v in DECISION_WORDS.values():
    if v not in ("employment", "neither"):
        sys.exit(f"DECISION_WORDS: {v!r} must be employment or neither")
for v in COLORADO_WORDS.values():
    if v not in ("yes", "no"):
        sys.exit(f"COLORADO_WORDS: {v!r} must be yes or no")
for v in OUTCOME_WORDS.values():
    if v not in ("adverse", "not adverse"):
        sys.exit(f"OUTCOME_WORDS: {v!r} must be adverse or not adverse")
for v in KIND_WORDS.values():
    if not (isinstance(v, str) and v.strip()):
        sys.exit(f"KIND_WORDS: {v!r} must be a name")
files = [REGISTER_FILE, DECISIONS_FILE, REQUESTS_FILE]
if all(f is None for f in files):
    reg_file, dec_file, req_file = generate(mode, gen_dir)
    print(f"input      : GENERATED records (not a real organization), seed 226, mode={mode}")
elif any(f is None for f in files):
    sys.exit("set REGISTER_FILE, DECISIONS_FILE and REQUESTS_FILE together")
else:
    reg_file, dec_file, req_file = files
    print("input      : " + ", ".join(files))
for label, url in SOURCES:
    print(f"source     : {label}, {url} (not fetched by this script)")

# ---- settings: an unset or unreadable one is a failed check, never a quiet default
as_of = read_day(as_of_text) if as_of_text else datetime.date.today()
if as_of is None:
    sys.exit(f"the as-of date {as_of_text!r} is not written as YYYY-MM-DD, for example 2026-10-31")
first_use = read_day(first_text) if first_text else None
public_since = read_day(pub_text) if pub_text else None
try:
    req_days = int(str(days_text)) if days_text is not None else None
    if req_days is not None and req_days <= 0:
        req_days = None
except ValueError:
    req_days = None
gap_months = [m for m in KNOWN_GAP_MONTHS if re.fullmatch(r"\d{4}-(0[1-9]|1[0-2])", str(m))]
print(f"as of      : {as_of}    request deadline: {str(req_days) + ' days' if req_days else 'NOT SET'}
first use: {first_use or 'NOT SET'}"
      f"    explained gap months: {' , '.join(gap_months) if gap_months else 'none'}")

con = duckdb.connect()
# every column your settings name must exist in your files; a wrong name stops the run with the
names it did find
for label, path, cols in [("register", reg_file, REG_COLS), ("decision log", dec_file,
DEC_COLS), ("request log", req_file, REQ_COLS)]:
    have = {r[0].lower() for r in con.execute("DESCRIBE SELECT * FROM read_csv(?, header=true,
all_varchar=true)", [path]).fetchall()}
    lost = [c for c in cols.values() if c.lower() not in have]
    if lost:
        sys.exit(f"{label}: no column named {' , '.join(lost)} in {path}; its columns are {' , '.join(sorted(have))}")
    for table, words in [("decision_map", DECISION_WORDS), ("colorado_map", COLORADO_WORDS),
("outcome_map", OUTCOME_WORDS),
("kind_map", KIND_WORDS)]:
        con.execute(f"CREATE TABLE {table}(word VARCHAR, meaning VARCHAR)")
        con.executemany(f"INSERT INTO {table} VALUES (?, ?)", [(k, str(v)) for k, v in

```

```

words.items()))
word = lambda c: f"lower(trim(coalesce({qcol(c)}, '')))"
ident = lambda c: f"nullif(lower(trim(coalesce({qcol(c)}, ''))), '')"
shown = lambda c: f"nullif(trim(coalesce({qcol(c)}, '')), '')"

def dates(cols, names):
    parts = []
    for n in names:
        parsed, given = strict_date(cols[n])
        parts.append(f"{parsed} AS {n}, {given} AS {n}_given")
    return ", ".join(parts)

con.execute(f"""CREATE TABLE register AS SELECT row_number() OVER () AS row_no,
{ident(REG_COLS['use_case_id'])} AS use_case_id,
    {shown(REG_COLS['use_case_id'])} AS id_shown, {word(REG_COLS['decision'])} AS
decision_word,
    {word(REG_COLS['colorado'])} AS colorado_word, {dates(REG_COLS, ['created_on'])}
FROM read_csv(?, header=true, all_varchar=true)""", [reg_file])
con.execute(f"""CREATE TABLE decisions AS SELECT row_number() OVER () AS row_no,
{ident(DEC_COLS['decision_id'])} AS decision_id,
    {shown(DEC_COLS['decision_id'])} AS id_shown, {ident(DEC_COLS['use_case_id'])} AS
use_case_id,
    {word(DEC_COLS['outcome'])} AS outcome_word,
    {dates(DEC_COLS, ['decided_on', 'notice_on', 'description_on', 'created_on'])}
FROM read_csv(?, header=true, all_varchar=true)""", [dec_file])
con.execute(f"""CREATE TABLE requests AS SELECT row_number() OVER () AS row_no,
{ident(REQ_COLS['request_id'])} AS request_id,
    {shown(REQ_COLS['request_id'])} AS id_shown, {word(REQ_COLS['kind'])} AS kind_word,
    {dates(REQ_COLS, ['received_on', 'answered_on', 'created_on'])}
FROM read_csv(?, header=true, all_varchar=true)""", [req_file])
q = lambda s, *p: con.execute(s, list(p)).fetchone()[0]
n_reg, n_dec, n_req = (q(f"SELECT count(*) FROM {t}") for t in ("register", "decisions",
"requests"))

# ---- input checks: each says which planted fault trips it
unread = lambda tbl, col, m: q(f"SELECT count(*) FROM {tbl} WHERE {col} NOT IN (SELECT word
FROM {m})")
dup = lambda tbl, col: q(f"SELECT count(*) FROM (SELECT {col} FROM {tbl} WHERE {col} IS NOT
NULL GROUP BY 1 HAVING count(*) > 1)")
bad = lambda tbl, c: f"({c}_given AND {c} IS NULL)"
after = lambda tbl, c: q(f"SELECT count(*) FROM {tbl} WHERE {c} > ?::DATE", str(as_of))
unreadable = (q(f"SELECT count(*) FROM decisions WHERE {bad('decisions', 'decided_on')} OR NOT
decided_on_given OR {bad('decisions', 'notice_on')}
    f" OR {bad('decisions', 'description_on')} OR {bad('decisions', 'created_on')}
OR NOT created_on_given")
    + q(f"SELECT count(*) FROM requests WHERE {bad('requests', 'received_on')} OR NOT
received_on_given"
        f" OR {bad('requests', 'answered_on')} OR {bad('requests', 'created_on')} OR
NOT created_on_given")
    + q(f"SELECT count(*) FROM register WHERE {bad('register', 'created_on')} OR NOT
created_on_given"))
dated_after = (sum(after("decisions", c) for c in ("decided_on", "notice_on", "description_on",
"created_on"))
    + sum(after("requests", c) for c in ("received_on", "answered_on",
"created_on")) + after("register", "created_on"))
checks = [
    ("as-of date not written as YYYY-MM-DD", 0 if as_of else 1, "an as-of setting such as
31/10/2026"),

```

```

        ("request deadline not set, or not a whole number of days above zero", 0 if req_days else
1,
        "no --request-days and REQUEST_DAYS left as None; 6-1-1705 states no deadline, so you set
one"),
        ("ids-are-sequential setting not set (True or False)", 0 if ids_seq in (True, False) else
1,
        "IDS_ARE_SEQUENTIAL left as None"),
        ("public-notice date not written as YYYY-MM-DD", 1 if pub_text and public_since is None
else 0,
        "a PUBLIC_NOTICE_SINCE such as 1/1/27"),
        ("first-use date not set, or not written as YYYY-MM-DD", 0 if first_use else 1,
"FIRST_USE_ON left as None"),
        ("explained gap months not written as YYYY-MM", len(KNOWN_GAP_MONTHS) - len(gap_months), "a
KNOWN_GAP_MONTHS entry such as 3/2026"),
        ("register ids listed more than once (ignoring case)", dup("register", "use_case_id"), "a
duplicated register row"),
        ("register rows with no use case id", q("SELECT count(*) FROM register WHERE use_case_id IS
NULL"), "a register row with no id"),
        ("register answers in no list (decision, Colorado)", unread("register", "decision_word",
"decision_map") + unread("register", "colorado_word", "colorado_map"),
        "an answer such as 'selection' that is not one of your words (an empty answer counts)"),
        ("decision ids listed more than once (ignoring case)", dup("decisions", "decision_id"), "a
duplicated decision row"),
        ("decision rows with no decision id", q("SELECT count(*) FROM decisions WHERE decision_id
IS NULL"), "a decision row with no id"),
        ("decisions naming a use that is not on the register", q("SELECT count(*) FROM decisions
WHERE use_case_id IS NULL OR use_case_id NOT IN (SELECT use_case_id FROM register WHERE use_case_id
IS NOT NULL)"),
        "a decision that names UC-999"),
        ("decision outcomes in no list", unread("decisions", "outcome_word", "outcome_map"), "an
outcome such as 'denied'"),
        ("request ids listed more than once (ignoring case)", dup("requests", "request_id"), "a
duplicated request row"),
        ("request rows with no request id", q("SELECT count(*) FROM requests WHERE request_id IS
NULL"), "a request row with no id"),
        ("request kinds in no list", unread("requests", "kind_word", "kind_map"), "a kind such as
'appeal'"),
        ("dates missing or not written as YYYY-MM-DD (a blank notice or description is not counted:
it is a finding)", unreadable,
        "a date such as 6/12/26, or a decision with no date"),
        ("dates after the as-of date", dated_after, "a record dated 2027-01-01"),
        ("descriptions dated before their decision, or answers before their request",
        q("SELECT count(*) FROM decisions WHERE description_on < decided_on") + q("SELECT count(*)
FROM requests WHERE answered_on < received_on"),
        "a request answered two days before it was received"),
    ]
    print(f"\n== Input checks   (register rows: {n_reg}; decision rows: {n_dec}; request rows:
{n_req})")
    ok = True
    for label, n, fault in checks:
        ok &= n == 0
        print(f"[{'ok' if n == 0 else 'FAIL'}] {label}: {n}" + (" (trips on:
{fault})"))
    # a word list entry that matches nothing in your files is reported, never silently ignored
    for tbl, col, src, w in [("register", "decision_word", "decision", "decision_map"),
("register", "colorado_word", "Colorado", "colorado_map"),
        ("decisions", "outcome_word", "outcome", "outcome_map"), ("requests",
"kind_word", "request kind", "kind_map")]:

```

```

        dead = [r[0] for r in con.execute(f"SELECT word FROM {w} WHERE word NOT IN (SELECT {col}
FROM {tbl}) ORDER BY 1").fetchall()]
        if dead:
            print(f"[info] {src} answers in your word list that no row uses: " + "; ".join(dead))

# ---- 1. the use sort
con.execute("""CREATE TABLE sorted_uses AS SELECT r.row_no, coalesce(r.id_shown, '-') AS
use_case_id,
CASE WHEN d.meaning IS NULL THEN 'FOR LEGAL TO CHECK'
    WHEN d.meaning = 'neither' THEN 'NOT AN EMPLOYMENT USE AS RECORDED'
    WHEN c.meaning IS NULL THEN 'FOR LEGAL TO CHECK'
    WHEN c.meaning = 'no' THEN 'NO COLORADO PEOPLE AS RECORDED'
    ELSE 'FOR LEGAL TO CHECK' END AS status,
CASE WHEN d.meaning IS NULL THEN 'decision type not recorded or not recognized'
    WHEN d.meaning = 'neither' THEN ''
    WHEN c.meaning IS NULL THEN 'Colorado not recorded or not recognized'
    WHEN c.meaning = 'no' THEN ''
    ELSE 'employment use that touches Colorado as recorded; SB 26-189' END AS reason
FROM register r LEFT JOIN decision_map d ON d.word = r.decision_word LEFT JOIN colorado_map
c ON c.word = r.colorado_word""")
r_uc = con.execute("SELECT row_no, use_case_id, status, reason FROM sorted_uses ORDER BY
row_no").fetchall()
population(out_dir, "use_sort", "SB 26-189: the use sort",
            "every register row, none dropped; first applicable status; an empty or unrecognized
answer is FOR LEGAL TO CHECK, never outside",
            r_uc, ["row_no", "use_case_id", "status", "reason"], ok)
for r in r_uc:
    print(" " + " | ".join(str(v) for v in r))
counts = {k: sum(1 for r in r_uc if r[2] == k) for k in sorted({r[2] for r in r_uc})}
print(" " + "; ".join(f"{k}: {v}" for k, v in counts.items()))
print(" This sorts uses by what the register says. It does not say the act reaches a use or
does not: Legal decides.")

# ---- 2. decisions: notice, and the description due within 30 days of an adverse outcome
rows = con.execute("""SELECT d.row_no, coalesce(d.id_shown, '-'), o.meaning, d.decided_on,
d.notice_on_given, d.notice_on,
                        d.description_on_given, d.description_on FROM decisions d
LEFT JOIN outcome_map o ON o.word = d.outcome_word ORDER BY
d.row_no""").fetchall()
r_dec, follow_dec, open_dec = [], 0, 0
for row_no, did, outcome, decided, n_given, notice, s_given, sent in rows:
    hard, soft = [], []
    if outcome is None:
        hard.append("outcome not recorded or not recognized")
    if decided is None:
        hard.append("decision date not read")
    else:
        if not n_given and public_since is not None and decided >= public_since:
            pass # 6-1-1704(2): the public notice you keep
meets the duty
        elif not n_given:
            hard.append("NO NOTICE RECORDED")
        elif notice is None:
            hard.append("notice date not read")
        elif notice > decided:
            hard.append(f"NOTICE DATED {(notice - decided).days} DAYS AFTER THE DECISION")
        if outcome == "adverse":
            due = decided + datetime.timedelta(days=DESCRIPTION_DAYS)

```

```

        if not s_given:
            (hard if as_of > due else soft).append(
                f"NO DESCRIPTION, {DESCRIPTION_DAYS} DAYS PASSED (due {due})" if as_of >
due else f"description due {due}")
            elif sent is None:
                hard.append("description date not read")
            elif sent > due:
                hard.append(f"DESCRIPTION SENT {(sent - decided).days} DAYS AFTER THE
DECISION")
            status = "FOR FOLLOW-UP" if hard else ("OPEN, INSIDE 30 DAYS" if soft else "NO FLAG")
            follow_dec += status == "FOR FOLLOW-UP"
            open_dec += status.startswith("OPEN")
            r_dec.append((row_no, did, status, "; ".join(hard + soft)))
            population(out_dir, "decision_review", "SB 26-189: notice and the 30-day description",
                f"every decision row; notice before the decision (6-1-1704(1)); public notice kept
since: {public_since or 'NOT SET (each decision needs its own)'}"; "
                f"adverse = outcome mapped to adverse; description due {DESCRIPTION_DAYS} days after
the decision date (6-1-1704(3)); "
                "a blank description is a finding, an unreadable date is not guessed",
                r_dec, ["row_no", "decision_id", "status", "reason"], ok)
            for r in r_dec:
                if r[2] != "NO FLAG":
                    print(" " + " | ".join(str(v) for v in r))
            dcounts = {k: sum(1 for r in r_dec if r[2] == k) for k in sorted({r[2] for r in r_dec})}
            print(" " + "; ".join(f"{k}: {v}" for k, v in dcounts.items()))
            print(" A flag says what the log records against the 30 days and the notice. It does not say
the act was or was not followed.")

# ---- 3. requests: answered, and within the deadline you set
rows = con.execute("""SELECT q.row_no, coalesce(q.id_shown, '-') , k.meaning, q.received_on,
q.answered_on_given, q.answered_on
FROM requests q LEFT JOIN kind_map k ON k.word = q.kind_word ORDER BY
q.row_no""").fetchall()
r_req, follow_req, open_req = [], 0, 0
for row_no, rid, kind, received, a_given, answered in rows:
    hard, soft = [], []
    if req_days is None:
        hard.append("DEADLINE NOT SET")
    if kind is None:
        hard.append("kind not recorded or not recognized")
    if received is None:
        hard.append("received date not read")
    elif req_days is not None:
        due = received + datetime.timedelta(days=req_days)
        if not a_given:
            (hard if as_of > due else soft).append(
                f"NOT ANSWERED, DEADLINE PASSED (due {due})" if as_of > due else f"open, due
{due}")
            elif answered is None:
                hard.append("answer date not read")
            elif answered > due:
                hard.append(f"ANSWERED {(answered - received).days} DAYS AFTER RECEIPT (deadline
{req_days})")
            status = "FOR FOLLOW-UP" if hard else ("OPEN, INSIDE DEADLINE" if soft else "NO FLAG")
            follow_req += status == "FOR FOLLOW-UP"
            open_req += status.startswith("OPEN")
            r_req.append((row_no, rid, status, "; ".join(hard + soft)))
            population(out_dir, "request_review", "SB 26-189: requests answered",

```

```

        f"every request row; deadline = {str(req_days) + ' days' if req_days else 'NOT SET'}
from receipt, a number YOU set (6-1-1705 states none); "
        "a blank answer date is a finding", r_req, ["row_no", "request_id", "status",
"reason"], ok)
    for r in r_req:
        if r[2] != "NO FLAG":
            print(" " + " | ".join(str(v) for v in r))
    rcounts = {k: sum(1 for r in r_req if r[2] == k) for k in sorted({r[2] for r in r_req})}
    print(" " + "; ".join(f"{k}: {v}" for k, v in rcounts.items()))

# ---- 4. records: a decision record in every month of the window, and no missing id numbers
gaps_found = 0 # every gap of any kind, for --strict
month_gaps, id_missing = 0, {"decision": 0, "request": 0}
rec_rows = []
if first_use is None:
    window_start = None
    print("\n== SB 26-189: records kept ** NOT RUN: first-use date not set **")
    gaps_found += 1
else:
    window_start = max(years_before(as_of, RETENTION_YEARS), first_use)
    have = {r[0]: r[1] for r in con.execute(
        "SELECT strftime(decided_on, '%Y-%m'), count(*) FROM decisions WHERE decided_on IS NOT
NULL GROUP BY 1").fetchall()}
    m, months = month_start(window_start), []
    while m <= as_of:
        months.append(m.strftime("%Y-%m"))
        m = next_month(m)
    running = as_of.strftime("%Y-%m") if next_month(month_start(as_of)) -
datetime.timedelta(days=1) != as_of else None
    partial = window_start.strftime("%Y-%m") if window_start.day != 1 else None
    for mo in months:
        c = have.get(mo, 0)
        if c:
            status = "RECORDS"
        elif mo == running:
            status = "MONTH NOT OVER, NO RECORDS YET"
        elif mo == partial:
            status = "PART MONTH AT WINDOW START, NO RECORDS"
        else:
            status = "NO RECORDS, EXPLAINED" if mo in gap_months else "NO RECORDS IN THIS
MONTH"

        gaps_found += status == "NO RECORDS IN THIS MONTH"
        month_gaps += status == "NO RECORDS IN THIS MONTH"
        rec_rows.append(("month", mo, c, status))
    if ids_seq:
        for log, tbl in [("decision", "decisions"), ("request", "requests")]:
            ids = [r[0] for r in con.execute(f"SELECT id_shown FROM {tbl} WHERE id_shown IS NOT
NULL").fetchall()]
            n_missing, listed = id_gaps(ids, MISSING_LISTED)
            gaps_found += n_missing
            id_missing[log] = n_missing
            status = "NO MISSING NUMBERS" if not n_missing else "MISSING: " + ", ".join(listed)
+ (
            f" (and {n_missing - len(listed)} more)" if n_missing > len(listed) else ""
            rec_rows.append((f"{log} ids", f"{len(ids)} ids read", n_missing, status))
    older = q("SELECT count(*) FROM decisions WHERE decided_on < ?::DATE", str(window_start))
    rec_rows.append(("older than the window", str(window_start), older,
        "NONE OLDER" if not older else "OLDER RECORDS: keep or not as you choose;

```

```

the window is the minimum"))
    population(out_dir, "records_kept", "SB 26-189: records kept for at least 3 years",
        f"window {window_start} to {as_of} (the later of {RETENTION_YEARS} years before
the as-of date and your first-use date); one record needed in each month "
        f"unless explained (a month still running on the as-of date, or a part month at
the window start, is not a gap); "
        f"id numbers {'checked for gaps' if ids_seq else 'NOT checked
(IDS_ARE_SEQUENTIAL is ' + str(ids_seq) + ')}'",
        rec_rows, ["what", "which", "count", "status"], ok)
    for r in rec_rows:
        if r[3] not in ("RECORDS", "NO MISSING NUMBERS", "NONE OLDER"):
            shown_status = r[3]
            if r[3].startswith("MISSING: "):
                parts = r[3][len("MISSING: "):].split(" (and ")
                parts = parts[0].split(", ")
                shown_status = "MISSING: " + ", ".join(parts[:12]) + (f" ... ({r[2]} in all;
every one is in records_kept.csv)" if r[2] > 12 else "")
            print(" " + " | ".join(str(v) for v in r[:3]) + " | " + shown_status)
            print(f" months in the window: {len(months)}; months with no record: {sum(1 for r in
rec_rows if r[3] == 'NO RECORDS IN THIS MONTH'))}")
            print(" A month with no record may be real (no decisions) or a deleted log: the script
cannot tell which. Explain it in KNOWN_GAP_MONTHS or find the records.")

    before = q("SELECT count(*) FROM decisions WHERE decided_on < ?::DATE", APPLIES_FROM)
    print(f"\n[info] decisions dated before {APPLIES_FROM}, when the act begins to apply (Section
5(3)): {before} of {n_dec}")
    uses_legal = sum(1 for r in r_uc if r[2] == "FOR LEGAL TO CHECK")
    print("\n== Result")
    print(f"input checks : {'all passed' if ok else 'FAILED, the lists above are not complete'}")
    print(f"for Legal : {uses_legal} use(s) marked FOR LEGAL TO CHECK")
    print(f"decisions : {follow_dec} for follow-up, {open_dec} open inside 30 days")
    print(f"requests : {follow_req} for follow-up, {open_req} open inside the deadline")
    if first_use is None:
        print("records : NOT RUN (first-use date not set)")
    else:
        ids_part = (f"{id_missing['decision']} missing decision id(s), {id_missing['request']}
missing request id(s)" if ids_seq
                    else "id numbers not checked")
        print(f"records : {month_gaps} month(s) with no decision record, {ids_part}")
        print("This is not legal advice and not a compliance test. Legal decides what the act
requires.")
        # --strict: FOR LEGAL TO CHECK uses are a prompt for Legal, not a failure, so they never set
the exit code
        return 1 if strict and (not ok or follow_dec or follow_req or gaps_found) else 0

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

```

The runs

Run 1: clean generated records

Seed 226, seven invented register rows, 120 invented decisions and 30 invented requests, with --as-of 2026-10-31 --request-days 45 --first-use-on 2026-01-01 --ids-sequential

yes.

```
input      : GENERATED records (not a real organization), seed 226, mode=clean
source     : SB 26-189, session law (Ch. 131), https://leg.colorado.gov/laws/session-laws/SB26-189/131/download (not fetched by this script)
source     : SB 26-189, bill page, https://leg.colorado.gov/bills/sb26-189 (not fetched by this script)
as of      : 2026-10-31    request deadline: 45 days    first use: 2026-01-01    explained gap months: none

== Input checks    (register rows: 7; decision rows: 120; request rows: 30)
[ok ] as-of date not written as YYYY-MM-DD: 0
[ok ] request deadline not set, or not a whole number of days above zero: 0
[ok ] ids-are-sequential setting not set (True or False): 0
[ok ] public-notice date not written as YYYY-MM-DD: 0
[ok ] first-use date not set, or not written as YYYY-MM-DD: 0
[ok ] explained gap months not written as YYYY-MM: 0
[ok ] register ids listed more than once (ignoring case): 0
[ok ] register rows with no use case id: 0
[ok ] register answers in no list (decision, Colorado): 0
[ok ] decision ids listed more than once (ignoring case): 0
[ok ] decision rows with no decision id: 0
[ok ] decisions naming a use that is not on the register: 0
[ok ] decision outcomes in no list: 0
[ok ] request ids listed more than once (ignoring case): 0
[ok ] request rows with no request id: 0
[ok ] request kinds in no list: 0
[ok ] dates missing or not written as YYYY-MM-DD (a blank notice or description is not counted: it is a finding): 0
[ok ] dates after the as-of date: 0
[ok ] descriptions dated before their decision, or answers before their request: 0
[info] decision answers in your word list that no row uses: discipline; other employment decision; termination

== SB 26-189: the use sort: use_sort
filters    : every register row, none dropped; first applicable status; an empty or unrecognized answer is FOR LEGAL TO CHECK, never outside
rows       : 7
sha256     : 8c7982900c4a567874a33a838bb20dd2780c6e0c83fbb50fc8e57be5e9b2429c
  1 | UC-201 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
  2 | UC-202 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
  3 | UC-203 | NOT AN EMPLOYMENT USE AS RECORDED |
  4 | UC-204 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
  5 | UC-205 | NOT AN EMPLOYMENT USE AS RECORDED |
  6 | UC-206 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
  7 | UC-207 | NO COLORADO PEOPLE AS RECORDED |
  FOR LEGAL TO CHECK: 4; NO COLORADO PEOPLE AS RECORDED: 1; NOT AN EMPLOYMENT USE AS RECORDED: 2
  This sorts uses by what the register says. It does not say the act reaches a use or does not: Legal decides.

== SB 26-189: notice and the 30-day description: decision_review
filters    : every decision row; notice before the decision (6-1-1704(1)); public notice kept since: NOT SET (each decision needs its own); adverse = outcome mapped to adverse; description due 30 days after the decision date (6-1-1704(3)); a blank description is a finding, an unreadable date is not guessed
rows       : 120
sha256     : 80291a70a20fa13e3596cc12f76bf6c60f20715c7a28e040b2e2c77d3c0ce047
```

NO FLAG: 120

A flag says what the log records against the 30 days and the notice. It does not say the act was or was not followed.

== SB 26-189: requests answered: request_review

filters : every request row; deadline = 45 days from receipt, a number YOU set (6-1-1705 states none); a blank answer date is a finding

rows : 30

sha256 : e44fc29dd127aa99165e40ffb8d7787bf5e9b74abcb81cac29642cec29cc6796

30 | R-030 | OPEN, INSIDE DEADLINE | open, due 2026-11-28

NO FLAG: 29; OPEN, INSIDE DEADLINE: 1

== SB 26-189: records kept for at least 3 years: records_kept

filters : window 2026-01-01 to 2026-10-31 (the later of 3 years before the as-of date and your first-use date); one record needed in each month unless explained (a month still running on the as-of date, or a part month at the window start, is not a gap); id numbers checked for gaps

rows : 13

sha256 : 9451d515206a2a9d357323a449b5f1302e479652e585a7127c4bb90619cf1d44

months in the window: 10; months with no record: 0

A month with no record may be real (no decisions) or a deleted log: the script cannot tell which. Explain it in KNOWN_GAP_MONTHS or find the records.

[info] decisions dated before 2027-01-01, when the act begins to apply (Section 5(3)): 120 of 120

== Result

input checks : all passed

for Legal : 4 use(s) marked FOR LEGAL TO CHECK

decisions : 0 for follow-up, 0 open inside 30 days

requests : 0 for follow-up, 1 open inside the deadline

records : 0 month(s) with no decision record, 0 missing decision id(s), 0 missing request id(s)

This is not legal advice and not a compliance test. Legal decides what the act requires.

Run 2: planted faults

The same command with --plant.

input : GENERATED records (not a real organization), seed 226, mode=plant

source : SB 26-189, session law (Ch. 131), <https://leg.colorado.gov/laws/session-laws/SB26-189/131/download> (not fetched by this script)

source : SB 26-189, bill page, <https://leg.colorado.gov/bills/sb26-189> (not fetched by this script)

as of : 2026-10-31 request deadline: 45 days first use: 2026-01-01 explained gap months: none

== Input checks (register rows: 12; decision rows: 109; request rows: 32)

[ok] as-of date not written as YYYY-MM-DD: 0

[ok] request deadline not set, or not a whole number of days above zero: 0

[ok] ids-are-sequential setting not set (True or False): 0

[ok] public-notice date not written as YYYY-MM-DD: 0

[ok] first-use date not set, or not written as YYYY-MM-DD: 0

[ok] explained gap months not written as YYYY-MM: 0

[FAIL] register ids listed more than once (ignoring case): 1 (trips on: a duplicated register row)

[FAIL] register rows with no use case id: 1 (trips on: a register row with no id)

[FAIL] register answers in no list (decision, Colorado): 4 (trips on: an answer such as

```

'selection' that is not one of your words (an empty answer counts))
[FAIL] decision ids listed more than once (ignoring case): 1 (trips on: a duplicated decision
row)
[FAIL] decision rows with no decision id: 1 (trips on: a decision row with no id)
[FAIL] decisions naming a use that is not on the register: 1 (trips on: a decision that names UC-
999)
[FAIL] decision outcomes in no list: 1 (trips on: an outcome such as 'denied')
[FAIL] request ids listed more than once (ignoring case): 1 (trips on: a duplicated request row)
[FAIL] request rows with no request id: 1 (trips on: a request row with no id)
[FAIL] request kinds in no list: 1 (trips on: a kind such as 'appeal')
[FAIL] dates missing or not written as YYYY-MM-DD (a blank notice or description is not counted: it
is a finding): 1 (trips on: a date such as 6/12/26, or a decision with no date)
[FAIL] dates after the as-of date: 1 (trips on: a record dated 2027-01-01)
[FAIL] descriptions dated before their decision, or answers before their request: 1 (trips on: a
request answered two days before it was received)
[info] decision answers in your word list that no row uses: other employment decision; termination

== SB 26-189: the use sort: use_sort ** INCOMPLETE: see input checks **
filters : every register row, none dropped; first applicable status; an empty or unrecognized
answer is FOR LEGAL TO CHECK, never outside
rows : 12
sha256 : 8eee422191a0591af1e1d3f541c1770cd94b5b4043fa5f032717335d2036eeef
 1 | UC-201 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
 2 | UC-202 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
 3 | UC-203 | NOT AN EMPLOYMENT USE AS RECORDED |
 4 | UC-204 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
 5 | UC-205 | NOT AN EMPLOYMENT USE AS RECORDED |
 6 | UC-206 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
 7 | UC-207 | NO COLORADO PEOPLE AS RECORDED |
 8 | UC-208 | FOR LEGAL TO CHECK | Colorado not recorded or not recognized
 9 | uc-209 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
10 | UC-210 | FOR LEGAL TO CHECK | decision type not recorded or not recognized
11 | UC-202 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
12 | - | FOR LEGAL TO CHECK | decision type not recorded or not recognized
FOR LEGAL TO CHECK: 9; NO COLORADO PEOPLE AS RECORDED: 1; NOT AN EMPLOYMENT USE AS RECORDED: 2
This sorts uses by what the register says. It does not say the act reaches a use or does not:
Legal decides.

== SB 26-189: notice and the 30-day description: decision_review ** INCOMPLETE: see input checks
**
filters : every decision row; notice before the decision (6-1-1704(1)); public notice kept since:
NOT SET (each decision needs its own); adverse = outcome mapped to adverse; description due 30 days
after the decision date (6-1-1704(3)); a blank description is a finding, an unreadable date is not
guessed
rows : 109
sha256 : 28c31ef1535cdf78fe531375c0ec3a227f92eff48fd98119fa873ce26a994529
38 | D-050 | FOR FOLLOW-UP | NO NOTICE RECORDED
39 | D-051 | FOR FOLLOW-UP | NO DESCRIPTION, 30 DAYS PASSED (due 2026-06-05)
40 | D-052 | FOR FOLLOW-UP | DESCRIPTION SENT 41 DAYS AFTER THE DECISION
41 | D-053 | FOR FOLLOW-UP | decision date not read
42 | D-054 | FOR FOLLOW-UP | outcome not recorded or not recognized
FOR FOLLOW-UP: 5; NO FLAG: 104
A flag says what the log records against the 30 days and the notice. It does not say the act was
or was not followed.

== SB 26-189: requests answered: request_review ** INCOMPLETE: see input checks **
filters : every request row; deadline = 45 days from receipt, a number YOU set (6-1-1705 states
none); a blank answer date is a finding

```

```

rows      : 32
sha256    : 7e2b7315095fdef72311b69f63f5253f8ddd96177e3aa36cadb9fd94b2c8c79d
  4 | R-004 | FOR FOLLOW-UP | NOT ANSWERED, DEADLINE PASSED (due 2026-04-21)
  6 | R-006 | FOR FOLLOW-UP | ANSWERED 60 DAYS AFTER RECEIPT (deadline 45)
 10 | R-010 | FOR FOLLOW-UP | kind not recorded or not recognized
 30 | R-030 | OPEN, INSIDE DEADLINE | open, due 2026-11-28
FOR FOLLOW-UP: 3; NO FLAG: 28; OPEN, INSIDE DEADLINE: 1

== SB 26-189: records kept for at least 3 years: records_kept  ** INCOMPLETE: see input checks **
filters   : window 2026-01-01 to 2026-10-31 (the later of 3 years before the as-of date and your
first-use date); one record needed in each month unless explained (a month still running on the as-
of date, or a part month at the window start, is not a gap); id numbers checked for gaps
rows      : 13
sha256    : 08003732da528a9e79577ef4fd2542da6d4bc85e8c1cd311cf4745e11c431ad7
  month | 2026-03 | 0 | NO RECORDS IN THIS MONTH
  decision ids | 108 ids read | 13 | MISSING: D-025, D-026, D-027, D-028, D-029, D-030, D-031, D-
032, D-033, D-034, D-035, D-036 ... (13 in all; every one is in records_kept.csv)
  months in the window: 10; months with no record: 1
  A month with no record may be real (no decisions) or a deleted log: the script cannot tell which.
  Explain it in KNOWN_GAP_MONTHS or find the records.

[info] decisions dated before 2027-01-01, when the act begins to apply (Section 5(3)): 108 of 109

== Result
input checks : FAILED, the lists above are not complete
for Legal    : 9 use(s) marked FOR LEGAL TO CHECK
decisions    : 5 for follow-up, 0 open inside 30 days
requests     : 3 for follow-up, 1 open inside the deadline
records      : 1 month(s) with no decision record, 13 missing decision id(s), 0 missing request
id(s)
This is not legal advice and not a compliance test. Legal decides what the act requires.

```

Run 3: no request deadline set

The same clean records, with no `--request-days`.

```

input      : GENERATED records (not a real organization), seed 226, mode=clean
source     : SB 26-189, session law (Ch. 131), https://leg.colorado.gov/laws/session-laws/SB26-
189/131/download (not fetched by this script)
source     : SB 26-189, bill page, https://leg.colorado.gov/bills/sb26-189 (not fetched by this
script)
as of      : 2026-10-31   request deadline: NOT SET   first use: 2026-01-01   explained gap months:
none

== Input checks (register rows: 7; decision rows: 120; request rows: 30)
[ok ] as-of date not written as YYYY-MM-DD: 0
[FAIL] request deadline not set, or not a whole number of days above zero: 1 (trips on: no --
request-days and REQUEST_DAYS left as None; 6-1-1705 states no deadline, so you set one)
[ok ] ids-are-sequential setting not set (True or False): 0
[ok ] public-notice date not written as YYYY-MM-DD: 0
[ok ] first-use date not set, or not written as YYYY-MM-DD: 0
[ok ] explained gap months not written as YYYY-MM: 0
[ok ] register ids listed more than once (ignoring case): 0
[ok ] register rows with no use case id: 0
[ok ] register answers in no list (decision, Colorado): 0
[ok ] decision ids listed more than once (ignoring case): 0

```

```
[ok ] decision rows with no decision id: 0
[ok ] decisions naming a use that is not on the register: 0
[ok ] decision outcomes in no list: 0
[ok ] request ids listed more than once (ignoring case): 0
[ok ] request rows with no request id: 0
[ok ] request kinds in no list: 0
[ok ] dates missing or not written as YYYY-MM-DD (a blank notice or description is not counted: it
is a finding): 0
[ok ] dates after the as-of date: 0
[ok ] descriptions dated before their decision, or answers before their request: 0
[info] decision answers in your word list that no row uses: discipline; other employment decision;
termination
```

```
== SB 26-189: the use sort: use_sort  ** INCOMPLETE: see input checks **
filters : every register row, none dropped; first applicable status; an empty or unrecognized
answer is FOR LEGAL TO CHECK, never outside
rows : 7
sha256 : 8c7982900c4a567874a33a838bb20dd2780c6e0c83fbb50fc8e57be5e9b2429c
 1 | UC-201 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
 2 | UC-202 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
 3 | UC-203 | NOT AN EMPLOYMENT USE AS RECORDED |
 4 | UC-204 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
 5 | UC-205 | NOT AN EMPLOYMENT USE AS RECORDED |
 6 | UC-206 | FOR LEGAL TO CHECK | employment use that touches Colorado as recorded; SB 26-189
 7 | UC-207 | NO COLORADO PEOPLE AS RECORDED |
FOR LEGAL TO CHECK: 4; NO COLORADO PEOPLE AS RECORDED: 1; NOT AN EMPLOYMENT USE AS RECORDED: 2
This sorts uses by what the register says. It does not say the act reaches a use or does not:
Legal decides.
```

```
== SB 26-189: notice and the 30-day description: decision_review  ** INCOMPLETE: see input checks
**
filters : every decision row; notice before the decision (6-1-1704(1)); public notice kept since:
NOT SET (each decision needs its own); adverse = outcome mapped to adverse; description due 30 days
after the decision date (6-1-1704(3)); a blank description is a finding, an unreadable date is not
guessed
rows : 120
sha256 : 80291a70a20fa13e3596cc12f76bf6c60f20715c7a28e040b2e2c77d3c0ce047
NO FLAG: 120
A flag says what the log records against the 30 days and the notice. It does not say the act was
or was not followed.
```

```
== SB 26-189: requests answered: request_review  ** INCOMPLETE: see input checks **
filters : every request row; deadline = NOT SET from receipt, a number YOU set (6-1-1705 states
none); a blank answer date is a finding
rows : 30
sha256 : 3c12b8ecd040e4b92a7b02a1b5856f0fbf5a3b405e3d4e2d17502c4e4516fd9f
 1 | R-001 | FOR FOLLOW-UP | DEADLINE NOT SET
 2 | R-002 | FOR FOLLOW-UP | DEADLINE NOT SET
 3 | R-003 | FOR FOLLOW-UP | DEADLINE NOT SET
 4 | R-004 | FOR FOLLOW-UP | DEADLINE NOT SET
 5 | R-005 | FOR FOLLOW-UP | DEADLINE NOT SET
 6 | R-006 | FOR FOLLOW-UP | DEADLINE NOT SET
 7 | R-007 | FOR FOLLOW-UP | DEADLINE NOT SET
 8 | R-008 | FOR FOLLOW-UP | DEADLINE NOT SET
 9 | R-009 | FOR FOLLOW-UP | DEADLINE NOT SET
10 | R-010 | FOR FOLLOW-UP | DEADLINE NOT SET
11 | R-011 | FOR FOLLOW-UP | DEADLINE NOT SET
12 | R-012 | FOR FOLLOW-UP | DEADLINE NOT SET
```

```

13 | R-013 | FOR FOLLOW-UP | DEADLINE NOT SET
14 | R-014 | FOR FOLLOW-UP | DEADLINE NOT SET
15 | R-015 | FOR FOLLOW-UP | DEADLINE NOT SET
16 | R-016 | FOR FOLLOW-UP | DEADLINE NOT SET
17 | R-017 | FOR FOLLOW-UP | DEADLINE NOT SET
18 | R-018 | FOR FOLLOW-UP | DEADLINE NOT SET
19 | R-019 | FOR FOLLOW-UP | DEADLINE NOT SET
20 | R-020 | FOR FOLLOW-UP | DEADLINE NOT SET
21 | R-021 | FOR FOLLOW-UP | DEADLINE NOT SET
22 | R-022 | FOR FOLLOW-UP | DEADLINE NOT SET
23 | R-023 | FOR FOLLOW-UP | DEADLINE NOT SET
24 | R-024 | FOR FOLLOW-UP | DEADLINE NOT SET
25 | R-025 | FOR FOLLOW-UP | DEADLINE NOT SET
26 | R-026 | FOR FOLLOW-UP | DEADLINE NOT SET
27 | R-027 | FOR FOLLOW-UP | DEADLINE NOT SET
28 | R-028 | FOR FOLLOW-UP | DEADLINE NOT SET
29 | R-029 | FOR FOLLOW-UP | DEADLINE NOT SET
30 | R-030 | FOR FOLLOW-UP | DEADLINE NOT SET
FOR FOLLOW-UP: 30

```

```

== SB 26-189: records kept for at least 3 years: records_kept  ** INCOMPLETE: see input checks **
filters   : window 2026-01-01 to 2026-10-31 (the later of 3 years before the as-of date and your
first-use date); one record needed in each month unless explained (a month still running on the as-
of date, or a part month at the window start, is not a gap); id numbers checked for gaps
rows      : 13
sha256    : 9451d515206a2a9d357323a449b5f1302e479652e585a7127c4bb90619cf1d44
months in the window: 10; months with no record: 0
A month with no record may be real (no decisions) or a deleted log: the script cannot tell which.
Explain it in KNOWN_GAP_MONTHS or find the records.

```

```

[info] decisions dated before 2027-01-01, when the act begins to apply (Section 5(3)): 120 of 120

```

```

== Result
input checks : FAILED, the lists above are not complete
for Legal    : 4 use(s) marked FOR LEGAL TO CHECK
decisions    : 0 for follow-up, 0 open inside 30 days
requests     : 30 for follow-up, 0 open inside the deadline
records      : 0 month(s) with no decision record, 0 missing decision id(s), 0 missing request
id(s)
This is not legal advice and not a compliance test. Legal decides what the act requires.

```

How to read the results

- **A status is a prompt for Legal, not a finding.** FOR LEGAL TO CHECK means the register says the use is an employment use that touches Colorado, or the register is silent or not recognized.
- **FOR FOLLOW-UP means the log records something against the 30 days, the notice or your deadline.** It does not say the act was breached: the log may be wrong, and the act is the authority.
- **OPEN is not a fault.** It means the due date has not yet passed on the as-of date.

- **A month with no record is a question.** Find the records, or write the month in `KNOWN_GAP_MONTHS` with your reason.
- **An INCOMPLETE list is not a result.** Fix the failed input checks first.

Checklist

Item	Where
☐ Read the signed act with your counsel, including 6-1-1306 on request timing	Before anything else
☐ Keep a register of Claude uses in HR with a Colorado answer for each	Step 2
☐ Record a notice date before every decision where Claude's output was used, or the day a public notice began	Step 3
☐ Record the outcome, and the date a description was sent after an adverse one	Step 3
☐ Log every request for data, correction or human review with received and answered dates	Step 4
☐ Set the request deadline with counsel (<code>REQUEST_DAYS</code> has no default)	Step 1
☐ Set <code>FIRST_USE_ON</code> , <code>IDS_ARE_SEQUENTIAL</code> and <code>AS_OF</code> ; write every date YYYY-MM-DD	Step 1
☐ Fix every failed input check before reading a list	Step 6
☐ Take every FOR LEGAL TO CHECK row to counsel	Step 2
☐ Explain or fill every month with no record, and every missing id number	Step 5
☐ Keep the logs under your own access controls	Limits

Run it on your own data

To run it on your own data, change only the settings between the lines marked **CHANGE THESE FOR YOUR OWN DATA**. Set `REGISTER_FILE`, `DECISIONS_FILE` and `REQUESTS_FILE` to the paths of your three CSV files. In `REG_COLS`, `DEC_COLS` and `REQ_COLS`, put the name of the column in your files that holds each field. In `DECISION_WORDS`, `COLORADO_WORDS`, `OUTCOME_WORDS` and `KIND_WORDS`, put the answers your files use, in lower case, each with the meaning the script gives it. Set `REQUEST_DAYS` to the number of days your counsel allows for answering a request, `FIRST_USE_ON` to the day you began using Claude in employment decisions, written YYYY-MM-DD, and `IDS_ARE_SEQUENTIAL` to True if your ids end in a number that goes up by one, or to False if they do not. If you keep a prominent public notice at points of consumer interaction, set `PUBLIC_NOTICE_SINCE` to the day you began, written YYYY-MM-DD. Leave `AS_OF` as None to

check as of today, or set it to a date written YYYY-MM-DD. An answer that is in none of those lists is counted and fails an input check, so the run tells you which one to add.

We also ran the script on a second set of generated files with other file names, other column names, other answer words, upper-case answers, a blank Colorado answer and a blank use id, and a fault planted for each input check that reads the files, except a description dated before its decision. It read `RECRUITMENT` and `Y` as an employment use that touches Colorado, sent the blank Colorado answer to Legal, and failed the checks for the repeated ids (including `hr-5` beside `HR-5`), the rows with no id, the decision that names a use not on the register, the date after the as-of date, and the answers in no list. It flagged a decision with no notice, an adverse decision with no description after 30 days, a description sent on day 48, a request never answered, a month with no record, and the missing id numbers, printed as the log writes them (`CASE-09`). With `PUBLIC_NOTICE_SINCE` set, it did not flag a later decision that had no notice of its own; with the setting removed, it did. With the as-of date in the middle of a month that had no records yet, it showed that month as `MONTH NOT OVER, NO RECORDS YET`, and with the first-use date in the middle of a month that had none, it showed that month as `PART MONTH AT WINDOW START, NO RECORDS`; it counted neither as a gap. We have not run it on a real company's records.

[Download colorado_sb26189_claude_check.py](#)

[Download the full guide \(PDF\)](#)

Method

- **Input:** generated records, written by the script with a fixed seed (226): seven invented register rows, 120 invented decisions and 30 invented requests. The outcomes, dates and delays are invented for the demonstration.
- **The law's text:** the duties, the 30 days, the 3 years, the dates and the enforcement points are quoted from the [signed act as published in the 2026 Session Laws, Chapter 131](#) (read 2026-10-31), with section numbers. The legislature's site refuses scripted downloads, so the script does not fetch the act and nothing on this page is pinned.
- **The 30 days:** the act counts the 30 days from the decision ("within thirty days after making the decision"); the script uses the decision date, so a description sent on the 30th day is not flagged.
- **The window:** the later of three years before the as-of date and the first-use date you give, to the as-of date.
- **Settings you can disagree with** are at the top of the script: the answer lists, the column names, the request deadline, the first-use date, whether ids are sequential and the public-notice date.

- **Every problem row is counted.** An unrecognized answer or date is counted and never guessed. A failed input check marks the lists INCOMPLETE.

Limits

- **This is not a client result.** The records are generated. The runs show the script finds planted faults. They say nothing about any real company.
- **This is not legal advice,** and the script does not decide whether SB 26-189 reaches a use. It sorts uses by what your register says. Legal decides.
- **We quote the act; we do not apply all of it.** The script does not test the definitions (covered ADMT, materially influence, consumer, deployer) or the act's provisions on compliance with other legal obligations, insurers and covered entities (6-1-1708). Legal applies them.
- **The rules are still to come.** The signed act takes effect January 1, 2027, and applies to consequential decisions made on or after that date (Section 5). The attorney general's rules on the description and on consumer rights are due on or before the same day (6-1-1704(4)(b), 6-1-1705(3)) and may change what a description must say. Check the attorney general's page for the current position.
- **The request deadline is yours.** 6-1-1705 states no deadline; it points to 6-1-1306, which Legal should read for response timing. 45 days in the demo is a number we chose.
- **Our reading is not Colorado's.** The monthly record test, the id-gap test and the mapping of your answers are ours. We have not asked the attorney general to review them.
- **A log can only show what it holds.** A decision never entered in the log cannot be checked, and a month with no record may be a quiet month.
- **The logs hold personal data.** Keep them under your own controls and out of the PDF, the CSVs and anything you share.

Tested on

Tested on

Platform

DuckDB 1.5.6, Python 3.14, run from Python with the duckdb module

Edition

Open-source DuckDB, in-memory

Date

2026-10-31

Dataset

Generated records, seed 226 (not a real organization). Duties and dates from the [signed act, Session Laws 2026, Ch. 131](#) (read 2026-10-31)

Code

[colorado_sb26189_claude_check.py](#)

When this is not enough

- **You need an answer on whether the act reaches a use.** Take the `FOR LEGAL TO CHECK` rows, and the rows whose answers you doubt, to your counsel with the signed act.
- **You do not yet keep these logs.** A notice, a description and a request have to be recorded with dates before this script has anything to check.
- **You need to know what the technology did in each decision.** The script checks dates and ids. It does not check what a description said or whether a human review was meaningful.
- **The attorney general's rules arrive.** Re-read what a description must contain, and change the settings and your logs to match.

Sources

- [SB 26-189, Colorado Session Laws 2026, Chapter 131 \(the signed act\)](#) (<https://leg.colorado.gov/laws/session-laws/SB26-189/131/download>), read 2026-10-31
- [Colorado General Assembly: SB26-189 \(bill page\)](#) (<https://leg.colorado.gov/bills/sb26-189>), read 2026-10-31
- [DuckDB: CSV overview](#) (<https://duckdb.org/docs/2.0/data/csv/overview.html>), read 2026-10-31
- [Data Meaning: Control Tower for Claude](#) (<https://www.datameaning.com/solutions/claude-control-tower.html>), read 2026-10-31
- [Data Meaning: NYC Local Law 144 and Claude in hiring \(guide\)](#) (<https://www.datameaning.com/guides/nyc-local-law-144-and-claude-in-hiring/>), read 2026-10-31

Run Claude in your own Azure subscription

Control Tower for Claude runs in your own Azure subscription. See the solution page: <https://www.datameaning.com/solutions/claude-control-tower.html>