pravidha

Pramana

Administrator Guide

Governance evidence for Snowflake — classification, policy coverage, drift, audit packs

Document	Pramana — Administrator Guide
Version	v1.0
Date	17 November 2026
Author	Pravidha — Snowflake Select Service Partner
Status	Released
Classification	Public

© 2026 Pravidha. This document and the software it describes are Pravidha intellectual property.

Contents

A week with Pramana
The consoles
The policy pack
    Two ways to maintain it
    Tuning what counts as sensitive
    Bindings
Working the review queue
Exemptions
Closing findings
Drift
Controls and frameworks
Recipes

© 2026 Pravidha · Public

For the governance engineers who maintain Pramana's policy pack and the data stewards who work its queue. It assumes Pramana is installed and has run at least once; Install and Setup covers that.

A week with Pramana

Pramana runs once a night. Everything else is people responding to what it found.

When	Who	What
Every night	Pramana	Scans, proposes classifications, evaluates the controls, compares with the night before, writes the evidence packs
Most mornings	A data steward	Works the review queue; looks at any weakening drift and acknowledges it
When findings appear	A governance or platform engineer	Generates the statements that close them and puts them through change control
When the organisation changes its mind	An administrator	Updates the policy pack: a new category, a new binding, a new database in scope, an exemption
When asked	Audit, risk, compliance	Read the consoles and take evidence packs away

The consoles

Three Streamlit consoles, one per role. Inside an application a console cannot see who is looking at it, so which console a role may open is what decides what it can do.

Console	Open to	Pages
PRAMANA_VIEWER (the default)	every role	Overview, Controls, Review queue, Drift, Evidence, Policy pack. All read-only
PRAMANA_REVIEW	stewards, admins	The same pages, plus: approve and reject proposals, acknowledge drift, add exemptions
PRAMANA_CONSOLE	admins	All of that, plus Setup, Remediation, Run and Sample

Page	What it is for
Overview	Overall status, failing controls, queue size, open weakening drift, coverage per database
Controls	Findings filtered by control and status; every sensitive column with the policy it should have and the one it has; add an exemption
Review queue	Approve or reject one proposal, or every proposal at or above a confidence
Drift	Open events, weakening first; acknowledge with a note
Evidence	Every pack: readable, as JSON, and the SQL to take it away
Policy pack	The ten tables as they stand, and every table with the governed tags it carries; validate

© 2026 Pravidha · Public

Page	What it is for
Setup	What Pramana may read, the scope, the classification tag, the governed tags, the frameworks
Remediation	Generate the statements, or the same as intents for Dhruva; the latest batch as one script
Run	Run any step; start and stop the schedule; the run log
Sample	Load, break, restore and clear the sample estate

The sidebar of a console says which one it is: the edition, the schema it reads and a chip reading ADMIN, STEWARD or READ-ONLY. In every table a status is tinted: red for what fails or is missing, amber for what warns or waits, green for what passes, grey for what is exempt or was skipped. The consoles follow Snowsight's light or dark mode. Their fonts are loaded by your browser from `fonts.gstatic.com`; where that host is blocked they use the system font and nothing else changes. A console fetches nothing else from outside Snowflake.

The policy pack

Your governance intent as data: ten tables in schema `GOV`. A fresh install has a taxonomy of nine categories, the thirteen controls, and the clause maps for UK GDPR, DORA and ISO 27001. Scope, bindings, the values of your governed tags and exemptions start empty.

Table	What you say in it
`GOV_SETTING`	The classification tag, the schedule, the frameworks, thresholds
`GOV_SCOPE`	Which databases Pramana examines, and which schemas and tables it leaves out
`GOV_CATEGORY`	Your taxonomy: the kinds of sensitive data, and how each is recognised
`GOV_BINDING`	Which masking policy and row access policy protect each category
`GOV_TAG`, `GOV_TAG_VALUE`	The table-level tags you govern by, such as retention, and the values each may carry
`GOV_CONTROL`	Which controls are on, and whether each fails or warns
`GOV_FRAMEWORK`	Which control is evidence for which clause
`GOV_EXEMPTION`	The findings you have accepted, with reason, approver and expiry
`GOV_EVIDENCE_SOURCE`	Other evidence to carry in each pack

Every column of every table is described in the Reference.

Two ways to maintain it

As rows. A `PRAMANA_ADMIN` holder has `INSERT`, `UPDATE` and `DELETE` on the ten tables:

```
INSERT INTO PRAMANA.GOV.GOV_BINDING (CATEGORY, MASKING_POLICY)
VALUES ('PII_EMAIL', 'GOVERNANCE.POLICIES.MASK_EMAIL');

INSERT INTO PRAMANA.GOV.GOV_BINDING (CATEGORY, MASKING_POLICY, ROW_ACCESS_POLICY, RAP_COLUMN)
VALUES ('HEALTH', 'GOVERNANCE.POLICIES.MASK_HEALTH', 'GOVERNANCE.POLICIES.RAP_REGION', 'REGION');

UPDATE PRAMANA.GOV.GOV_CONTROL SET SEVERITY = 'WARN' WHERE CONTROL_ID = 'ACC-02';
```

© 2026 Pravidha · Public

As CSV files in git. One file per table, named after it in lower case, headers equal to the column names. Download the template pack from the documentation page, edit it, and load it:

```
PUT file://gov_*.csv @PRAMANA.GOV.GOV_CONFIG_STAGE AUTO_COMPRESS = FALSE OVERWRITE = TRUE;
CALL PRAMANA.CORE.GOV_LOAD_CONFIG();
CALL PRAMANA.CORE.GOV_RUN('VALIDATE', NULL);
```

`PUT` sends files from your own machine, so it runs from the Snowflake CLI or SnowSQL and not from a Snowsight worksheet. To stay in a worksheet, upload the files to a stage of your own (Snowsight can do that from the stage's page), copy them across and load:

```
-- from a stage of yours that holds the gov_*.csv files, into the application's stage
COPY FILES INTO @PRAMANA.GOV.GOV_CONFIG_STAGE FROM @<your database>.<your schema>.<your stage>/pack/;
LIST @PRAMANA.GOV.GOV_CONFIG_STAGE;
CALL PRAMANA.CORE.GOV_LOAD_CONFIG();
CALL PRAMANA.CORE.GOV_RUN('VALIDATE', NULL);
```

The role that runs it needs `PRAMANA_ADMIN` and read access to your stage. `LIST` shows what arrived, and `GOV_LOAD_CONFIG` answers with the rows it read from each file.

Settings and controls are merged, so a file may carry only the keys you change. The other seven tables are replaced by their file. A file that is absent, or has no rows, leaves its table as it is.

If you keep the pack in git, treat the files as the master copy. An exemption added in a console exists only in the application until you add it to `gov_exemption.csv` as well; the next load of a file that has rows replaces the table with that file.

Tuning what counts as sensitive

A column reaches the review queue in one of two ways: Snowflake's classifier examined its table and assigned a category your taxonomy maps, or the column's name matches a category's `NAME_REGEX`.

- **Too many proposals.** Tighten the name rule of the category that produces them, or raise `CLASSIFY_MIN_CONFIDENCE` so that low-confidence classifier results stop counting. A single column that is not what its name suggests is better rejected in the queue than excluded by a rule.
- **Too few.** Tables the classifier has never examined produce proposals only from name rules. Run Snowflake's sensitive data classification on them and scan again. CLS-04 lists the sensitive tables with no recent classifier result.
- **A new kind of data.** Add a row to `GOV_CATEGORY`, bind its policies in `GOV_BINDING`, and run `VALIDATE`.

Bindings

A binding says which policy protects a category. A category with no binding is still governed, in that the classification controls apply to it, but no policy is required for it. `VALIDATE` warns about each unbound category, so that leaving one unprotected is a decision and not an oversight.

Working the review queue

Each proposal is a column that looks like a category and is not labelled with it. Pramana applies nothing; a steward decides. In PRAMANA_REVIEW → *Review queue*, or:

```
CALL PRAMANA.CORE.GOV_DECIDE('<proposal id>', 'APPROVE', 'confirmed with the data owner');
CALL PRAMANA.CORE.GOV_DECIDE('<proposal id>', 'REJECT',  'a merchant, not a person');
CALL PRAMANA.CORE.GOV_DECIDE_ALL('APPROVE', 'HIGH');
```

© 2026 Pravidha · Public

A rejection is remembered: Pramana treats that column as exempt from then on, and the table stops counting as sensitive because of it. An approval turns into a `SET TAG` statement the next time remediation is generated. Once a scan sees that tag on the column, Pramana closes the proposal (its status becomes `APPLIED`) and stops writing the statement.

Exemptions

An exemption accepts one finding, for one object and one control, until a date. Add it from the *Controls* page of PRAMANA_REVIEW, or:

```
CALL PRAMANA.CORE.GOV_ADD_EXEMPTION(
  'FINANCE_DB.CORE.CUSTOMER.EMAIL_HASH', 'CLS-01',
  'One-way hash of the address. Reviewed by the DPO.', '2027-06-30'::DATE);

-- withdraw it: the next evaluation raises the finding again
CALL PRAMANA.CORE.GOV_CLOSE_EXEMPTION('FINANCE_DB.CORE.CUSTOMER.EMAIL_HASH', 'CLS-01');
```

The user who makes the call is recorded as the approver. A target can be a column or a whole table; a table-level exemption covers its columns. Every exemption in force is printed in every evidence pack, so write the reason for the person who will read it there.

Closing findings

Every approved proposal and every fixable failure becomes an `ALTER TABLE … SET TAG`, `SET MASKING POLICY`, `ADD ROW ACCESS POLICY` or `REVOKE` statement. Pramana records them; you run them through your change process.

```
CALL PRAMANA.CORE.GOV_RUN('REMEDIATE', NULL);
SELECT STEP, STATUS, CONTROL_ID, TARGET, ACTION, STATEMENT FROM PRAMANA.CORE.GOV_REMEDIATION_OPEN_V ORDER BY STEP;
```

Each row has one of three statuses, and `STEP` puts them in the order a change would take them:

Status	What it is	What you do
`PROPOSED`	A statement that can run as written	Put it through your change process
`TEMPLATE`	A `CREATE MASKING POLICY` or `CREATE ROW ACCESS POLICY` for a policy your pack binds and the account does not have. The name is right, the signature comes from the columns waiting for it, and the body is a placeholder that does not run as written	Complete the body: who may see clear values, and what everyone else sees. That rule is yours to decide, so Pramana does not guess it
`BLOCKED`	The statement that attaches that policy to a column or table. It is correct, and it cannot run until the policy exists	Run it after the completed template

Who runs them. Not Pramana, and not just any role. A statement changes a table, so it needs a role that may: the owner of the table, or a role that holds the account-level privileges for the job (`APPLY TAG`, `APPLY MASKING POLICY`, `APPLY ROW ACCESS POLICY`, and `MANAGE GRANTS` for a `REVOKE`), with `USAGE` on the database and schema. Run by a role with no access to the database, a statement fails with "Database … does not exist or not authorized". That is Snowflake refusing the role, not a fault in the statement.

When a statement leaves the list. Pramana runs nothing, so it learns that you have run a statement only when a scan shows it the result. A batch is written from the latest evaluation, and an evaluation goes by the latest scan. A statement you ran an hour ago is therefore still in the next batch if no

© 2026 Pravidha · Public

scan has seen the change, and `ACCOUNT_USAGE` runs up to two or three hours behind your account. Once it has caught up, scan and evaluate again (the nightly run does both) and generate the statements once more. What has been done is no longer listed, and an approved proposal whose tag is now on the column is closed. When nothing is left the batch is empty, `GOV_REMEDIATION_OPEN_V` returns no rows and the Remediation page says so. Generate after an evaluation, not straight after a scan: a snapshot that has not been evaluated has no findings to write statements from, and the run tells you.

While the built-in sample is loaded, the statements are for the sample's fictional databases. They show what remediation looks like and cannot be run: there is no such database in your account.

If the columns waiting for one policy have more than one data type, the template says so: a masking policy takes one type, so the category needs a policy per type or a narrower binding.

What stays with a person: the body of a policy that does not exist yet (POL-04), a tag value the taxonomy does not know (CLS-02), which value of a governed tag a table should carry (RET-01, RET-02 and the controls of your own tags), a table owned by the wrong role (ACC-02), and the judgement calls (CLS-03, POL-03).

Drift

From the second run on, each snapshot is compared with the one before it. Every difference is recorded as an event; the ones that weaken your position are flagged and stay on the *Drift* page until someone acknowledges them.

```
SELECT DRIFT_ID, EVENT_TYPE, TARGET, BEFORE_VALUE, AFTER_VALUE, WEAKENING FROM PRAMANA.CORE.GOV_DRIFT_OPEN_V;

CALL PRAMANA.CORE.GOV_ACKNOWLEDGE_DRIFT('<drift id>', 'Change CHG-1042, approved by the data owner');
```

Acknowledging records who looked and what they concluded. It does not undo the change, and the event stays in every evidence pack that covers that night. The Reference lists each event and when it counts as weakening.

Controls and frameworks

The control catalogue is Pramana's; how strictly each control is graded is yours.

```
-- report a control as a warning, not a failure
UPDATE PRAMANA.GOV.GOV_CONTROL SET SEVERITY = 'WARN' WHERE CONTROL_ID = 'ACC-02';

-- switch a control off
UPDATE PRAMANA.GOV.GOV_CONTROL SET ENABLED = FALSE WHERE CONTROL_ID = 'CLS-04';
```

Evidence packs are built for the frameworks named in the `FRAMEWORKS` setting. UK GDPR and DORA are on from the start; ISO 27001 is mapped and waiting:

```
UPDATE PRAMANA.GOV.GOV_SETTING SET SETTING_VALUE = 'GDPR_UK,DORA,ISO27001' WHERE SETTING_KEY = 'FRAMEWORKS';
```

A framework of your own is a set of rows saying which control is evidence for which clause, and its name added to the same setting:

```
INSERT INTO PRAMANA.GOV.GOV_FRAMEWORK (FRAMEWORK, CONTROL_ID, CLAUSE, CLAUSE_TITLE) VALUES
  ('INTERNAL_POLICY', 'POL-01', 'DP-7', 'Personal data is masked outside production support'),
  ('INTERNAL_POLICY', 'ACC-01', 'DP-9', 'No general read access to personal data');
```

Recipes

A new source system lands. Add its database to the scope, run, work the queue, generate the statements. It appears in the next pack with its coverage.

© 2026 Pravidha · Public

A retired tag value. CLS-02 lists every column carrying a value the taxonomy does not know. Either add the value as a category, or retag: the queue proposes the taxonomy's category for such columns.

Masking applied through the tag. Pramana checks what reaches each column, however it got there. Bind that policy to the category so Pramana knows it is the expected one.

A column that is not personal data despite its name. Reject the proposal (it will not come back), or add an exemption with the reason. The second shows in every pack.

Hand remediation to the platform team. `GOV_REMEDIATION_OPEN_V` is the script, in `STEP` order; the Remediation page shows the latest batch as one block to copy. Ready statements come first, so the block runs up to the first policy template and stops there until someone has completed it.

Prove a point in time.

```
SELECT PACK_MD, PACK_SHA256
  FROM PRAMANA.GOV.GOV_EVIDENCE_PACK
 WHERE GENERATED_AT::DATE = '2026-09-30' AND FRAMEWORK = 'GDPR_UK';
```

Carry Vigil's reconciliation results. If you also run Vigil, Pravidha's data quality and reconciliation application:

```
GRANT APPLICATION ROLE VIGIL.VIGIL_EVIDENCE_READER TO APPLICATION PRAMANA;
UPDATE PRAMANA.GOV.GOV_EVIDENCE_SOURCE SET ENABLED = TRUE WHERE SOURCE_NAME = 'VIGIL';
```

Pramana stands alone without Vigil; `DQ-01` is then skipped.

Problems and requests: **support@pravidha.com**. General questions: **info@pravidha.com**.

© 2026 Pravidha · Public