pravidha

Pramana

Install and Setup

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

Document	Pramana — Install and Setup
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

Before you start
Install the application
See it work first: the built-in sample
What Pramana asks for
Set it up on your account
- Let it read the account's metadata
- Hand out the application roles
- Choose the scope and the classification tag
- Name your governed tags
- If you keep the policy pack as files

Your first run, step by step
Check your setup
Where to go next

© 2026 Pravidha · Public

For the platform engineer or account administrator who installs Pramana, and the governance engineer who configures it. It takes you from the Marketplace listing to a first evidence pack for your own account. The examples assume the application is installed as `PRAMANA`; if you chose another name, substitute it. Allow about an hour, most of it spent reading your first results.

Before you start

You need	Detail
Snowflake Enterprise Edition or higher	Tags, masking and row access policies and sensitive data classification are Enterprise features, and they are what Pramana checks. The built-in sample runs on any edition
`ACCOUNTADMIN`, once	To install the application and make three grants
A decision on scope	Which databases Pramana should examine first. One or two that hold personal data make a good start
Your classification tag, if you have one	The tag whose value says what a column is, for example `GOVERNANCE.TAGS.DATA_CLASSIFICATION`. Pramana works without one for a first look
Nothing else	No warehouse, integration or network change is needed

The order that works best: install, load the built-in sample to see every screen with something in it, clear the sample, then set Pramana up on your own account.

Install the application

Press **Get** on the Marketplace listing and Snowsight installs the application under the name you choose. To install from a worksheet instead, it takes two statements. The first accepts the listing's terms, which Snowflake requires before anything can be installed from a listing. The second installs. Both take the listing's global name: its short code, the `global_name` column of `SHOW AVAILABLE LISTINGS`, not its title.

```
CALL SYSTEM$ACCEPT_LEGAL_TERMS('DATA_EXCHANGE_LISTING', '<listing global name>');
CREATE APPLICATION PRAMANA FROM LISTING '<listing global name>';
```

Run the first as yourself: Snowflake does not take an acceptance of terms from a service user, and expects your user to have a verified e-mail address. The terms can also be accepted on the listing's page in Snowsight. Without either, the install is refused with "the terms for the associated listing … have not been accepted".

The Marketplace listing is a trial of the whole product, for the period shown on the listing. While it runs, the consoles show **TRIAL** in the sidebar. Operations and Troubleshooting says what to keep before it ends.

See it work first: the built-in sample

Pramana ships with a small sample estate inside it: two fictional databases, five tables, thirty-five columns, with governance defects planted in it. Loading it needs no grant and reads nothing of your account. It is the quickest way to see what every screen is for.

The sample's databases exist only inside Pramana. Everything you see is what Pramana would show for a real estate, but anything that names those databases is there to be read, not run: a

© 2026 Pravidha · Public

remediation statement generated from the sample would fail in a worksheet, because your account has no such database.

After installing, open **Data Products → Apps → PRAMANA** in Snowsight, choose **PRAMANA_CONSOLE**, go to **Sample** and press **Load the sample**. Or in a worksheet:

```
CALL PRAMANA.CORE.GOV_SAMPLE_LOAD();

SELECT CONTROL_ID, TITLE, STATUS, FAILS, WARNS, EXEMPT FROM PRAMANA.CORE.GOV_CONTROL_STATUS_V;
SELECT CONTROL_ID, TARGET, STATUS, MESSAGE, ACTION     FROM PRAMANA.CORE.GOV_FINDING_LATEST_V;
SELECT TARGET, PROPOSED_CATEGORY, SOURCE, REASON       FROM PRAMANA.CORE.GOV_REVIEW_QUEUE_V;
SELECT FRAMEWORK, OVERALL, PACK_SHA256                 FROM PRAMANA.CORE.GOV_EVIDENCE_LATEST_V;
```

What you should see, and why:

Control	Status	What the sample planted
CLS-01	FAIL	`CUSTOMER.DATE_OF_BIRTH` and `ACCOUNT.SORT_CODE` are not tagged. `TRANSACTION.MERCHANT_NAME` is flagged too: a false positive from a name rule, there for you to reject. `CUSTOMER.EMAIL_HASH` is *exempt*, with the reason a data protection officer gave
CLS-02	FAIL	`CLAIMANT.EMAIL` carries `EMAIL_ADDRESS`, a tag value the taxonomy no longer knows
POL-01	FAIL	`CUSTOMER.PHONE` is tagged but not masked; `CLAIMANT.MEDICAL_NOTES` is tagged as health data with no masking
POL-02	FAIL	`CLAIMANT` holds health data and lacks the row access policy that category requires
POL-04	FAIL	The pack binds a policy, `MASK_HEALTH`, that does not exist; two others exist and are used nowhere (warnings)
ACC-01	FAIL	`PUBLIC` can read `CLAIMANT`
ACC-02	WARN	`CLAIMANT` is owned by `SYSADMIN`, not a data-owner role
RET-01	FAIL	`CLAIMANT` holds health data and carries no retention class
RET-02	FAIL	`TRANSACTION` carries `KEEP_FOREVER`, a retention class the pack does not define
CLS-03, POL-03	PASS	
CLS-04, DQ-01	SKIPPED	The sample carries no classifier results and no reconciliation source; the note on each says so

Now weaken four things the way an estate drifts, and look at the drift:

```
CALL PRAMANA.CORE.GOV_SAMPLE_BREAK();

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

Four events are flagged as weakening: a masking policy removed from `CUSTOMER.EMAIL`, the row access policy removed from `CUSTOMER`, `SELECT` granted to `PUBLIC`, and a new `MOBILE_PHONE` column nobody tagged.

© 2026 Pravidha · Public

```
-- optional: the original catalogue again
CALL PRAMANA.CORE.GOV_SAMPLE_RESTORE();

-- removes the sample and everything it produced
CALL PRAMANA.CORE.GOV_SAMPLE_CLEAR();
```

Everything built from the sample is marked `SAMPLE`. While it is loaded Pramana will not scan your account, so the two never mix. Clearing it empties the history it produced and puts the default policy pack back. Once Pramana holds a snapshot of your own account, the sample can no longer be loaded into that installation: install a second copy if you want to explore it again.

What Pramana asks for

Request	Why	When
`EXECUTE TASK`, `EXECUTE MANAGED TASK`	One serverless nightly task, so Pramana needs no warehouse of its own	Granted at install. Nothing is scheduled until you start it
`OBJECT_VIEWER`, `GOVERNANCE_VIEWER`, `SECURITY_VIEWER` (SNOWFLAKE database roles)	The eight `ACCOUNT_USAGE` views Pramana reads	You grant them, before the first scan of your account
`READ SESSION`	So a review decision carries the name of the user who made it	Optional

The eight views are `TABLES`, `COLUMNS`, `TAG_REFERENCES`, `POLICY_REFERENCES`, `MASKING_POLICIES`, `ROW_ACCESS_POLICIES`, `DATA_CLASSIFICATION_LATEST` and `GRANTS_TO_ROLES`. Pramana does not ask for `IMPORTED PRIVILEGES` on the SNOWFLAKE database, for `USAGE_VIEWER` (usage and cost history), or for anything on your own databases. It creates no warehouse, database or role and makes no network call.

Security and Trust sets out what each of these allows, for a security review.

Set it up on your account

Let it read the account's metadata

As `ACCOUNTADMIN`:

```
GRANT DATABASE ROLE SNOWFLAKE.OBJECT_VIEWER     TO APPLICATION PRAMANA;
GRANT DATABASE ROLE SNOWFLAKE.GOVERNANCE_VIEWER TO APPLICATION PRAMANA;
GRANT DATABASE ROLE SNOWFLAKE.SECURITY_VIEWER   TO APPLICATION PRAMANA;

-- optional, recommended
GRANT READ SESSION ON ACCOUNT TO APPLICATION PRAMANA;
```

`CALL PRAMANA.CORE.GOV_RUN('ACCESS', NULL);` tells you at any time what Pramana can read and gives the exact `GRANT` statements still missing. The **Setup** page of PRAMANA_CONSOLE shows the same.

© 2026 Pravidha · Public

Hand out the application roles

```
GRANT APPLICATION ROLE PRAMANA.PRAMANA_ADMIN   TO ROLE <platform or governance
engineering>;
GRANT APPLICATION ROLE PRAMANA.PRAMANA_STEWARD TO ROLE <data stewards>;
GRANT APPLICATION ROLE PRAMANA.PRAMANA_AUDITOR TO ROLE <audit, risk, anyone who only
reads>;
```

Role	Can
`PRAMANA_ADMIN`	Everything: the policy pack, runs, the schedule, the sample
`PRAMANA_STEWARD`	Read everything; decide proposals, acknowledge drift, add and close exemptions
`PRAMANA_AUDITOR`	Read findings, drift, evidence packs and the policy pack
`PRAMANA_EVIDENCE_READER`	Read `CORE.PRAMANA_EVIDENCE_V` only. For a GRC tool's service role

Choose the scope and the classification tag

In **PRAMANA_CONSOLE → Setup**, pick the databases from the list and name your classification tag. Type the full name, or take one from the list of tags the account views already show; that list runs up to two hours behind, so a tag created today is typed. The same page names each governed tag (below). In SQL:

```
INSERT INTO PRAMANA.GOV.GOV_SCOPE (DATABASE_NAME) VALUES ('FINANCE_DB'), ('CLAIMS_DB');

UPDATE PRAMANA.GOV.GOV_SETTING
   SET SETTING_VALUE = 'GOVERNANCE.TAGS.DATA_CLASSIFICATION'
 WHERE SETTING_KEY = 'CLASSIFICATION_TAG';

CALL PRAMANA.CORE.GOV_RUN('VALIDATE', NULL);
```

The classification tag is the tag whose *value* says what a column is. If you do not have one yet, create it (`CREATE TAG GOVERNANCE.TAGS.DATA_CLASSIFICATION;`) or leave the setting empty for a first look: Pramana will still tell you which columns look sensitive and carry no tag.

A scope row can narrow a database: `SCHEMA_INCLUDE_REGEX` (default `.*`), `SCHEMA_EXCLUDE_REGEX` and `TABLE_EXCLUDE_REGEX` are case-insensitive regular expressions, for example `^(STAGING|TMP)$` and `^(TMP_|STG_)`.

Name your governed tags

Governed tags (optional). Beside the classification tag, which sits on columns, most organisations govern by a few tags on tables: how long the data is kept, who owns it, where it may be held. Each one Pramana should check is a row in `GOV_TAG`. No tag name is built in, and nothing has to be created by Pramana: you name the tags you already have, whatever sets them.

Retention ships as a row with no tag named. A retention class says how long a table keeps its data. Name your tag and list the classes it may carry:

© 2026 Pravidha · Public

```
UPDATE PRAMANA.GOV.GOV_TAG
   SET TAG_NAME = 'GOVERNANCE.TAGS.DATA_RETENTION'
 WHERE TAG_CLASS = 'RETENTION';

-- the rank is the number of days the class keeps data; leave it empty for a class with no fixed period
INSERT INTO PRAMANA.GOV.GOV_TAG_VALUE (TAG_CLASS, TAG_VALUE, VALUE_RANK, DESCRIPTION) VALUES
  ('RETENTION', 'STANDARD_7Y', 2555, 'Customer and financial records'),
  ('RETENTION', 'LEGAL_HOLD',  NULL, 'Kept until the hold is released');
```

From the next scan Pramana checks that every table holding sensitive data carries a retention class (RET-01) and that every class it finds is one you listed (RET-02). Leave `TAG_NAME` empty and both controls are skipped.

A tag of your own is one more row. Give it a class name (one word; it appears in findings, drift events and evidence), the tag, which tables must carry it, and the ids of the two controls it reports under:

```
INSERT INTO PRAMANA.GOV.GOV_TAG (TAG_CLASS, TAG_NAME, REQUIRED_ON, MISSING_CONTROL, VALUE_CONTROL, DESCRIPTION) VALUES
  ('RESIDENCY', 'GOVERNANCE.TAGS.DATA_RESIDENCY', 'ALL', 'RES-01', 'RES-02', 'Where the data may be held');

INSERT INTO PRAMANA.GOV.GOV_TAG_VALUE (TAG_CLASS, TAG_VALUE) VALUES ('RESIDENCY', 'UK'), ('RESIDENCY', 'EU');

CALL PRAMANA.CORE.GOV_RUN('VALIDATE', NULL);
```

`VALIDATE` adds `RES-01` ("Tables that need one carry a RESIDENCY tag") and `RES-02` ("RESIDENCY tag values are from the pack") to the control catalogue, graded `FAIL`. From then on they are controls like the others: re-grade or disable them in `GOV_CONTROL`, exempt a table in `GOV_EXEMPTION`, and map them to clauses in `GOV_FRAMEWORK` to have them in a pack's clause table. Leave either control id empty and that check is not made; list no values and any value is accepted.

Three things hold for every governed tag. The tag may sit on the table, or on its schema or database and be inherited. Pramana does not decide which value a table should carry and does not write the tag: that decision is made wherever you govern your data, and a finding sends it back there. And a tag is evaluated from the first scan after you name it: a snapshot taken before says nothing about it, so its controls show `SKIPPED`, with the reason, until the next scan.

If you keep the policy pack as files

Everything above can also be kept as CSV files, one per policy pack table, and loaded in one call. That suits a team that reviews its governance rules in git. The template pack is on the documentation page, and the steps are in the Administrator Guide under Two ways to maintain it. You can start on the Setup page and move to files later: the two routes fill the same tables.

Your first run, step by step

`CALL PRAMANA.CORE.GOV_NIGHTLY();` does all of this in one call. Taking it a step at a time once shows what each part is for.

Scan. A snapshot of the estate as Pramana sees it: tables with owners and row access policies, columns with their tag value (resolved through inheritance from table, schema or database), masking policy and classifier result, grants, and the policies your pack binds.

```
CALL PRAMANA.CORE.GOV_RUN('SCAN', NULL);
SELECT * FROM PRAMANA.CORE.GOV_COVERAGE_V;
```

`ACCOUNT_USAGE` runs up to two or three hours behind your account, so objects created or changed in the last couple of hours appear in a later scan. Where policies are attached is the slowest of these to arrive: Snowflake documents two hours for `POLICY_REFERENCES`, and for the first policies an

account attaches we have seen it take more than seven. Until that view lists an attachment, POL-01 and POL-02 report the policy as missing. If the view holds nothing at all for your account while policies exist, the scan says so in its notes and the four policy controls carry the same remark, so a finding that only reflects the delay can be told from one that does not. When the view catches up, the findings clear in the next run, and the attachments are not reported as drift: nothing changed that day.

See what it thinks is sensitive. Each proposal is a column that the classifier (if it has run on that table) or one of your name rules says is a category, and that is not tagged with it. Nothing is applied.

```
CALL PRAMANA.CORE.GOV_RUN('CLASSIFY', NULL);
SELECT TARGET, PROPOSED_CATEGORY, SOURCE, CONFIDENCE, REASON FROM PRAMANA.CORE.GOV_REVIEW_QUEUE_V;
```

If the queue is noisy, tighten `NAME_REGEX` in the taxonomy or raise `CLASSIFY_MIN_CONFIDENCE`. If it is quiet and you expected more, your tables may never have been classified: run Snowflake's sensitive data classification on them and scan again.

Evaluate the controls. Expect failures the first time. That is the point.

```
CALL PRAMANA.CORE.GOV_RUN('EVALUATE', NULL);
SELECT CONTROL_ID, TITLE, STATUS, TARGETS_CHECKED, FAILS, WARNS, EXEMPT, NOTE FROM PRAMANA.CORE.GOV_CONTROL_STATUS_V;
SELECT CONTROL_ID, STATUS, TARGET, MESSAGE, ACTION FROM PRAMANA.CORE.GOV_FINDING_LATEST_V;
```

Review. 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');
```

A rejection is remembered: Pramana treats that column as exempt from then on, and the table stops counting as sensitive because of it.

Close the 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;
```

Run them with a role that may change those tables, such as their owner; a role with no access to the database gets "does not exist or not authorized" from Snowflake. The statements come in an order a change can follow, and some wait for a policy that does not exist yet. A statement you have run stays in the batch until a scan has seen the change, which can take two or three hours; then scan, evaluate and generate again. Closing findings in the Administrator Guide explains the three kinds.

Evidence, and the night shift.

```
CALL PRAMANA.CORE.GOV_RUN('EVIDENCE', NULL);
SELECT FRAMEWORK, OVERALL, GENERATED_AT, PACK_SHA256 FROM PRAMANA.CORE.GOV_EVIDENCE_LATEST_V;

CALL PRAMANA.CORE.GOV_START();
```

From the second run on, each snapshot is compared with the one before it. Weakening events stay on the *Drift* page until someone acknowledges them.

Check your setup

Before you hand Pramana to its users, confirm each of these:

© 2026 Pravidha · Public

Check	How
Pramana can read what it needs	`CALL PRAMANA.CORE.GOV_RUN('ACCESS', NULL);` lists nothing as missing
The policy pack is consistent	`CALL PRAMANA.CORE.GOV_RUN('VALIDATE', NULL);` reports no problems
The scan found your tables	`SELECT * FROM PRAMANA.CORE.GOV_COVERAGE_V;` shows each database in scope
Decisions carry a name	Take one decision in PRAMANA_REVIEW and check it does not read `UNRECORDED`
The schedule is running	`CALL PRAMANA.CORE.GOV_STATUS();` shows the task started
Each role sees the right console	A steward opens PRAMANA_REVIEW; an auditor opens PRAMANA_VIEWER and can change nothing

Where to go next

You want to	Read
Maintain the policy pack, work the queue, close findings	Administrator Guide
Look up a setting, a view, a procedure or a control	Reference
Keep it running, or work out why something looks wrong	Operations and Troubleshooting
Explain the results to audit or compliance	Evidence Guide

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

© 2026 Pravidha · Public