

— PERMISSION-AWARE RETRIEVAL

Access Control and Metadata

How Corporate GPT represents, propagates, and enforces access control lists, so every retrieval **honours the permissions of the person asking**

AUTHOR

Neurologic AI

PUBLISHED

September 2026

PLATFORM

Corporate GPT

ABSTRACT

Corporate GPT answers questions from an organization's own documents, tickets, and web pages. It is only safe to deploy if every retrieval honours the permissions of the person asking. This paper describes how Corporate GPT represents, propagates, and enforces access control lists (ACLs), and how the same payload metadata that carries those ACLs also carries document structure, provenance, and governance signals. We cover the principal model (users, roles, groups

synchronized from Active Directory, and user-owned projects); the two-sided ACL model, in which PostgreSQL is the source of truth and every vector in Qdrant carries a denormalized copy of its document's ACL; query-time enforcement pushed down into the vector store as a fail-closed payload filter composed with document selection and project isolation; an asynchronous, resumable, and rollback-capable permission-synchronization pipeline; the extension of the same ACL model to connector corpora (Jira, Zoho Desk), with an explicit trust boundary between source data and visibility; and the chunk-level metadata schema that makes citations, figure rendering, and structured filtering possible. We close with the known limitations and the hardening roadmap.

The design goals throughout are:

Authorization by construction

No retrieval path without a filter.

A single source of truth

The relational database decides, the index follows.

Fail-closed behaviour

An error narrows visibility, never widens it.

01

Introduction

A retrieval-augmented assistant is, among other things, a very efficient search engine over everything it has indexed. In an enterprise that is a liability as much as a feature: a single unfiltered vector search can surface a salary sheet, a board memo, or another team's customer tickets to anyone who asks the right question. Post-filtering retrieved chunks in application code is not sufficient either. It is easy to forget on a new code path; it leaks through ranking (a filtered-out document still displaced an authorized one from the top-k); and it offers no protection when a bug skips the filter altogether.

Corporate GPT treats access control as **a property of the data**, not of the request handler.

Corporate GPT therefore treats access control as a property of the data, not of the request handler. Every point written to the vector store carries the ACL of the document it came from. Every query carries a filter derived from the caller's identity. The retrieval layer refuses to run without one. The ACL is materialized from relational tables at ingestion time and resynchronized whenever permissions change, so the index never becomes the authority on who may see what.

The same mechanism, a flat payload stored alongside each vector, also carries the metadata that the rest of the system depends on: document identity, page and section structure, figure bounding boxes, website provenance and staleness, PII-redaction status, and project scope. Metadata and ACL are therefore two uses of one design, and this paper treats them together.

The remainder of this paper documents the implementation as it exists in the current codebase: the principal model (Section 3), the document ACL and how it is built (Sections 4 and 5), query-time enforcement (Section 6), permission changes (Section 7), project isolation (Section 8), connector corpora (Section 9), the metadata schema (Section 10), reuse of the pattern beyond documents (Section 11), and limitations (Section 12).

02

Design goals and threat model

The ACL subsystem is designed against four failure modes:

Cross-principal leakage

A user retrieves, is cited, or is shown a file listing entry for a document they are not entitled to.

Stale grants

Access is revoked in the application (a user leaves a group, a group is deleted, a connector is made private) but the vector index still carries the old ACL.

Self-publishing source data

An external system (for example a ticketing API) returns a field that, if copied verbatim into the payload, would set the document's own visibility.

Anonymous exposure

An embeddable widget running on a third-party origin, with no individual identity behind it, reaches data meant for authenticated employees.

Three rules follow from these and recur throughout the implementation:

1

Enforcement happens inside Qdrant.

Filters are pushed down as payload conditions, so unauthorized points are never scored or returned. There is no Python-side post-filtering of chunks.

2

PostgreSQL is the source of truth.

`document.ac1` is rebuilt from relational grants and is never hand-edited. The Qdrant payload is a denormalized copy of it.

3

A minimum safe filter is always applied.

If the database cannot be reached at query time, the filter degrades to "public documents, plus documents owned by this user". It never degrades to "no filter".

03

Principals and the authorization model

3.1 Users and roles

Each user carries exactly one role (`UserRole`). Two families of roles affect document visibility: `ADMIN` and `GLOBAL_CURATOR` unlock documents flagged admin-public, and `CURATOR` and `GLOBAL_CURATOR` unlock documents flagged curator-public. The remaining roles gain no document visibility from the role itself.

Role	Admin-public docs	Curator-public docs
admin	Yes	No

Role	Admin-public docs	Curator-public docs
<code>global_curator</code>	Yes	Yes
<code>curator</code>	No	Yes
<code>basic</code> , <code>limited</code>	No	No
<code>slack_user</code> , <code>ext_perm_user</code>	No	No

Table 1. Role-derived document visibility. Every role additionally sees public documents, its own documents, and documents granted to its groups.

Administrative surfaces (the Permissions API, group management, dashboards, chat and ingestion logs) are mounted behind a `require_admin_user` dependency at the router level, so authorization for managing ACLs is enforced before any handler code runs.

3.2 Groups

Groups (`user_group`) are the main unit of shared access. Membership lives in `user_user_group` . Each group row records its provenance (`GroupSource`):

MANUAL

Created and maintained by an administrator in the application.

AD

Synchronized one-to-one from an Active Directory group just in time, at every login, when `AD_GROUP_SYNC_ENABLED` is set. Group names arrive either from the `groups` claim of an Okta OIDC token (AD groups mirrored into Okta) or from the `memberOf` attribute during a direct LDAP bind. AD-sourced groups are read-only in the application because membership is owned by the directory.

AD synchronization contains two safeguards that matter for authorization:

- **Name collisions are refused.** If a *manual* group already holds the name of an incoming AD group, the user is *not* linked into it. The two populations may be unrelated, and because group membership *is* document authorization, silently merging them would grant access that the reconcile pass (which only touches AD-sourced groups) would never revoke. The conflict is logged for an administrator to resolve.
- **Revocation only runs on a clean sync.** Memberships in AD groups the user no longer belongs to are removed only when every group in the login's list was processed without error. A partial list would otherwise strip access to documents gated by a group that merely failed to process.

AD groups share the ordinary group namespace, so they flow into the ACL and retrieval pipeline with no special casing. This is what makes permission-aware retrieval work directly off the corporate directory.

3.3 Projects

A *project* is a user-owned collaboration space, stored in the same `user_group` table with `kind = 'project'` (administrative groups use `kind = 'group'`). Project ownership is recorded as `user_user_group.is_curator`. Attaching an access group to a project snapshot-expands its members into project memberships. The table `project_group_member` records which memberships came from which attached group, so detaching a group removes exactly the members it brought in and leaves directly invited members untouched. Because a project is a group, the ACL machinery described below needs no separate code path to authorize project members. Section 8 describes the additional isolation layer.

3.4 Connector-credential pairs as the unit of permission

Documents enter the system through a *connector-credential pair* (cc-pair): a directory, an upload channel, or an external source bound to a credential. Administrators grant access at the cc-pair level:

- group grants in `user_group_connector_credential_pair` (with an `is_current` flag);
- individual user grants in `user_connector_credential_pair`;
- role-wide flags on the credential: `Credential.admin_public` and `Credential.curator_public`;

- a public/private `access_type` and an aggregated `acl` snapshot (plus `previous_acl` during updates) on the cc-pair itself.

Each grant row also records a per-entry access type for the administration UI. Retrieval visibility itself is binary: a principal either can or cannot see a document.

04

The document ACL

4.1 Shape

Every indexed document has an `acl` column (PostgreSQL `JSONB`) with a fixed shape:

Field	Meaning
<code>user_ids</code>	Users granted access: primary owners, secondary owners, explicit per-document grants, and cc-pair user grants.
<code>user_group_ids</code>	Groups granted access: cc-pair group grants (<code>is_current = true</code>) and explicit per-document group grants (including project groups).
<code>is_public</code>	Anyone may access. Supersedes every other grant.
<code>is_admin_public</code>	Users with role <code>ADMIN</code> or <code>GLOBAL_CURATOR</code> may access.
<code>is_curator_public</code>	Users with role <code>CURATOR</code> or <code>GLOBAL_CURATOR</code> may access.

Table 2. The `document.acl` object.

A GIN index with `jsonb_path_ops` on `document.acl` makes containment (`@>`) queries over this column index-backed. A `NULL` ACL identifies a document ingested before the ACL migration (February 2026). Those documents are served through a legacy path that reads `is_public` , `primary_owners` , `secondary_owners` , and `external_user_group_ids` directly.

4.2 Relational mirrors

Two junction tables mirror the ACL relationally: `document_by_user` (one row per user in `acl.user_ids`) and `document_by_user_group` (explicit document-to-group grants). They support ACL reconstruction and audit queries ("who can see document X?") without

touching the vector store, and both cascade on document deletion.

4.3 Aggregation: `build_document_acl`

The ACL is never written by hand. It is computed by `build_document_acl(document_id)`, which walks every authoritative source:

```
user_ids      ← primary_owners ∪ secondary_owners ∪ DocumentByUser
               ∪ UserConnectorCredentialPair(current)
user_group_ids ← UserGroupConnectorCredentialPair(current) ∪
DocumentByUserGroup
is_admin_public ← ∃ linked credential with admin_public
is_curator_public ← ∃ linked credential with curator_public
is_public      ← Document.is_public
```

Figure 1. How `build_document_acl` aggregates each field from its authoritative sources.

The credential- and pair-level sources are reached through `document_by_connector_credential_pair`, so a document that belongs to several cc-pairs accumulates the union of their grants.

COERCION RULE

If `is_public` is true, every other field is cleared: `user_ids` and `user_group_ids` become empty and both role flags become false. Public access is already broader than any grant, so keeping the grants would only create stale state that could re-activate unexpectedly if the document were later made private. The rule is enforced at every entry point that can set visibility: directory registration, the parsing step, `build_document_acl`, the Permissions API handler, and the permission-update task.

05

ACL lifecycle through ingestion

Documents are ingested asynchronously by Celery workers. The ACL is established in four stages.

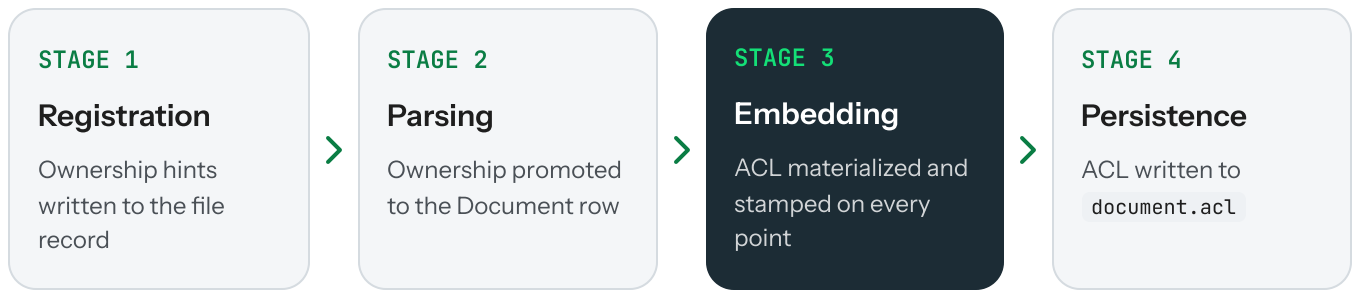


Figure 2. The four stages that establish a document's ACL.

Stage 1: Registration

Ownership hints are written into `FileRecord.additional_metadata`. A user upload is always private (`is_public = false`) and names the uploader as its primary owner. A directory registration carries the administrator-chosen public/admin-public/curator-public flags, with the coercion rule applied immediately, and records the registering user as primary owner. For auto-synced sources, the ACL slice is derived from the cc-pair's ACL snapshot, falling back to its access type and credential flags. Without this step, newly synced chunks from a public cc-pair would land with `acl_is_public = false` and be invisible to everyone.

Stage 2: Parsing

The parsing task promotes the ownership fields onto the `Document` row (`is_public`, `primary_owners`) and re-applies the coercion rule in case the metadata arrived without it. The full ACL is not built yet.

Stage 3: Embedding (ACL materialization)

Immediately before vectors are written:

1. If the file was uploaded into a project, a `document_by_user_group` grant for the project group is inserted, so that every project member, not only the uploader, passes the ACL check.
2. `build_document_acl` computes the ACL from all sources.
3. The five fields are stamped onto the chunk payload as `acl_user_ids`, `acl_user_group_ids`, `acl_is_public`, `acl_is_admin_public`, and `acl_is_curator_public`, together with `project_id` (sentinel `0` for personal documents).

4. Any raw registration keys (`is_public` , `is_admin_public` , `is_curator_public` , `primary_owners` , `secondary_owners`) are stripped from node metadata, so only the authoritative `acl_*` fields reach the index.
5. Existing points for the same `document_id` are deleted from both the main and summary collections before insertion, which makes re-embedding idempotent.

The same ACL and project fields are written to the document's entry in the parallel summary collection (`file_system_summary`), so file-level search is filtered identically to chunk-level search.

Stage 4: Persistence

After the vectors are stored, the ACL is written to `document.acl` and `document_by_user` is rebuilt by deleting and re-inserting, so an owner change on re-embedding never leaves a stale grant behind.

A cooperative pause/stop check runs just before the vector write, the point of no return. A cancelled ingestion therefore never leaves vectors without a matching relational record.

— 06

Query-time enforcement

6.1 Building the filter

For every document query and file search, `get_user_acl_filters(user_id)` builds a LlamaIndex `MetadataFilters` object that is translated into a Qdrant payload filter. It combines, with OR semantics (a Qdrant `should` clause):

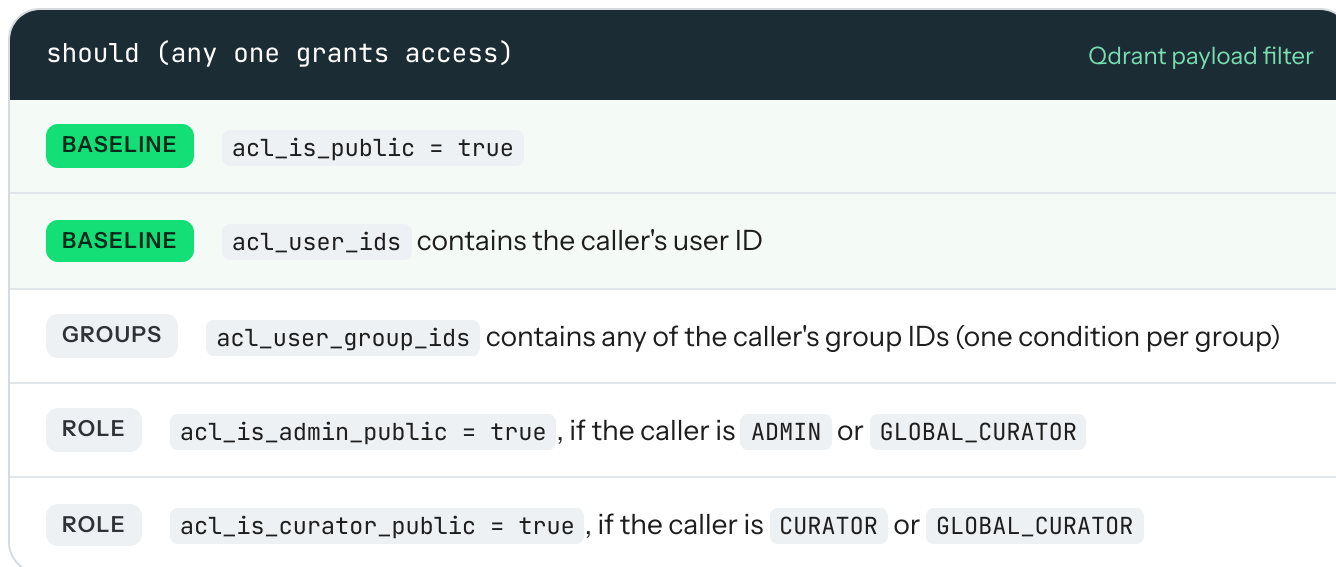


Figure 3. The OR group built by `get_user_acl_filters` . The two baseline arms are built before any database access.

Odrant's equality match on an array field is a membership test, which is what makes per-user and per-group filtering native to the vector store.

Fail-closed baseline

The first two conditions (public, and "owned by me") are constructed *before* any database access. If the group or role lookup fails, the function logs the error and returns only the baseline, so an outage narrows visibility instead of opening the corpus.

No filter, no query

The retriever builder and the summary-search method raise an error if they are called with `acl_filters = None` . An unfiltered document query is therefore a crash, not a silent default. The RAG tool also refuses to run when the session state carries no `user_id` .

6.2 Composition with selection and project scope

Corporate GPT uses a two-stage *preview* → *select* → *answer* retrieval flow. When the user has selected specific documents, and when project isolation is enabled, further conditions are combined with AND:

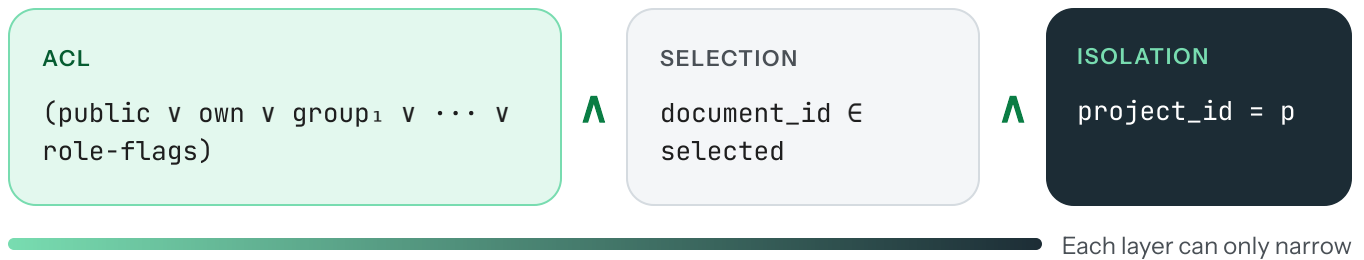


Figure 4. The full filter, pushed into Qdrant as one expression.

Each layer can only *narrow* what the one before it permits. Selection cannot reach a document the ACL forbids, and project scope cannot widen the ACL. Because the whole expression is pushed into Qdrant, both the dense and sparse (BM25) channels of hybrid search, and the candidate pool used by LLM reranking in deep-search mode, are drawn only from authorized points.

6.3 The relational path: listings and point checks

File listing and file search endpoints run against PostgreSQL only.

`build_accessible_documents_query` generates a dual-path predicate over non-hidden documents:

- **ACL path** (`acl IS NOT NULL`): JSONB containment tests, for example `acl @> {"user_ids": ["<uid>"]}`, one per group and role flag. These are served by the GIN index.
- **Legacy path** (`acl IS NULL`): the pre-migration columns `is_public`, `primary_owners`, `secondary_owners`, and `external_user_group_ids`.

Documents uploaded into a project are excluded from the personal library, the relational counterpart of the Qdrant `project_id` isolation. Files that are still processing are shown only to the user who uploaded them. The programmatic check `check_document_access` applies the same rules to a single document, and `check_file_access_by_path` resolves a file path to its document (falling back to upload ownership while the file is still being processed).

Permission changes and synchronization

Because every vector carries a copy of its document's ACL, changing permissions means rewriting payloads on potentially millions of points. Corporate GPT treats this as a durable, background, resumable job.

7.1 The Permissions API

`PUT /permissions/{cc_pair_id}` (administrators only) replaces a cc-pair's public flags, group grants, and user grants. The handler:

- 409** Rejects the change while ingestion is in progress on the pair.
- 200** Applies the coercion rule and returns with no job if the requested state equals the current ACL.
- 409** Rejects if a permission job is already pending or in progress. A partial unique index on active jobs per cc-pair turns a concurrent race into the same 409.
- 503** Verifies that the Celery broker is reachable and that a worker is consuming the permissions queue (503 otherwise), so a change is never accepted into a queue nobody drains.
- 202** Creates a `PermissionJob` and dispatches it atomically, returning the job ID for polling.

Personal upload pairs (`is_user_file = true`) are excluded from this API. Their ACL is owned by the uploader and, where applicable, the project.

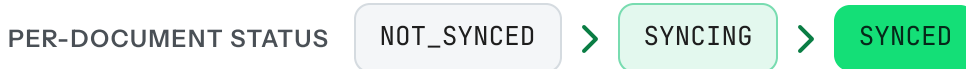
7.2 The synchronization algorithm

The `update_cc_permission` task, on the dedicated `permissions` queue, performs the update:

1. **Relational update.** The credential's role flags are updated, and the group and user grant rows are upserted to match the request (rows no longer requested are deleted).
2. **Snapshot swap.** The current ACL is saved to `cc_pair.previous_acl`, the new ACL is written to `cc_pair_acl`, and every document of the pair is marked `NOT_SYNCED`.

3. **Batched sweep.** In batches of `QDRANT_PERMISSION_BATCH_SIZE` (default 100), documents move `NOT_SYNCED` → `SYNCING`. Their `document_by_user_group` rows are rewritten, `document.acl` and `is_public` are updated, the five `acl_*` payload fields are overwritten with Qdrant `set_payload` on every point of those documents in both the main and summary collections, and the batch is committed as `SYNCED`. Group IDs that no longer exist are dropped defensively before the write.

4. **Completion.** `previous_acl` is cleared and the job is marked `COMPLETED`.



A pair-level update overwrites each affected document's ACL with the pair's ACL. The cc-pair, not the individual document, is the unit of permission for connector-sourced content.

Resumability. Per-document sync status is the work queue. On a soft time limit the task schedules a continuation with the same arguments. Documents left in `SYNCING` by an interrupted run are reset to `NOT_SYNCED`, and an already-set `previous_acl` tells the continuation to skip the swap and resume the sweep.

Rollback. Any other failure schedules `rollback_cc_permission`, which restores `previous_acl`, re-marks the pair's documents, and replays the sweep with the restored ACL. Rollback is itself idempotent and resumable, and retries up to three times before marking the job `ROLLBACK_FAILED`.

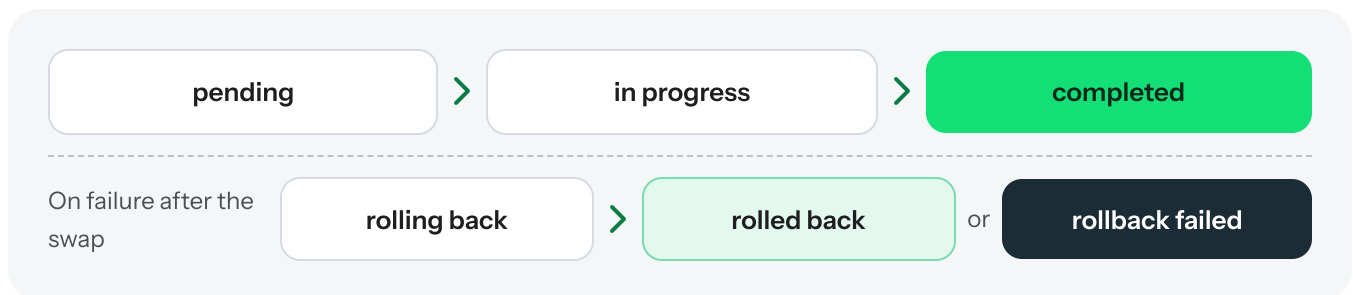


Figure 5. How a `PermissionJob` moves between states.

Job status	Meaning
pending	Accepted and queued.
in progress	Sweep running.
completed	All documents synced; API reflects new grants.

Job status	Meaning
rolling back	Update failed after the swap; restore in progress.
rolled back	Previous ACL restored on every document, or the update failed before any swap.
rollback failed	Restore retries exhausted; operator attention required.

Table 3. `PermissionJob` lifecycle.

7.3 The durable backstop

Registering a directory can also change a pair's ACL. That work is dispatched as `sync_cc_pair_document_acl`, which uses the same batched sweep with exponential-backoff auto-retry (up to five attempts). A Celery Beat task, `redispatch_stale_acl_syncs`, runs every 30 minutes and re-dispatches the sweep for any pair that still has `NOT_SYNCED` documents. It skips pairs that an active permission job currently owns, so a backstop sweep can never overwrite a rollback with a superseded, possibly wider, ACL. A partial index on the `NOT_SYNCED` predicate keeps this scan cheap in the steady state, where almost every row is `SYNCED`.

7.4 Deleting principals

Deleting a group or user is a two-phase operation. The endpoint marks the principal for deletion (for users, a soft delete that also deactivates the account), then `purge_group` or `purge_user` removes it from every cc-pair ACL through the same tracked, resumable update, and only then deletes the row. Purging a project additionally deletes its chat sessions, files, memberships, and activity log, and removes the project's vectors from both collections *after* the SQL commit, so a failed commit never leaves the database pointing at vectors that no longer exist.

7.5 Directory overlap

A file path belongs to exactly one connector. When a newly registered directory overlaps an existing one, the merge re-points documents instead of re-ingesting them, and the surviving parent's ACL is applied to every absorbed document. Because this can widen access, both the dry-run plan and the merge report the grant delta explicitly.

Projects: isolation on top of authorization

Projects need two separate properties, and the implementation keeps them in two separate mechanisms:

AUTHORIZATION

Every project member may read the project's files

Achieved with ordinary ACL machinery: a project upload receives a `document_by_user_group` grant for the project group, which flows into `acl_user_group_ids`.

ISOLATION

Project data stays in the project

A project chat sees only that project's documents, and project documents never appear in personal or group chats, even for a user who is authorized to read them. Achieved with the `project_id` payload field and the AND-composed scope filter.

A project-scoped chat session can only be created by an actual member of the project. The check is strict: a site administrator can manage a project but cannot open a chat in it without being added. The scope is persisted on the session and seeded into agent state, and personal chats use the sentinel `0`. The scope is fail-closed: an unresolved scope defaults to `0`, so a plumbing bug hides project data instead of leaking it.

Rollout ordering

A Qdrant equality match excludes points that lack the key. Enabling the filter before every pre-existing point carries `project_id` would make the personal scope match nothing. The filter is therefore gated behind `PROJECT_SCOPE_FILTER_ENABLED` (default off), to be enabled only after the idempotent `scripts/backfill_project_id.py` has stamped `project_id = 0` on every legacy point in both collections.

Connector corpora

Content from external systems (Jira issues, Zoho Desk tickets) is indexed into per-connector Qdrant collections by the connector platform and searched by dedicated agent tools. The connector path applies the same ACL vocabulary as the document path, deliberately using the same `acl_*` field names, in two layers.

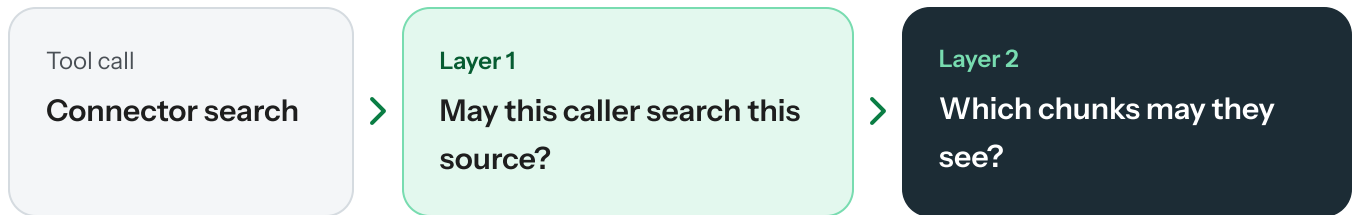


Figure 6. Two layers of connector authorization.

9.1 Layer 1: May this caller search this source?

`resolve_connector_access` decides, per tool call:

- **Anonymous widget sessions** are denied unless the connector is listed in `CONNECTOR_WIDGET_ALLOWED`. Even then the opt-in is refused while `CONNECTOR_DEFAULT_ACL_PUBLIC` is true, because "public" would then mean the entire corpus. An allowed widget session gets a public-only scope that ignores the widget service account's own group memberships and requires an explicit public stamp.
- **Identity.** A missing or malformed user ID is denied.
- **Connector-type scope.** The caller must pass the per-type grant (`connector_type_user`, `connector_type_user_group`, and the admins/curators flags), and the type's `enabled` kill switch must be on. Search scopes use a separate key namespace (`connector_search:<name>`), so granting or disabling a credential type never silently changes who may search that source.
- **Database failure** is denied with a distinct reason.

Every denial returns an explicit message instructing the model to tell the user the source is unavailable, not that nothing was found. An authoritative-sounding "no results" would be a false negative answer.

9.2 Layer 2: Which chunks may they see?

For a permitted caller, a `ConnectorACL` mirroring the document ACL (own ID, groups, public, and role-gated admin/curator arms) is AND-ed into every connector search and lookup as a nested `should` group. The search function takes the ACL as a keyword-only argument with no default, so no call site can omit it by accident. Opting out requires passing `None` explicitly, which is reserved for administrative tooling. An ACL that grants nothing returns an empty result instead of an unfiltered query.

9.3 Trust boundary and defaults

Connector metadata is built from data returned by the external system, so it must never be able to set its own visibility. `strip_raw_acl` removes every `acl_*` key from source metadata before the payload is built. The authoritative fields are then stamped last, from connector code (`Document.acl`) or from deployment defaults. Values are normalized (booleans and string lists), and a JSON `null` is treated as "unspecified" rather than "false", which would otherwise make a chunk invisible to everyone.

Connector corpora are organization-wide by nature, so the shipped default stamps `acl_is_public = true` . Operators can restrict a source with `CONNECTOR_DEFAULT_ACL_PUBLIC=false` and `CONNECTOR_DEFAULT_ACL_GROUP_IDS` . Points ingested before ACL stamping carry no `acl_*` keys, so the public arm also matches `IsEmpty(acl_is_public)` for authenticated callers during upgrade. After a full re-ingest, `CONNECTOR_ACL_REQUIRE_STAMP=true` removes that arm.

The connector store creates payload indexes (`KEYWORD` on `source` , `document_id` , `acl_user_ids` , `acl_user_group_ids` ; `BOOL` on `acl_is_public`) idempotently on every ingest, so ACL filtering stays index-backed as collections grow.

— 10

Metadata: the payload schema

Every point in the main document collection carries a flat JSON payload. Beyond the ACL, this payload is what lets the system cite sources, render figures, distinguish websites from files, and scope retrieval. Table 4 groups the fields by purpose.

Group	Fields and purpose
Identity	<code>document_id</code> (the join key to PostgreSQL; <code>file_record_<id></code> for files), <code>doc_id</code> , <code>file_name</code> , <code>file_path</code> , <code>file_type</code> , <code>doc_count</code> .
Structure	<code>section</code> (body, table, figure, chart, image description), <code>page_label</code> , <code>headings</code> (breadcrumb, also prefixed to chunk text), <code>caption</code> , <code>object_index</code> , <code>table_part</code> (e.g. 2/3 for split tables), <code>language</code> (code files).
Figures	<code>bbox</code> (normalized to a bottom-left origin at ingestion for precise cropping), <code>type</code> (layout label), <code>detector</code> .
Web provenance	<code>page_url</code> , <code>page_title</code> ; <code>is_stale</code> and <code>stale_detected_at</code> set in place when a crawled page is detected as stale.
Governance	<code>acl_user_ids</code> , <code>acl_user_group_ids</code> , <code>acl_is_public</code> , <code>acl_is_admin_public</code> , <code>acl_is_curator_public</code> , <code>project_id</code> , <code>pii_anonymized</code> (true when the PII audit log shows redactions for this document).

Table 4. Chunk payload fields in the document collection.

The summary collection holds one point per document with a smaller payload: `file_name`, `file_path`, `file_type`, `document_id`, `summary_updated_at`, and the same ACL and project fields. The LLM-generated summary itself is the embedded text, and it is deliberately excluded from the chunk payload.

10.1 Metadata hygiene

- **Type sanitization.** `sanitize_meta` drops `None` values and coerces nested structures into JSON primitives, so every payload is filterable and serializable.
- **Single authoritative ACL.** Raw registration fields are stripped before the `acl_*` fields are stamped (Section 5), so a chunk never carries two conflicting statements of its visibility.
- **Derived presentation fields.** At query time each source is given a `source_type` (`website` if `page_url` is present; `image` for figure or image-description sections or image MIME types; `audio`; otherwise `file`) and a `file_url`. The UI can therefore render citations, inline figures, and web links without re-reading the source.

- **Payload indexes.** The main collection has `KEYWORD` indexes on `doc_id` , `document_id` , `section` , and `file_path` , and the summary collection on `doc_id` and `document_id` . These back deletion-by-document, ACL sweeps (which select points by `document_id`), and the figure-inventory scroll that retrieves all figures of a matched document without a vector search.

10.2 Metadata as a query language for connectors

For ticketing sources, metadata is also the primary way users query. Connector metadata is built from explicit allowlists. For example, a Zoho Desk ticket yields `ticket_id` , `subject` , `status` , `priority` , `department` , `assignee` , contact e-mail, `tags` , created and modified times, and `ticket_url` . A lightweight, rule-based (no LLM) query-understanding layer turns natural language into structured payload filters using a vocabulary each connector supplies:

Query	Filters	Residual semantic text
"tickets assigned to user3"	<code>assignee = user3</code>	(none)
"high priority authentication bugs"	<code>priority = High</code>	"authentication bugs"

Priority and label detection is *cue-gated*: "high" only counts next to a priority cue, and a label only counts after "tagged" or "label:". Semantic phrases such as "high latency" or "security incident" therefore remain semantic. Purely structural queries become filter-only scrolls, and mixed queries run a vector search inside the filter. In both cases the ACL group is AND-ed on top.

11

Beyond documents: one ACL pattern

The document ACL's shape (an aggregated JSONB snapshot with a GIN index, a `previous_acl` for safe swaps, and junction tables as the relational truth) has been reused for other shared resources:

Agent Builder resources

Agents and workflows store an ACL snapshot of every principal with any access. Visibility is decided by JSONB containment, and the effective level (view, run, edit) is resolved by a ladder: built-in resources are run-only; owners edit; administrators edit published resources; otherwise the highest explicit user or group grant applies. Snapshot rebuilds are serialized per resource with a PostgreSQL advisory lock.

Model access

LLM slot assignments were migrated from per-scope rows to the same hybrid of a global flag plus an ACL snapshot and user and group junction tables.

Connector types

Per-type user and group grants with admin and curator flags gate the SQL agent, credential checkout, OAuth connection, and connector search.

Reusing one vocabulary means one mental model for administrators and one set of review rules for engineers.

12

Known limitations and roadmap

The following items are either known gaps in the current implementation or planned work. They are listed so that deployments and reviewers can account for them.

Payload indexes for ACL fields on the document collection

The connector store indexes its `acl_*` fields, but the main document collection currently indexes only identity and structure fields. The ACL filter is still applied correctly by Qdrant; indexing `acl_user_ids`, `acl_user_group_ids`, the three boolean flags, and `project_id` would keep filtered search fast as the corpus grows.

Governance metadata in embedded text

Chunk metadata is attached through LlamaIndex nodes whose `excluded_embed_metadata_keys` are not set. By LlamaIndex's default, metadata key-value pairs, including the `acl_*` ID lists and `project_id`, are therefore concatenated into the text used for dense and BM25 embedding. This does not affect authorization, which is enforced by the payload filter, but it adds non-semantic tokens to the representations. Excluding governance and path fields from embedding (and re-indexing) is planned.

Figure inventory re-filtering

The figure-inventory scroll selects figures by `document_id` or `file_path` of a document that was already returned by an ACL-filtered retrieval. It inherits authorization from that result instead of re-applying the ACL filter. Adding the filter to the scroll would make this path defense-in-depth.

Role flags in path-based checks

`check_file_access_by_path` does not pass the caller's role to `check_document_access`, so admin-public and curator-public documents are not recognized on that path. This fails closed (narrower access), but it is inconsistent with the retrieval filter.

Project scope rollout

`PROJECT_SCOPE_FILTER_ENABLED` is off by default until the `project_id` backfill has run on each deployment. Until then, project *authorization* is enforced but project *isolation* in chat retrieval is not.

External permission sync **FUTURE**

The schema anticipates mirroring permissions from source systems (an `AccessType.SYNC` mode, an `ext_perm_user` role, external user emails on documents), but no connector populates them yet. Today, connector visibility is set by deployment defaults or connector code.

Widget document exposure **FUTURE**

Embedded widget sessions currently retrieve documents with the ACL of a dedicated, administrator-provisioned widget user. An explicit `widget_exposed` payload condition for widget sessions is planned.

Per-document overrides on connector content

Because a cc-pair update rewrites each document's ACL to the pair's ACL, individual documents within a connector cannot hold narrower or wider grants that survive a pair-level change.

Legacy dual path

Documents ingested before the ACL migration are served by the legacy relational predicate. Backfilling `document.acl` for them would allow that path to be retired.

Reserved visibility flags on shared resources

The public, admin-public, and curator-public flags in Agent Builder ACL snapshots are reserved and always false until a visibility-control feature is added.

13

End-to-end request flow

Putting the pieces together, an authorized document question proceeds as follows:

- 1 Authenticate.** The user authenticates (local, Okta, or LDAP). On SSO login, AD groups are synchronized into RBAC groups.
- 2 Open a session.** A chat session is created. For a project chat, strict membership is verified and `project_id` is seeded into session state; otherwise the scope is `0`.
- 3 Delegate.** The orchestrator delegates to the RAG agent, whose retrieval tool reads the `user_id` from session state and refuses to run without it.
- 4 Build the filter.** `get_user_acl_filters` builds the OR group from the user's groups and role, falling back to the baseline on any database error. Document selection and project scope are AND-ed on.
- 5 Retrieve.** The retriever, which refuses a `None` filter, runs hybrid dense and sparse search in Qdrant with the filter pushed down. Only authorized points are scored.
- 6 Cite.** Sources are returned with their metadata (page, section, bounding box, source type) for citation and inline rendering. The file list the user browses is computed from

PostgreSQL with the same rules.

7

Stay current. If an administrator later changes permissions, a tracked job rewrites the ACL in PostgreSQL and on every affected point, and the next query sees the new state.

14

Design principles

01 Authorization by construction

Every retrieval path requires an explicit ACL argument. A missing filter is an error, and the connector search function has no default for it. Filters live in the vector store, not in post-processing.

02 One source of truth

Grants live in relational tables; `document.accl` and the Odrant payload are derived copies, rebuilt rather than edited, and resynchronized by a durable job with an idempotent backstop.

03 Fail closed, report honestly

Database errors narrow visibility to the safe baseline; unresolved project scope defaults to personal; connector denials are reported as denials, never as empty results.

04 Keep visibility out of source data

Only code decides who can see a chunk. Raw ACL-like keys from registration metadata or external APIs are stripped before the authoritative fields are stamped.

05 Metadata is infrastructure

The payload that carries the ACL also carries the identity, structure, and provenance that citations, figure rendering, staleness handling, and structured filtering depend on, and it is sanitized and indexed accordingly.

Conclusion

Access control in a retrieval system is only as strong as the weakest path to the index. Corporate GPT closes those paths structurally. ACLs are materialized from the relational source of truth onto every vector. Every query carries a fail-closed filter that Qdrant evaluates before scoring. Permission changes propagate through a resumable, rollback-capable job, and the same model extends to external connector corpora with an explicit trust boundary. Because the ACL travels in the same metadata payload that carries document structure and provenance, authorization, citation, and scoping are served by one mechanism.

THE RESULT

An assistant whose answers are drawn only from **what each user is entitled to see.**

APPENDIX A

Configuration reference

Variable	Default	Effect
<code>QDRANT_COLLECTION_PREFIX</code>	<code>file_system</code>	Main collection; the summary collection is <code><prefix>_summary</code> .
<code>QDRANT_PERMISSION_BATCH_SIZE</code>	<code>100</code>	Documents per ACL-sync batch.
<code>STRICT_CELERY_WORKER_CHECK</code>	<code>true</code>	Reject permission updates when no worker responds.
<code>PROJECT_SCOPE_FILTER_ENABLED</code>	<code>false</code>	AND <code>project_id</code> into retrieval (enable after backfill).

Variable	Default	Effect
<code>AD_GROUP_SYNC_ENABLED</code>	<code>false</code>	JIT sync of AD groups at login.
<code>CONNECTOR_ACL_FILTER_ENABLED</code>	<code>true</code>	Per-user payload filtering on connector search.
<code>CONNECTOR_ACL_REQUIRE_STAMP</code>	<code>false</code>	Exclude unstamped legacy connector points.
<code>CONNECTOR_DEFAULT_ACL_PUBLIC</code>	<code>true</code>	Default <code>acl_is_public</code> for connector chunks.
<code>CONNECTOR_DEFAULT_ACL_GROUP_IDS</code>	(empty)	Default group grants for connector chunks.
<code>CONNECTOR_WIDGET_ALLOWED</code>	(empty)	Connectors searchable from anonymous widget sessions.
<code>WIDGET_USER_EMAIL</code>	<code>widget-cgpt@widget.internal</code>	Service identity whose ACL scopes widget document retrieval.

Table 5. ACL-relevant configuration.