Customer Isolation Policy

How we keep one customer's data separate from every other customer's data at the database, token, and application layers.

Last updated May 29, 2026

Scope

This policy describes the technical and procedural controls that prevent one Notion Scan customer from accessing another customer's data. It covers data stored in our application database, Notion integration tokens, and application-layer queries.

Multi-tenancy model

Notion Scan uses logical isolation within a shared database. Each customer has a unique tenant record, and customer data is associated with that tenant either directly or through related scan records. Every request is constrained to data that matches the requesting user's authorized customer scope.

We do not use separate physical databases per customer. Isolation is enforced primarily in the application layer: backend services resolve the authenticated user's authorized client scope and constrain every query to that scope before any data is returned. Postgres Row-Level Security policies on customer-scoped tables provide an independent defense-in-depth backstop at the database layer. Both layers are described below.

Tenant identification

When an authenticated user makes a request:

1. The session resolves to a user account.

2. The user account is linked to one or more customer workspaces through role-based access records.
3. Backend services constrain every query to that user's authorized client scope before returning data, and reject requests for any client the user is not associated with. Row-Level Security policies at the database layer back this enforcement (see below).

Administrative users with cross-client visibility are explicitly granted the `admin` role, which is limited to authorized Notion State personnel.

Database-level controls (Row-Level Security)

As a defense-in-depth backstop to the application-layer checks above, Postgres Row-Level Security (RLS) policies are defined on the customer-scoped tables that hold scan, analysis, and calibration data. For any query that runs under an authenticated end-user database session, these policies enforce three rules:

1. A user can only `SELECT` rows belonging to their associated clients.
2. Where end-user writes are permitted (currently the calibration tables), a user can only `INSERT`, `UPDATE`, or `DELETE` rows belonging to their associated clients. Every other customer-scoped table defines no end-user write policy and is written only by service-role backends.
3. Administrative roles must be explicitly named in the policy; default access is denied.

Because our backend services and scan worker operate with elevated database privileges, they enforce customer scope in application code on every request rather than relying on RLS for those connections. RLS provides an independent guarantee that any path running under an end-user identity is still denied cross-tenant access.

Every scan, analysis, and calibration table that stores customer-scoped data has RLS enabled and tenant-scoping policies defined; client-facing share links follow the separate model described below. New customer-scoped tables are reviewed for isolation before merge.

Client-facing share links

Client-facing deliverables (shareable links to a generated report) use a capability-URL model rather than tenant-scoped RLS. Each deliverable is addressed by an unguessable token and is intended to be opened, without a login, by anyone the customer chooses to share the link with. Row-Level Security is enabled on the underlying table, end-user database access (the anonymous and authenticated roles) is revoked, and the token lookup is performed server-side under a privileged service role, so the table cannot be read or enumerated through an end-user database session. Access control therefore rests on the secrecy of the token rather than on tenant-scoped row policies. A customer can request that a deliverable link be revoked at any time.

Token isolation

Each customer's Notion integration token is encrypted at rest using AES-256-GCM authenticated encryption. The decryption key is held only by the application runtime. Tokens are decrypted in memory only when Notion API access is needed; during a scan, the Railway-hosted worker holds the decrypted token in memory for the scan runtime. Tokens are never logged or persisted in plaintext.

When a customer revokes their integration in Notion, the token becomes useless to us. We delete the encrypted token from our database on confirmed customer deletion request, including offboarding instructions that include deletion.

Cross-tenant access prevention

- **API routes** require authentication and resolve the calling user's client scope before any data access.
- **Privileged backend workflows** are constrained to the customer context they are operating on and are audited via code review.
- **Background jobs** carry the target customer context through the entire job lifecycle.
- **Admin tools** that operate across clients are gated behind the `admin` role and restricted to authorized Notion State personnel.

Audit logging

- Authentication events (login, logout, session refresh) are logged by our auth provider.
- Scan worker activity is logged with the target customer context and timestamp.

Review

Customer isolation controls are reviewed in two cases:

- When a new table or API route is added that touches customer data.
- During quarterly internal review of the database schema and RLS policy set.

Questions

For isolation, multi-tenancy, or architecture questions: security@notionscan.com.