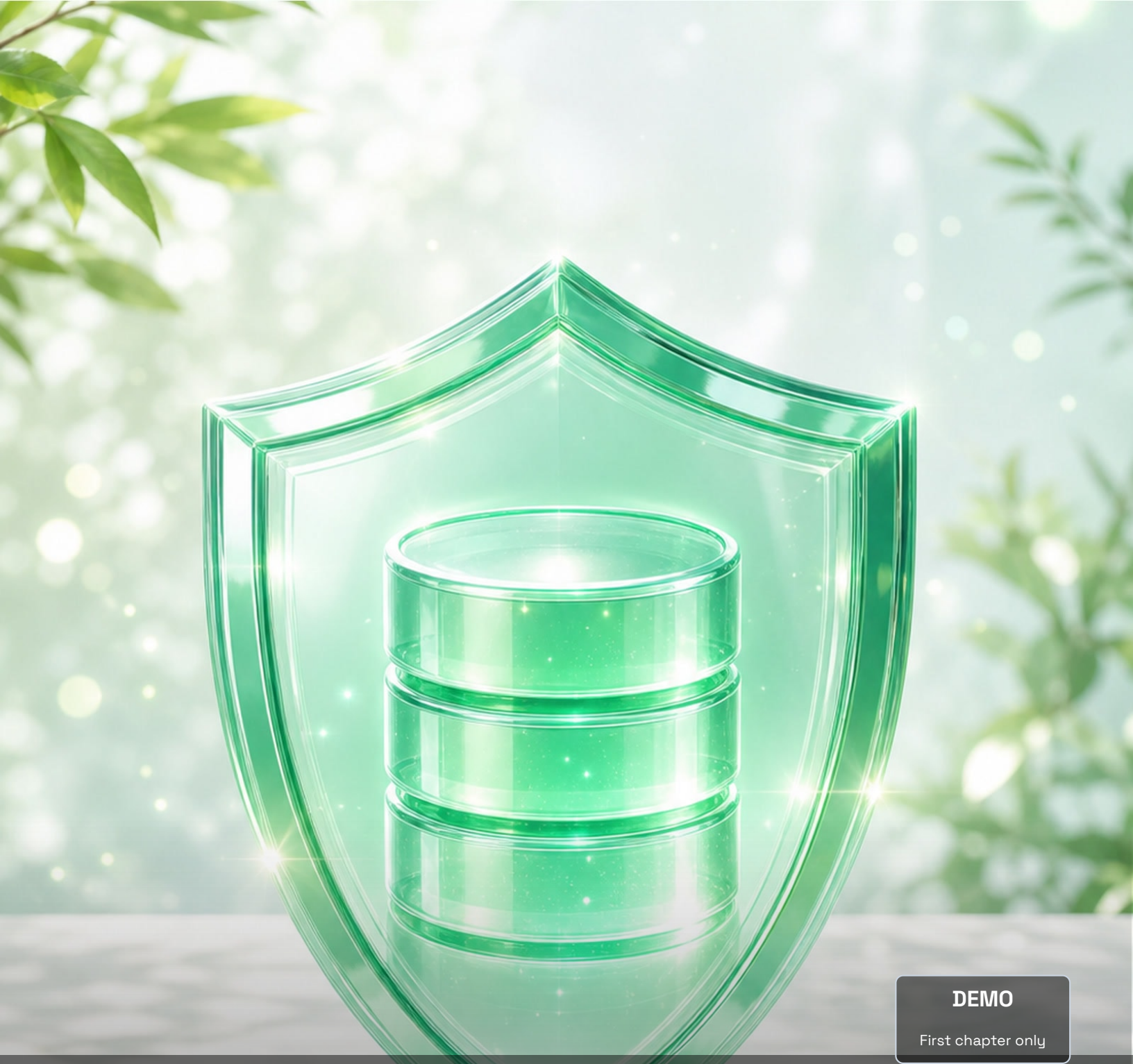

DEMO

First chapter only

Supabase Security Hardening

RLS for Multi-Tenant AI Platforms

pragma.vision

SOFT.HOUSE

Supabase Security Hardening

Trademark Notice

Google, Google Pay, Google Cloud, and Android are trademarks of Google LLC. Stripe is a trademark of Stripe, Inc. Cloudflare and Cloudflare Workers are trademarks of Cloudflare, Inc. Supabase is a trademark of Supabase, Inc. OpenAI and ChatGPT are trademarks of OpenAI, Inc. Claude is a trademark of Anthropic, PBC. W3C is a trademark of the World Wide Web Consortium. Visa is a trademark of Visa International Service Association. OWASP is a trademark of the OWASP Foundation. Midjourney is a trademark of Midjourney, Inc. Canva is a trademark of Canva Pty Ltd. Etsy is a trademark of Etsy, Inc. Amazon is a trademark of Amazon.com, Inc. All other trademarks are the property of their respective owners.

No Affiliation

This book is an independent publication. It is not authorized, sponsored, or endorsed by any of the companies or organizations whose products or services are mentioned herein.

No Professional Advice

The information in this book is provided for educational purposes only. It does not constitute legal, financial, investment, tax, or other professional advice. Readers should consult qualified professionals for guidance specific to their situation.

Code Examples

Code examples in this book are provided for illustration only. They may not be suitable for production use without additional validation, error handling, and security review.

Published by Pragma Vision LLC

First edition, 2026.

Contents

1 Why RLS Is Non-Negotiable 6

1.1 About Pragma.Vision 7

1.2 The Supabase API Exposes Everything by Default 8

1.3 Why Application-Level Filtering Is Not Enough 8

1.4 What This Book Covers 9

1.5 Who This Book Is For 10

2 Supabase RLS Fundamentals 12

2.1 Enabling RLS: The Single Most Important Command 13

2.2 The USING Clause: Controlling Read Access 14

2.2.1 How USING Applies to Different Operations 15

2.3 The WITH CHECK Clause: Controlling Write Access 15

2.3.1 UPDATE Requires Both Clauses 16

2.4 Policy Composition: How Multiple Policies Interact 16

2.5 The auth.jwt() Function: Accessing Token Claims 17

2.5.1 Setting Custom Claims 18

2.6 Performance: Indexing RLS Columns 19

3 Multi-Tenant Isolation Patterns 21

3.1 Three Models of Tenancy 22

3.2 User-Level Isolation 23

3.3 Organization-Level Isolation 24

3.3.1 Optimizing Org Lookups with Security Definer Functions 27

3.4 Role-Based Access Control 28

3.5 Choosing Your Isolation Model 30

4 Protocol-Specific Access Control 32

4.1 The Dual-Protocol Challenge 33

4.2 AP2 Mandate Access Policies 35

4.3 ACP Session Access Policies 36

4.4 Cross-Protocol Policies 37

4.5 Merchant and Agent Access 38

4.6 Protocol-Specific JSONB Metadata 39

5 The Two-Database Strategy 41

5.1 Why One Database Is Not Enough 42

5.2 The Two Projects 43

5.3 Code-Level Isolation Enforcement 44

5.4 Schema Differences Between Projects 45

5.5 Cross-Database Communication 47

5.6 Migration Strategy for Two Projects 48

6 Ephemeral Development Workflow 50

6.1 The Problem with Persistent Dev Databases 51

6.2 The Ephemeral Workflow 52

6.3 Writing Effective Seed Files 53

6.4 Testing RLS in Ephemeral Environments 55

6.5 OAuth Testing Without Ephemeral Auth 56

7 Testing RLS Policies 58

7.1 The Testing Mindset 59

7.2 Positive Tests: Verifying Authorized Access 60

7.3 Negative Tests: Verifying Isolation 61

7.4 Attack Tests: Escalation Attempts 64

7.5 Automated RLS Coverage Checks 66

7.6 Penetration Testing Patterns 67

8 Production Hardening Checklist 69
8.1 Beyond RLS: The Complete Security Picture 70
8.2 Audit Logging 70
8.3 Connection Pooling with PgBouncer 72
8.4 Backup Security 73
8.5 The Production Hardening Checklist 74
8.5.1 RLS Foundation 74
8.5.2 Index Coverage 75
8.5.3 Authentication 75
8.5.4 Audit and Monitoring 75
8.5.5 Connection Security 76
8.5.6 Environment Isolation 76
8.5.7 Disaster Recovery 76
8.6 Closing: Security Is a Foundation, Not a Feature 77

What's Next 79

About Pragma.Vision 81

1

Why RLS Is Non-Negotiable

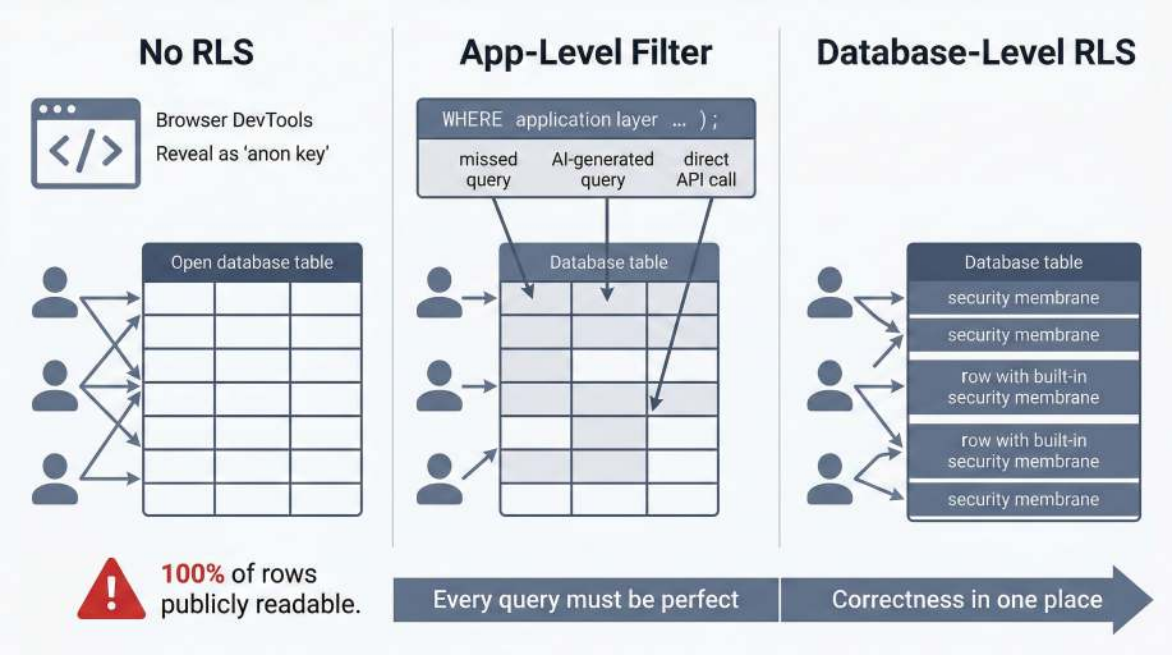

Figure 1. Three access models compared — no RLS leaves 100% of rows publicly readable through the exposed anon key, app-level filters demand every query be perfect against missed or AI-generated queries, and database-level RLS enforces correctness once with a per-row security membrane

1.1 About Pragma.Vision

Pragma.Vision is an AI-native commerce ecosystem—a growing family of interconnected platforms sharing infrastructure, identity, and payment systems. When an AI agent on wish.now processes a purchase, when a developer on phantoid.com deploys a verified agent, when trust.authority issues a credential, every operation touches the same underlying database layer.

That database stores financial mandates, user identities, agent credentials, transaction histories, and protocol-specific metadata. A single misconfigured access policy does not just expose one user's data—it exposes every user on every platform simultaneously.

This book exists because Row-Level Security is the single most important security feature in the Pragma.Vision database architecture, and it is the feature most commonly misconfigured by development teams building multi-tenant AI platforms.

1.2 The Supabase API Exposes Everything by Default

Supabase generates a PostgREST API automatically for every table. This is its greatest strength and its greatest danger. The moment you create a table, it is queryable from any client with your project's public anon key. Without Row-Level Security enabled, every row in that table is accessible to every user.

100%

of Supabase tables without RLS are publicly readable through the auto-generated API

This is not a bug. It is the design. Supabase trusts that you will define access policies at the database level rather than the application level. If you skip that step, the API exposes your data with the same enthusiasm it uses to serve it.

Warning

The Supabase SQL Editor and server-side clients bypass RLS entirely. Testing queries in the dashboard does not validate your security policies. You must test from the client SDK—the same path your users and attackers will take.

1.3 Why Application-Level Filtering Is Not Enough

Many developers add `WHERE user_id = current_user` to their application queries and consider the problem solved. This approach fails for three reasons:

1. **Every query must be perfect.** A single missing filter in any query, in any endpoint, in any edge function, exposes data. Application-level security requires perfection across every code path. Database-level security requires correctness in one place.
2. **AI-generated queries cannot be trusted.** If your platform uses AI agents that construct or modify database queries, application-level filtering depends on the AI never generating a query that omits the user filter. This is an unreliable assumption.
3. **Direct API access bypasses your application entirely.** Supabase's PostgREST API is accessible to anyone with your project URL and anon key—both of which are embedded in your client-side JavaScript and visible to anyone who opens browser DevTools.

Key Insight

Row-Level Security is not an additional layer on top of application logic. It is the *foundation* beneath it. Application filtering is an optimization for reducing unnecessary data transfer. RLS is the enforcement mechanism that makes unauthorized access structurally impossible at the database level.

1.4 What This Book Covers

This book is a practitioner's guide to implementing comprehensive RLS for multi-tenant AI platforms built on Supabase. It is grounded in real implementation experience—the Pragma.Vision ecosystem enforces RLS on every table across two separate Supabase projects, supporting multiple protocols, tenant models, and compliance requirements.

You will learn:

1. How Supabase RLS fundamentals work—`USING` clauses, `WITH CHECK`, `auth.uid()`, and policy composition (Chapter 2).

2. Multi-tenant isolation patterns for user-level, organization-level, and role-based access (Chapter 3).
3. Protocol-specific access control for dual-protocol architectures like AP2 and ACP (Chapter 4).
4. The two-database strategy that separates consumer platforms from compliance-critical infrastructure (Chapter 5).
5. Ephemeral development workflows that prevent PII from leaking into development environments (Chapter 6).
6. Testing strategies for RLS policies including IDOR testing and penetration test patterns (Chapter 7).
7. Production hardening patterns including audit logging, connection pooling, and backup security (Chapter 8).

Pro Tip

Every SQL example in this book uses Supabase PostgreSQL 17 syntax and can be applied directly as migration files. The patterns are portable to any PostgreSQL database with RLS support, but the `auth.uid()` and `auth.jwt()` functions are Supabase-specific helpers.

1.5 Who This Book Is For

This book is for backend developers, platform architects, and security engineers building multi-tenant applications on Supabase—particularly those whose platforms involve AI agents, multiple user types, or financial transactions. If you have ever asked "how do I make sure User A cannot see User B's data," this book gives you the complete answer.

DEMO

This is a free preview of the full edition.

Get the complete book at:

https://shop.pragma.vision