

warning "Backend update (October 2026)" GitHub Models inference was retired on July 30, 2026. Any references below are retained only for historical context while this lab is migrated. Use Lab 013 ? Free LLM backends to configure Foundry Local or Ollama before continuing.

Lab 032: Row Level Security for Agents

Level: L300 Path: Foundry + Security Time: ~45 min ? Cost: Free (local Docker) or Azure PostgreSQL (~\$15/mo)

What You'll Learn

Why agents need data isolation (multi-tenant risk) Implement PostgreSQL Row Level Security (RLS) ? the database enforces access, not your app Create database roles for agent identities Test that an agent literally cannot see another user's data ? even with injection attacks Apply RLS to the orders.csv sample dataset

Introduction

When your agent queries the database, it might be serving Customer A ? but the SQL query it generates could accidentally return Customer B's orders. Prompt injection attacks exploit exactly this.

Row Level Security moves access control into the database itself. Even if the agent generates `SELECT * FROM orders`, PostgreSQL automatically adds `WHERE customer_id = current_user_id` based on the session role. The agent cannot escape it.

Prerequisites

Docker Desktop (free) OR Azure PostgreSQL from Lab 031 Python 3.11+, pip install `psycopg2-binary` `openai` `GITHUB_TOKEN` set

? Supporting Files

note "Download these files before starting the lab" Save all files to a `lab-032/` folder in your working directory.

File | Description | Download | `002_rls.sql` | Row-level security migration | ? Download |

Lab Exercise

Step 1: Start PostgreSQL with Docker

[diagram or code sample]

Step 2: Load sample orders and set up RLS

[diagram or code sample]

[diagram or code sample]

Step 3: Enable Row Level Security

[diagram or code sample]

[diagram or code sample]

Step 4: Test isolation ? agents can't cross boundaries

[diagram or code sample]

Expected output: [diagram or code sample]

Step 5: Build an RLS-aware agent

[diagram or code sample]`sql").strip("[diagram or code sample]`

Why This Matters

Without RLS | With RLS | App must filter: `WHERE customer_id = ?` | Database filters automatically | Prompt injection can bypass app filters | Database-level policy ? cannot be bypassed | Bug in agent = data leak | Bug in agent = still only sees own data | Must trust every query the agent generates | Trust is in the database role |

Cleanup

[diagram or code sample]

Next Steps

Agent Observability: ? Lab 033 ? App Insights Tracing RLS on Azure PostgreSQL: ? Lab 031 ?
pgvector on Azure

Supporting kit

The site build generates stable PDF and ZIP downloads. The ZIP contains the lab guide and every supporting file declared for this lab.

[Download generated PDF](#) [Download generated ZIP](#)