

warning "Atualização de backend (outubro de 2026)" A inferência do GitHub Models foi descontinuada em 30 de julho de 2026. As referências abaixo são mantidas apenas como contexto histórico enquanto este lab é migrado. Use o Lab 013 ? Backends de LLM gratuitos para configurar o Foundry Local ou o Ollama antes de continuar.

Lab 032: Row Level Security para Agentes

Nível: L300 Trilha: Foundry + Security Tempo: ~45 min ? Custo: Gratuito (Docker local) ou Azure PostgreSQL (~\$15/mês)

O que Você Vai Aprender

Por que agentes precisam de isolamento de dados (risco multi-tenant) Implementar PostgreSQL Row Level Security (RLS) ? o banco de dados impõe o acesso, não sua aplicação Criar roles de banco de dados para identidades de agentes Testar que um agente literalmente não consegue ver os dados de outro usuário ? mesmo com ataques de injeção Aplicar RLS ao conjunto de dados de exemplo orders.csv

Introdução

Quando seu agente consulta o banco de dados, ele pode estar atendendo o Cliente A ? mas a consulta SQL que ele gera pode acidentalmente retornar os pedidos do Cliente B. Ataques de injeção de prompt exploram exatamente isso.

Row Level Security move o controle de acesso para dentro do próprio banco de dados. Mesmo que o agente gere `SELECT * FROM orders`, o PostgreSQL adiciona automaticamente `WHERE customer_id = current_user_id` com base no role da sessão. O agente não pode escapar disso.

Pré-requisitos

Docker Desktop (gratuito) OU Azure PostgreSQL do Lab 031 Python 3.11+, pip install psycopg2-binary openai GITHUB_TOKEN configurado

? Arquivos de Apoio

note "Baixe estes arquivos antes de iniciar o lab" Salve todos os arquivos em uma pasta lab-032/ no seu diretório de trabalho.

Arquivo | Descrição | Download | 002_rls.sql | Migração de segurança em nível de linha | ? Download |

Exercício do Lab

Passo 1: Iniciar o PostgreSQL com Docker

[diagram or code sample]

Passo 2: Carregar pedidos de exemplo e configurar RLS

[diagram or code sample]

[diagram or code sample]

Passo 3: Habilitar Row Level Security

[diagram or code sample]

[diagram or code sample]

Passo 4: Testar isolamento ? agentes não conseguem cruzar limites

[diagram or code sample]

Saída esperada: [diagram or code sample]

Passo 5: Construir um agente com suporte a RLS

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

Por que Isso Importa

Sem RLS | Com RLS | A aplicação deve filtrar: `WHERE customer_id = ?` | O banco de dados filtra automaticamente | Injeção de prompt pode contornar filtros da aplicação | Política no nível do banco ? não pode ser contornada | Bug no agente = vazamento de dados | Bug no agente = ainda vê apenas seus próprios dados | Precisa confiar em toda consulta que o agente gera | A confiança está no role do banco de dados |

Limpeza

[diagram or code sample]

Próximos Passos

Observabilidade de Agentes: ? Lab 033 ? App Insights Tracing RLS no Azure PostgreSQL: ? Lab 031 ? pgvector on Azure

Kit de apoio

O build do site gera downloads estáveis em PDF e ZIP. O ZIP contém o guia do lab e todos os arquivos de apoio declarados para este lab.

Baixar PDF gerado Baixar ZIP gerado