Diretrizes de Arquitetura: Conciliador Financeiro Multi-Tenant (10k Clientes)

1. Estratégia de Isolamento: Shared Schema com RLS

Decisão: Utilizar **uma única tabela por entidade** (Shared Schema) com coluna discriminadora `tenant_id` e isolamento forçado via **Row-Level Security (RLS)** do PostgreSQL. **NÃO USAR:** "Schema-per-tenant" (um schema por cliente).

- **Justificativa Técnica:**
  - **Escalabilidade:** O PostgreSQL sofre degradação severa no catálogo do sistema (tabelas `pg_class`, `pg_attribute`) e no *autovacuum* ao gerenciar centenas de milhares de tabelas (50 tabelas x 10.000 clientes = 500.000 tabelas).
  - **DevOps/Prisma:** Rodar migrações (deploy) em 10.000 schemas levaria horas e bloquearia o CI/CD. O Prisma ORM gerencia mal conexões dinâmicas para múltiplos schemas.
- **Implementação:**
  - Adicionar coluna `tenant_id` (UUID) indexada em todas as tabelas principais.
  - Ativar RLS: `ALTER TABLE x ENABLE ROW LEVEL SECURITY;`
  - Política RLS: `USING (tenant_id = current_setting('app.current_tenant_id')::uuid)`.
  - **Prisma:** Usar **Prisma Client Extensions** para envolver queries em transações que executam `SET LOCAL app.current_tenant_id = '...'` antes da query.
  - **PgBouncer:** Configurar em **Transaction Mode**.

2. Segurança de Credenciais: Envelope Encryption (Camada de Aplicação)

Decisão: Criptografar segredos (`client_secret`, tokens) na aplicação (Node.js) usando **Envelope Encryption** com um KMS externo (AWS KMS / Vault). **NÃO USAR:** Extensão `pgcrypto` como solução principal.

- **Justificativa Técnica:**
  - **Performance:** Criptografia AES consome muita CPU. Fazer isso no banco cria um gargalo vertical difícil de escalar.

    - **Segurança (PCI-DSS):** O banco de dados nunca deve ver a chave de descriptografia em texto plano (o que ocorreria com `pgcrypto` na query SQL). Se houver um dump do banco, os dados estão seguros pois a chave mestre está no KMS.
- **Implementação:**
    - **KMS:** Gera uma Data Key (DEK) para cada segredo (ou tenant).
    - **Node.js:** Usa a DEK para cifrar o dado com **AES-256-GCM**.
    - **Banco:** Armazena o *Ciphertext* (dado cifrado) + *Encrypted Data Key* (chave cifrada pelo KMS). Nunca armazena a chave em texto plano.

3. Auditoria (Audit Log): Particionamento + JSONB Diff

Decisão: Tabela única de eventos particionada nativamente por tempo (Partitioning) armazenando apenas o delta das alterações em JSONB.

- **Justificativa Técnica:**
    - **Volume:** Logs crescem 100x mais rápido que dados. Uma tabela não particionada ficará lenta e difícil de manter (vacuum/índices).
    - **Retenção:** Permite deletar logs antigos instantaneamente com `DROP TABLE partition_2023_01` (custo zero de IO) em vez de `DELETE WHERE date <...` (que fragmenta o banco).
- **Implementação:**
    - Usar extensão **`pg_partman`** para criar partições mensais automaticamente.
    - Estrutura: `event_id`, `tenant_id`, `actor_id`, `resource`, `action`, `changes (JSONB)`.
    - JSONB Diff: Armazenar apenas o que mudou: `{ "taxa": { "old": 2.0, "new": 2.5 } }`. Não copiar a linha inteira.

4. Tipos Numéricos: DECIMAL de Alta Precisão

Decisão: Usar `DECIMAL` (ou `NUMERIC`) com precisão explícita. **NÃO USAR:** `FLOAT` (impreciso) ou `INTEGER` (centavos - problemático para rateio/câmbio).

- **Diretrizes:**
    - **Valores Monetários:** **`DECIMAL(19, 4)`**. Permite trilhões de reais e 4 casas decimais para suportar frações de centavos em cálculos de impostos/juros diários sem erro de arredondamento prematuro.
    - **Taxas de Câmbio/Juros:** **`DECIMAL(24, 12)`**. Taxas financeiras exigem precisão extrema para evitar divergências ao multiplicar grandes volumes.
    - **Moeda:** Coluna separada `currency_code CHAR(3)` (ISO 4217).