Guia de Banco de Dados – Bot Cripto Telegram

# 1. Introdução

Este guia descreve a modelagem e o uso do banco de dados para o projeto Bot Cripto Telegram. A arquitetura de dados combina o uso de \*\*Supabase/Postgres\*\* para persistência e \*\*Redis\*\* para cache de dados quentes (preços, símbolos, notícias).

# 2. Estrutura de Persistência (Supabase/Postgres)

As tabelas principais foram definidas no schema Prisma. Elas representam entidades persistentes e consultas históricas.

- \*\*User\*\*: Representa cada usuário do bot (no MVP apenas 1). Contém chatId do Telegram, idioma e status ativo.  
- \*\*Alert\*\*: Alertas de preço definidos pelo usuário, incluindo direção (acima/abaixo), preço alvo, ativo ou não.  
- \*\*Trade\*\*: Registro de compras e vendas reais, com quantidade, preço unitário, taxa e timestamp.  
- \*\*BotLog\*\*: Log geral das operações e eventos do bot (observabilidade).  
- \*\*ApiUsage\*\*: Controle de uso de APIs externas, com contagem por período, endpoint e serviço.  
- \*\*ErrorState\*\*: Captura e registro centralizado de erros para análise posterior.

## 2.1 Esquema Lógico

- Cada `User` pode ter vários `Alert`, `Trade` e `BotLog`.  
- `Alert` e `Trade` possuem índices para otimizar consultas por usuário, símbolo e estado.  
- O campo `Decimal` é usado para quantidades e valores monetários, garantindo precisão para cripto.  
- Logs e erros são centralizados para auditoria.

## 2.2 Funções e Views Auxiliares

- \*\*fn\_positions\_cma(user\_uuid)\*\*: Calcula posição líquida e custo médio por símbolo para um usuário.  
- \*\*v\_positions\_current\*\*: View que exibe saldo atual por moeda com custo médio.  
- \*\*fn\_pnl\_cma(user\_uuid, prices jsonb)\*\*: Calcula PnL realizado e não realizado dado um snapshot de preços atuais.

# 3. Estrutura de Cache (Redis)

Redis é usado para armazenar dados voláteis e de alta frequência, reduzindo chamadas às APIs externas e acelerando respostas.

## 3.1 Chaves Redis

- `px:spot:<SYMBOL><QUOTE>` → preço spot em BRL ou USDT (TTL 5–10s)  
- `px:conv:USDTBRL` → taxa de conversão USDT para BRL (TTL 30–60s)  
- `meta:symbols` → lista de símbolos suportados pela Binance (TTL 6–12h)  
- `news:top` → cache de manchetes de notícias (TTL 120–300s)

## 3.2 Padrão Cache-Aside

1. Workflow consulta Redis primeiro.  
2. Se cache válido → retorna direto.  
3. Se cache expirado ou ausente → chama API externa.  
4. Resultado é gravado no Redis com TTL adequado.

# 4. Estratégia de Integração N8N ↔ Banco

- Para Supabase:  
 - Usar \*\*dupla integração\*\*: nó Supabase API (CRUD simples) + nó Postgres (consultas SQL complexas).  
- Para Redis:  
 - Usar \*\*Upstash Redis (HTTP API)\*\* ou um microserviço bridge HTTP.  
 - No N8N, chamadas via \*\*HTTP Request Node\*\* para GET/SETEX.

# 5. Segurança e Boas Práticas

- Todas as credenciais armazenadas no \*\*N8N Vault\*\*.  
- Limitar TTLs no Redis para evitar dados defasados.  
- Sanitizar inputs antes de gravar no Postgres.  
- Usar índices em colunas de busca frequente (`userId`, `symbol`, `active`, `ts`).  
- Backups automáticos do Supabase já cobrem Disaster Recovery.

# 6. Futuras Extensões

- Suporte a múltiplos usuários (já previsto no schema).  
- Implementação FIFO em vez de custo médio móvel para PnL.  
- Dashboard web consumindo as views do Supabase.  
- Cache distribuído com Redis Cluster (se necessário).