Idea y propósito.

Idea y propósito. Proponemos un servidor local llamado SQLScout MCP que inspecciona consultas SQL y planes de ejecución para detectar cuellos de botella comunes (full scans innecesarios, falta de índices, funciones no sargables, JOINs desbalanceados) y devuelve recomendaciones accionables (índices sugeridos, reescrituras de predicados, hints opcionales). El objetivo es apoyar al desarrollador a mejorar el rendimiento antes de desplegar cambios, manteniendo la lógica de análisis encapsulada como herramienta MCP invocable por el anfitrión (chatbot).

Uso por el cliente. Desde mi chatbot en consola, el LLM podrá llamar métodos como sql.load(schema), sql.explain(query) y sql.optimize(query, dialect) para obtener (1) el plan de ejecución y métricas básicas, (2) un diagnóstico con reglas estáticas (por ejemplo, “LIKE '%texto' rompe índice”), y (3) sugerencias priorizadas (p. ej., “crear índice compuesto (user_id, created_at) reduce costo estimado de 185k a 12k”). El host mantiene contexto y registra cada solicitud/respuesta como parte del log de interacciones MCP exigido por el proyecto; así puedo demostrar el flujo completo y la trazabilidad desde la línea de comandos.

API/Librerías y experiencia. Implementaré el servidor en Python usando el SDK oficial de MCP (JSON-RPC) y conectores nativos según el SGBD (por ejemplo, psycopg para PostgreSQL). Para análisis usaré EXPLAIN/EXPLAIN ANALYZE donde aplique, más un conjunto de reglas estáticas (AST con sqlglot) y heurísticas simples (selectividad aproximada, cardinalidad esperada). Para generar reportes cortos y claros, formatearé salidas tabulares y, si es útil, diagramas Mermaid del plan. Tengo experiencia con SQL y Python; es mi primera vez creando un servidor MCP, pero seguiré la especificación y ejemplos oficiales. El servidor será público en un repo independiente, con README y ejemplos de uso, tal como solicita la guía.

Ejemplo:

Entrada:
SELECT * FROM orders WHERE LOWER(email) LIKE '%gmail.com' ORDER BY created_at DESC;

Diagnóstico (estático):

- SELECT * → recomienda columnas específicas.
- LOWER(email) + LIKE '%...' → no sargable; sugiere columna email_normalized y LIKE 'gmail.com%' si aplica.
- ORDER BY created_at sin índice → sort costoso.

Sugerencias:

- ALTER TABLE orders ADD COLUMN email_normalized TEXT GENERATED ALWAYS AS (lower(email)) STORED; *(o mantenerla en app)*
- CREATE INDEX idx_orders_emailnorm_created ON orders(email_normalized, created_at DESC);
- Reescritura de consulta seleccionando columnas necesarias.