Bringing GenAI to every MySQL instance

Don't migrate your data – add a transparent AI proxy layer

FOSDEM 2026 • (date) • (presenter)

Agenda (and what is ready today)

- Why MySQL is still "AI-blind" in most organizations
- The ProxySQL-as-an-AI-layer architecture
- What exists in v4.0 today (MCP, Catalog, Search tools)
- What is still WIP (replication indexing, embeddings, DISTANCE())
- Demo flow and next steps

Important caveat: this is a prototype showcase.

About the author

Rene' Cannao'

Author of ProxySQL • CEO of ProxySQL

- Author of ProxySQL since 2013
- Over 2 decades as system, network, and database administrator
- Last 15 years focused mainly on MySQL performance

ProxySQL (the company)

We build a high-performance, high-availability proxy for MySQL and compatible systems.

- Mission: maximize performance and scalability at the database layer
- Offerings: enterprise support, consulting, and training
- Deep focus on reliability engineering and real-world production ops

We are hiring. Join us to push databases and AI integration forward.

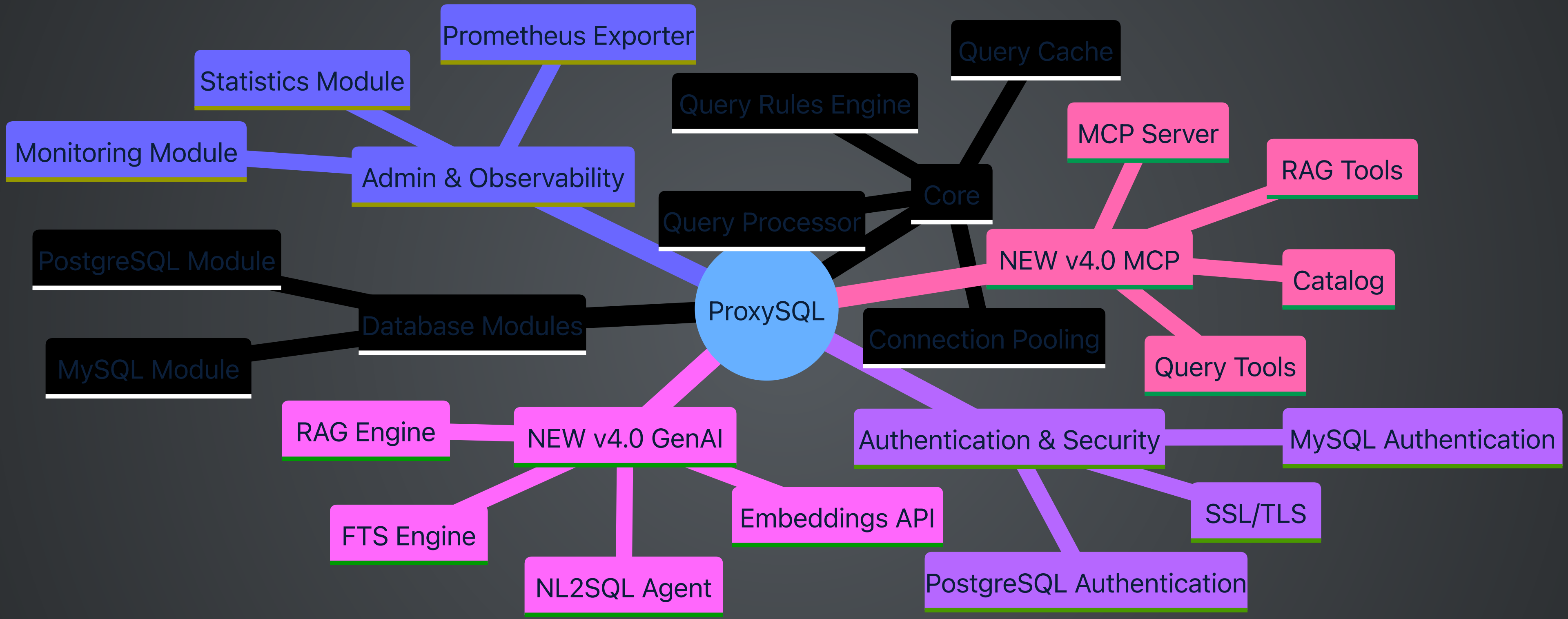
The GenAI adoption bottleneck: existing DB infrastructure

- Most MySQL estates are a mix of Community, managed cloud services, and legacy versions
- Teams want "ask the data" and RAG — but the database layer is not designed for it
- AI integrations usually land in the app layer (and sprawl quickly)

One-sentence summary

ProxySQL turns any MySQL into a governed AI endpoint without moving data.

ProxySQL: Modular Architecture



v4.0 adds two major modules: GenAI (for AI capabilities) and MCP (for agent integration).

Why "just upgrade/migrate" is rarely the answer

- Operational risk: upgrades, downtime windows
- Schema changes: `ALTER TABLE`, backfills, new indexes
- New infra: vector DBs, embedding services, extra monitoring
- Lock-in: AI stacks couple apps tightly to a vendor/tooling choice

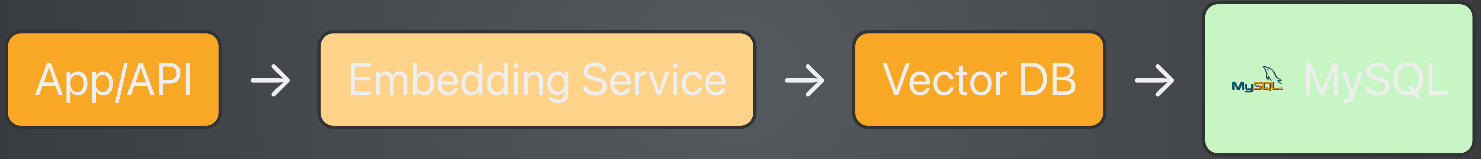
Typical GenAI data architecture (and why it gets messy)

App/API → Embedding service → Vector DB

Pain points:

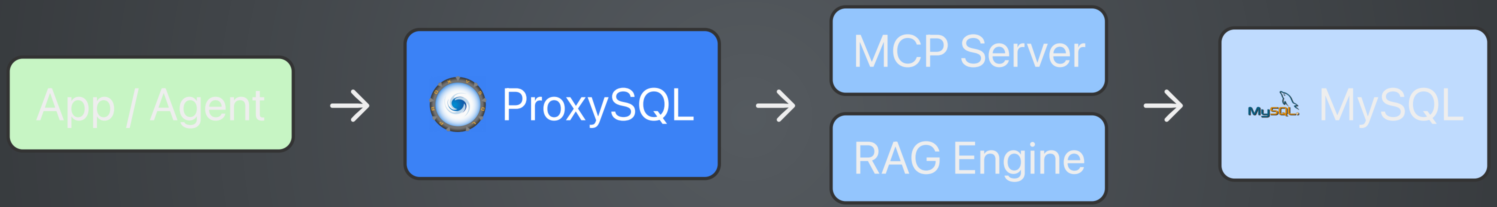
- Dual-write / synchronization headaches
- Hard-to-govern access paths (agent bypass, prompt injection risk)
- Latency + cost per query
- "Vector schema drift" becomes another product to maintain

Traditional GenAI Architecture



App also talks directly to MySQL for OLTP; vector DB syncs back.

ProxySQL AI Layer Architecture



Key difference: ProxySQL integrates AI features at the proxy layer — no migration needed.

Thesis: move intelligence to the proxy layer

"ProxySQL as a transparent AI-acceleration layer"

- No changes to the application connection string
- No schema changes on source tables
- Centralized governance: rules, auth, audit, throttling
- Add AI close to data while preserving MySQL stability

The proxy layer abstracts the complexity of vector stores and LLM APIs.

App-Layer vs. ProxySQL AI Layer

	App-Layer Integration	ProxySQL AI Layer
Complexity	High (logic in every app)	Low (centralized offload)
Schema Changes	Often required	None (transparent)
Governance	Fragmented per service	Unified (Query Rules)
Security	Hard to audit globally	Centralized audit/auth
Latency	Multiple network hops	Single proxy layer
Maintenance	Per-service code changes	Central configuration

What teams get on day 1

- A single, governable entry point for apps and agents
- Immediate schema discovery and safe, read-only querying
- Hybrid search over catalog + documents without app changes
- Auditability via MCP query rules and stats tables

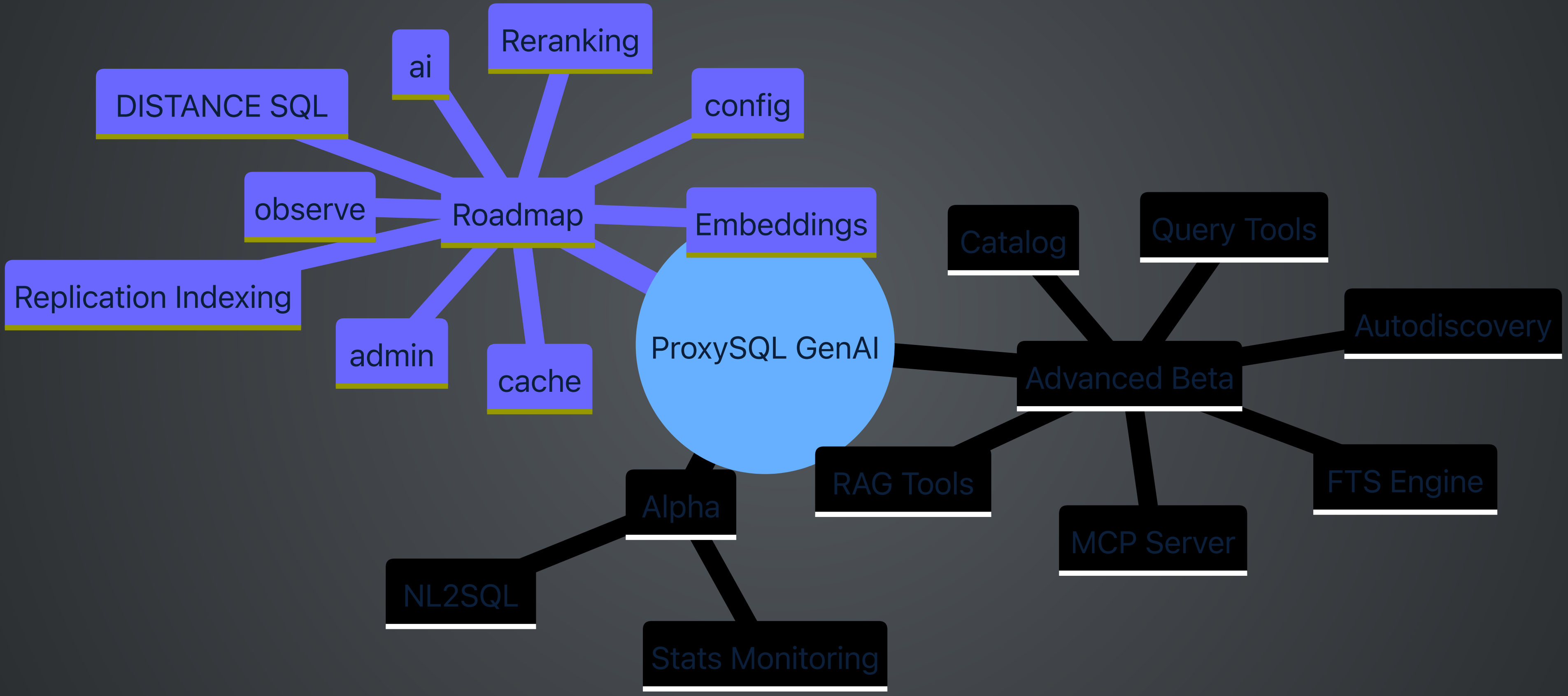
Why ProxySQL is a natural place for AI guardrails

- Protocol-aware: understands MySQL sessions/traffic
- Policy engine: query rules, rewriting, routing, caching
- Operationally mature: deployed in front of critical DBs
- Single choke-point: observability + enforcement for apps and agents

Design principles

- **Transparent**: no app rewrites, no schema migrations
- **Deterministic**: tools, rules, and audit trails
- **Operational**: same uptime, same SLOs, same governance

ProxySQL v4.0: Generative AI building blocks



Maturity Levels

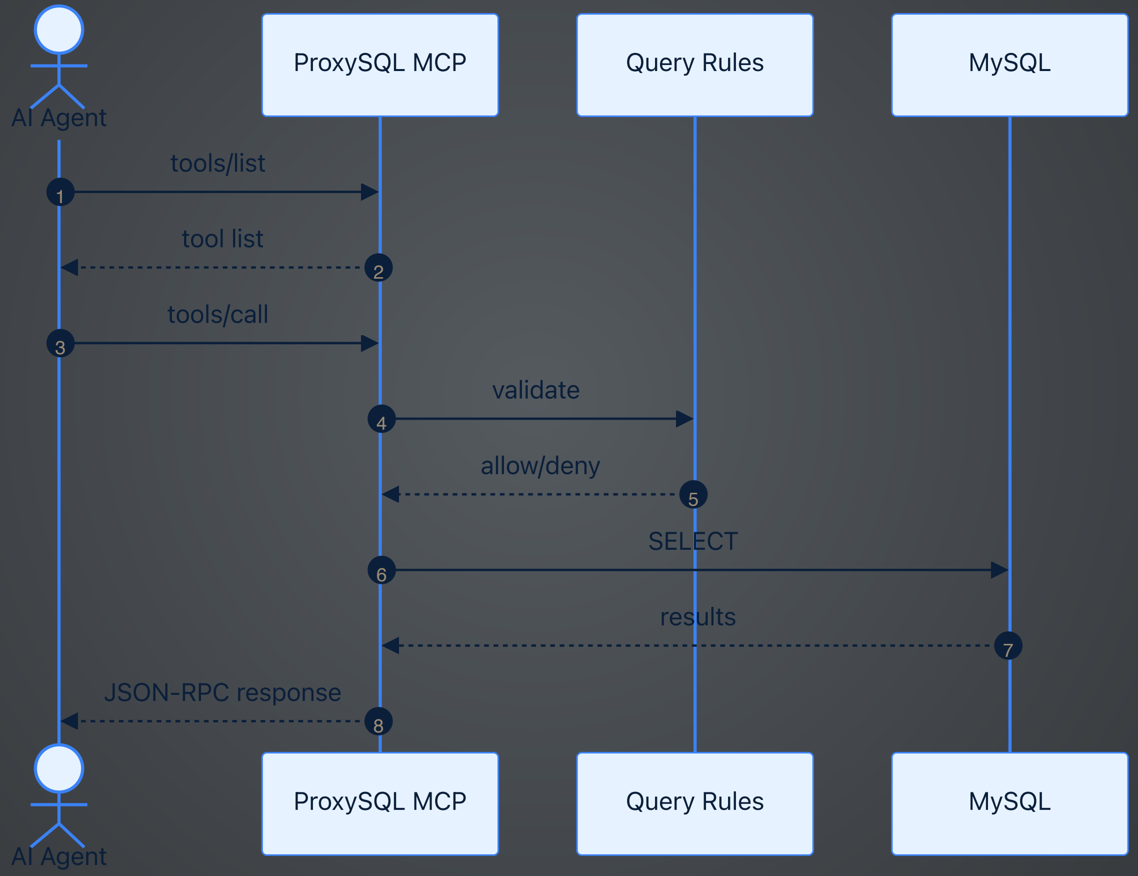
- ✅ **Advanced Beta**: MCP Server, Query Tools, RAG Tools, FTS, Autodiscovery, Catalog
- 🚧 **Beta**: NL2SQL, Stats/Monitoring
- 🔮 **Roadmap**: Embeddings, Replication Indexing, Reranking, DISTANCE SQL, /mcp/config, /mcp/admin, /mcp/cache, /mcp/observe, /mcp/ai
- ...and more

MCP in 60 seconds

- **MCP** (Model Context Protocol) standardizes how agents discover and call **tools**
- Tools are deterministic functions (e.g., `list_tables`, `run_sql_readonly`, `rag.search_hybrid`)
- JSON-RPC over HTTP/HTTPS
- Outcome: less bespoke glue code; clearer governance and auditing

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "tools/list",
  "params": {}
}
```

MCP Tool Interaction



ProxySQL as an MCP Server

- ProxySQL v4.0 introduces native MCP server support
- Agents can discover schemas, execute queries, and access metrics via a standardized interface
- Requests are subject to MCP Query Rules (safety + governance)
- Connectivity: HTTP/HTTPS, JSON-RPC 2.0, default port `6071`, bearer-token auth

Agents Integration

ProxySQL's MCP server works with any MCP-compatible agent:

- **Popular Agents**: Claude Code, GitHub Copilot, Codex, Gemini Code Assist, Cursor, Warp, others
- **Integration methods**: HTTP/HTTPS (native JSON-RPC), STDIO bridge, direct library integration

30+ Tools Available via MCP

- **Query**: `list_schemas`, `list_tables`, `run_sql_readonly`, `explain_sql`, `suggest_joins`
- **RAG**: `search_fts`, `search_vector`, `search_hybrid`, `get_chunks`, `get_docs`
- **Catalog**: `discovery.run_static`, `llm.search`, `llm.summary_upsert`, `llm.note_add`
- **Admin**: `rag.admin.stats`, session tracking, tool usage monitoring

Enabling the MCP server

```
SET mcp-enabled='true';
SET mcp-use_ssl='true';
SET mcp-port='6071';
LOAD MCP VARIABLES TO RUNTIME;
```

Key variables:

- `mcp-enabled`, `mcp-port`, `mcp-use_ssl`, `mcp-timeout_ms`
- Per-endpoint auth tokens (optional)

Endpoint model: Query Tools

Tools available for `/mcp/query`:

- `list_schemas`, `list_tables`, `list_columns`, `list_indexes`
- `get_constraints`, `get_foreign_keys`
- `run_sql_readonly`, `explain_sql`
- `suggest_joins` — AI-driven join suggestions
- `llm.search` — Search catalog with semantic understanding

Endpoint model: RAG Tools

Tools available for `/mcp/rag`:

- `rag.search_fts`, `rag.search_vector`, `rag.search_hybrid`
- `rag.get_chunks`, `rag.get_docs`, `rag.fetch_from_source`
- `rag.admin.stats`

Endpoint model: Under Development

Coming soon in future releases:

- `/mcp/config` — AI-driven configuration management
- `/mcp/admin` — Administrative tasks (backup, restore, maintenance)
- `/mcp/cache` — Cache inspection via natural language
- `/mcp/observe` — Real-time metrics and observability
- `/mcp/ai` — Direct LLM bridge for custom agent workflows

Authentication & transport

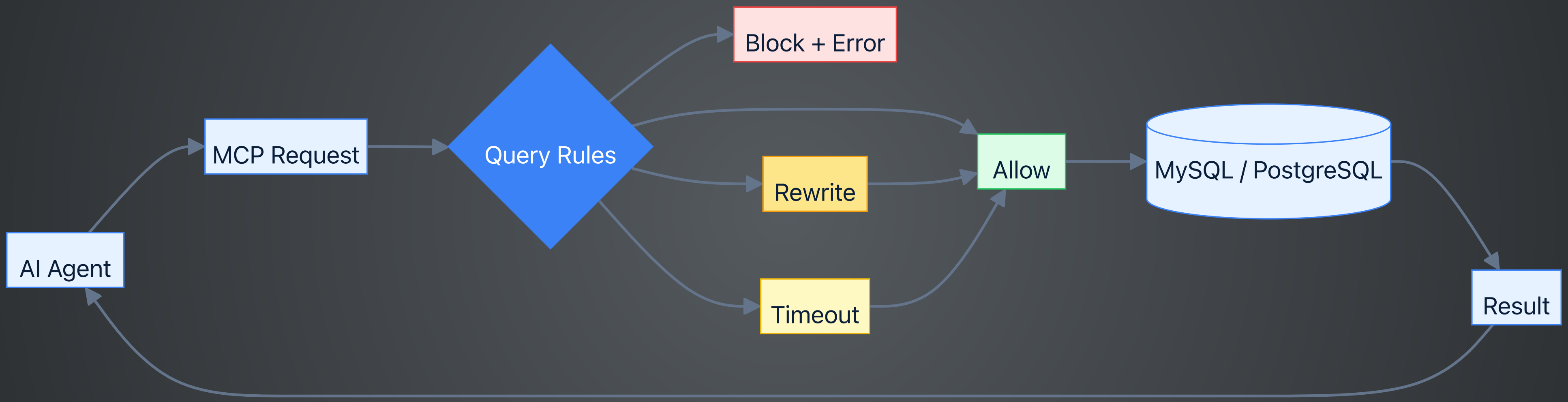
- HTTP/HTTPS transport; HTTPS can be enforced via `mcp-use_ssl`
- Bearer token authentication is configurable per endpoint
- Tokens via Authorization header or query parameter

```
SET mcp-query_endpoint_auth='your_secure_token_here';
LOAD MCP VARIABLES TO RUNTIME;
```

Safety and determinism: MCP Query Rules

- MCP requests are subject to MCP Query Rules (analogous to `mysql_query_rules`)
- Allow-list tools and constrain query patterns (e.g., read-only only)
- Reduce risk of accidental writes, exfiltration, or "agent surprises"

How MCP Query Rules Work

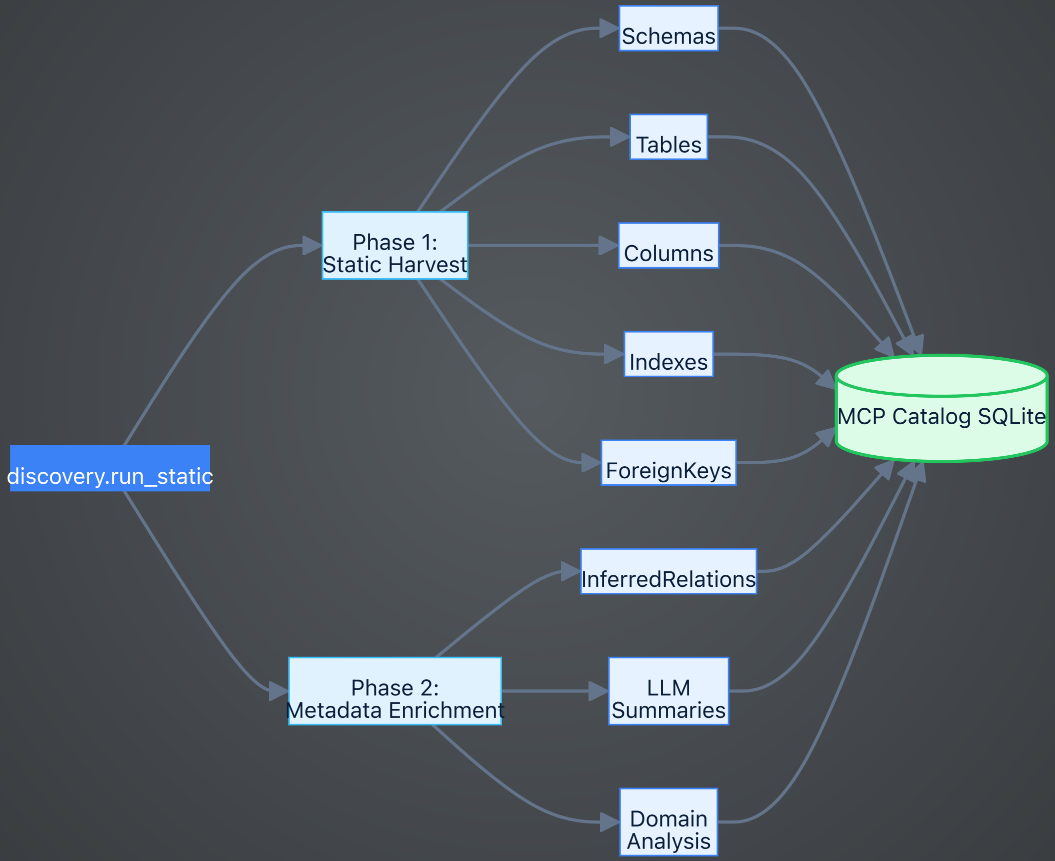


Query Rules provide fine-grained control: block dangerous operations, rewrite for safety, or enforce timeouts.

Autodiscovery: build context

- Deterministic "Two-Phase Discovery" for accuracy
- Triggered by `discovery.run_static`
- Iterates backend schema and populates the MCP Catalog

Autodiscovery: build context



MCP Catalog: Core Metadata

- Local SQLite database: `mcp_catalog.db`
- Stores high-fidelity context for LLMs
- `schemas`, `tables`, `columns`
- `indexes`, `foreign_keys`
- `constraints`, `relationships`, `dependencies`

MCP Catalog: AI & Sessions

- **Contextual metadata**: `llm_summaries`, `llm_notes`, `llm_embeddings`
- **Intelligence & tracking**: `agent_sessions`, `question_templates`

Query tools (`/mcp/query`)

- Exploration, schema discovery, and safe data retrieval
- **Discovery**: `list_schemas`, `list_tables`, `list_columns`, `list_indexes`
- **Analysis**: `get_constraints`, `get_foreign_keys`, `suggest_joins`
- **Execution**: `run_sql_readonly`, `explain_sql`
- **LLM-assisted**: `llm.search` for semantic catalog search

RAG tools: Search

Hybrid Search Endpoints

- `rag.search_fts` — Keyword search with BM25 ranking
- `rag.search_vector` — Semantic similarity search
- `rag.search_hybrid` — Combined keyword + semantic with fusion

RAG tools: Retrieve & Monitor

Data Retrieval

- `rag.get_chunks` — Fetch text chunks with citations
- `rag.get_docs` — Get document metadata
- `rag.fetch_from_source` — Retrieve from original source

Performance

- `rag.admin.stats` — RAG engine statistics and performance

Full-Text Search inside ProxySQL

- FTS engine integrated into the MCP server
- Keyword discovery across indexed documents and metadata
- Based on SQLite FTS5; foundational for RAG workflows

Demo flow: Step by step

1. Enable MCP server and configure endpoint auth
2. `tools/list` to see exposed capabilities
3. `discovery.run_static` to build catalog
4. `run_sql_readonly` for safe retrieval

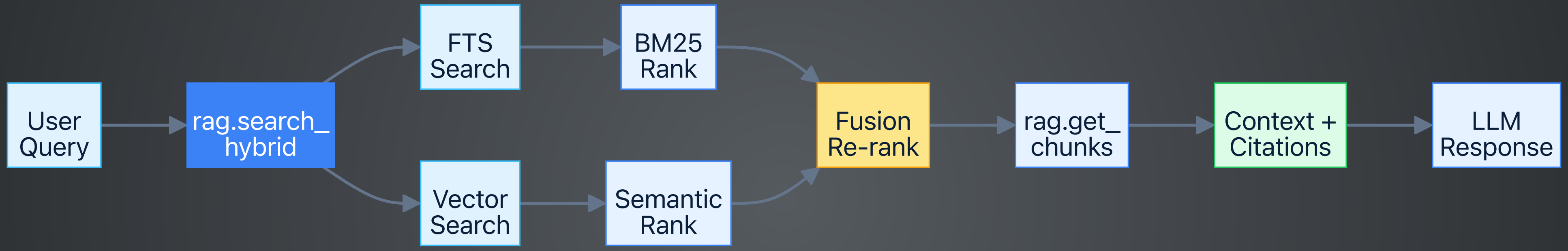
Demo: curl example

```
curl -k https://127.0.0.1:6071/mcp/query \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $TOKEN" \
  -d '{"jsonrpc":"2.0","id":1,"method":"tools/list","params":{}}'
```

Response:

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "tools": [
      {"name": "list_schemas", "description": "..."},
      {"name": "run_sql_readonly", "description": "..."}
    ]
  }
}
```

Hybrid Search walkthrough



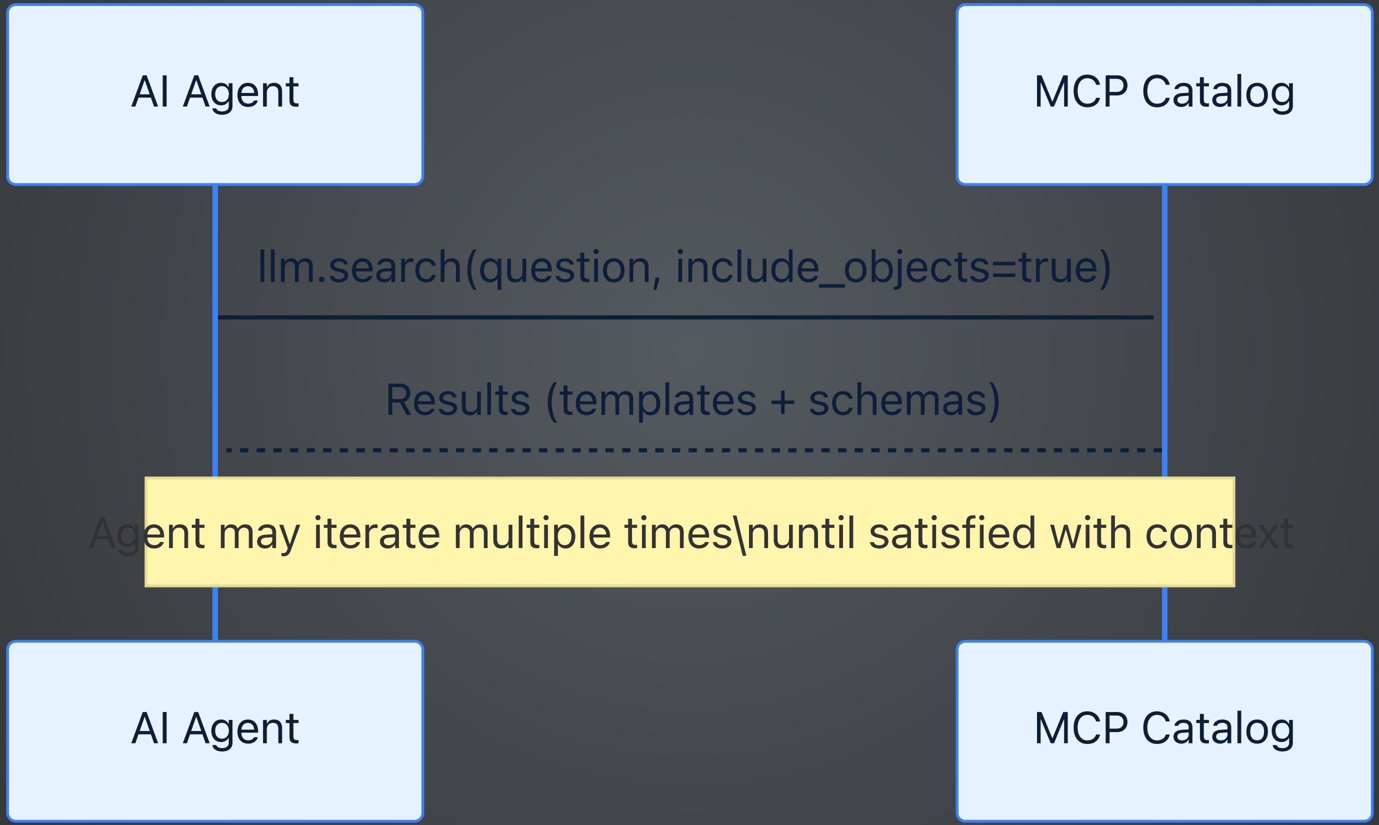
- `rag.search_hybrid` → find relevant docs (keywords + embeddings)
- `rag.get_chunks` → fetch the grounded source text
- `run_sql_readonly` → optional live data query (safely)

NL2SQL: Agentic Architecture

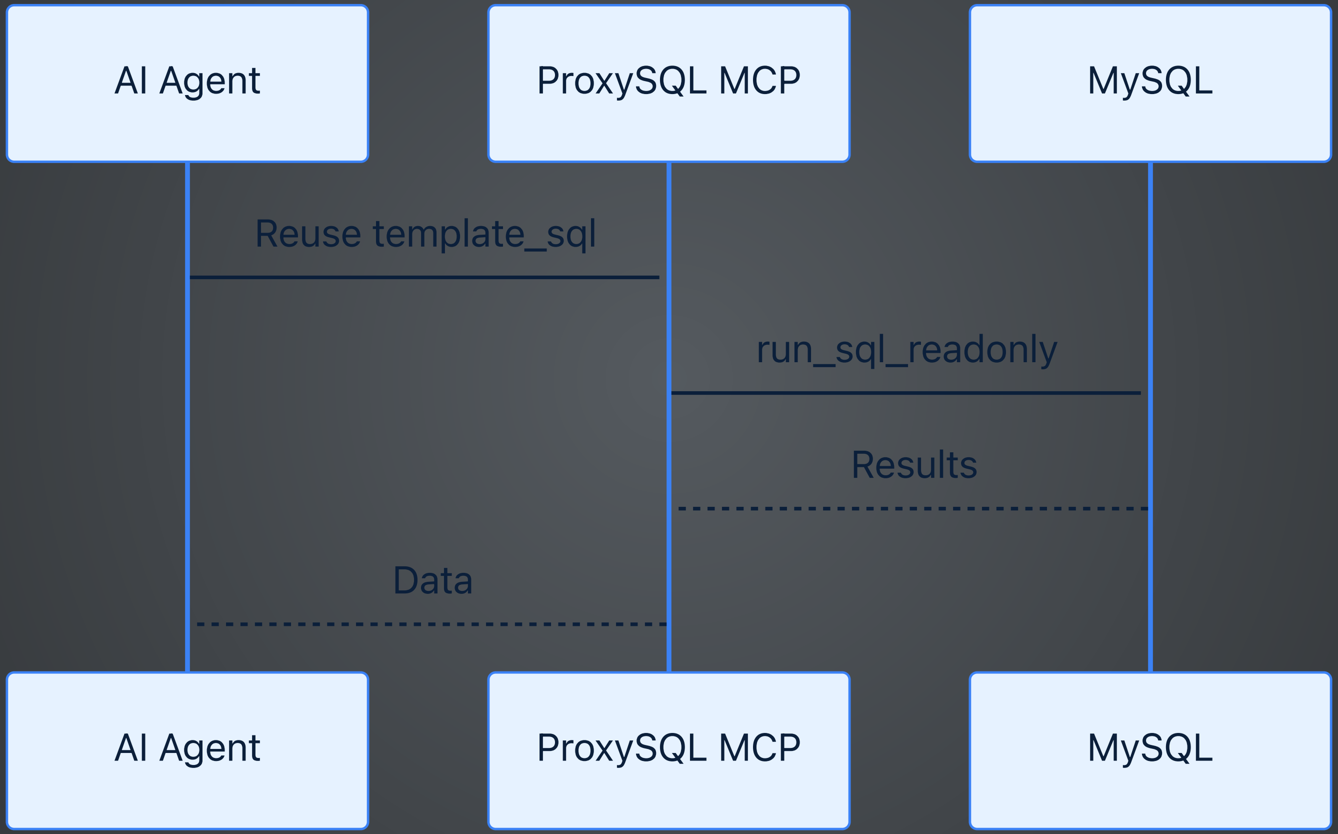
The 4-Step Agentic Workflow

- **Semantic Search**: Find relevant schemas/tables via `llm.search`
- **Analysis & Reasoning**: Match question templates or synthesize schema
- **Safe Execution**: Run via `run_sql_readonly` with query rules
- **Continuous Learning**: Store successful patterns as templates

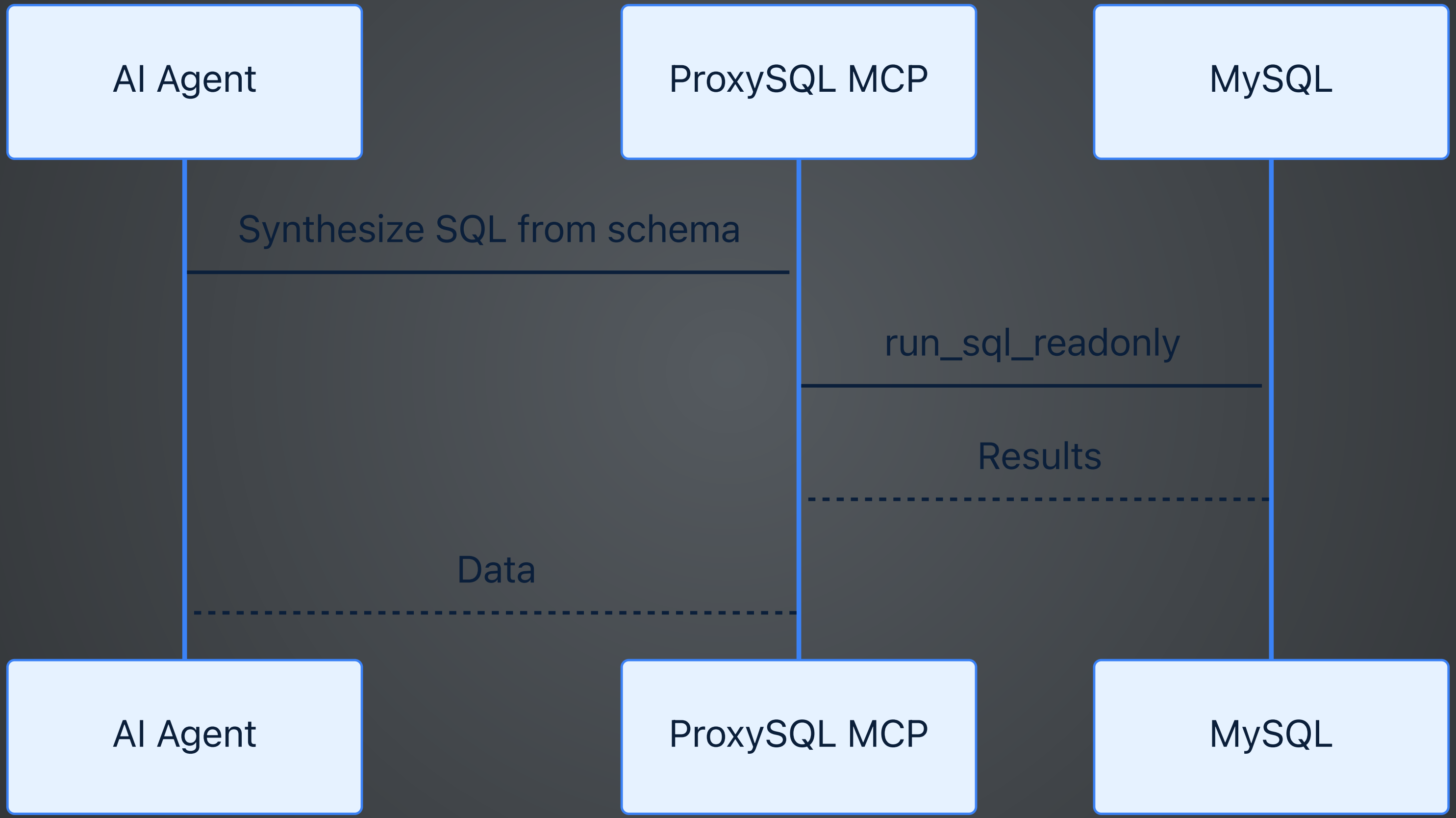
Steps 1 & 2 : Semantic Search , Analysis & Reasoning


AI Agent
MCP Catalog
llm.search(question, include_objects=true)
Results (templates + schemas)
Agent may iterate multiple times\nuntil satisfied with context
AI Agent
MCP Catalog

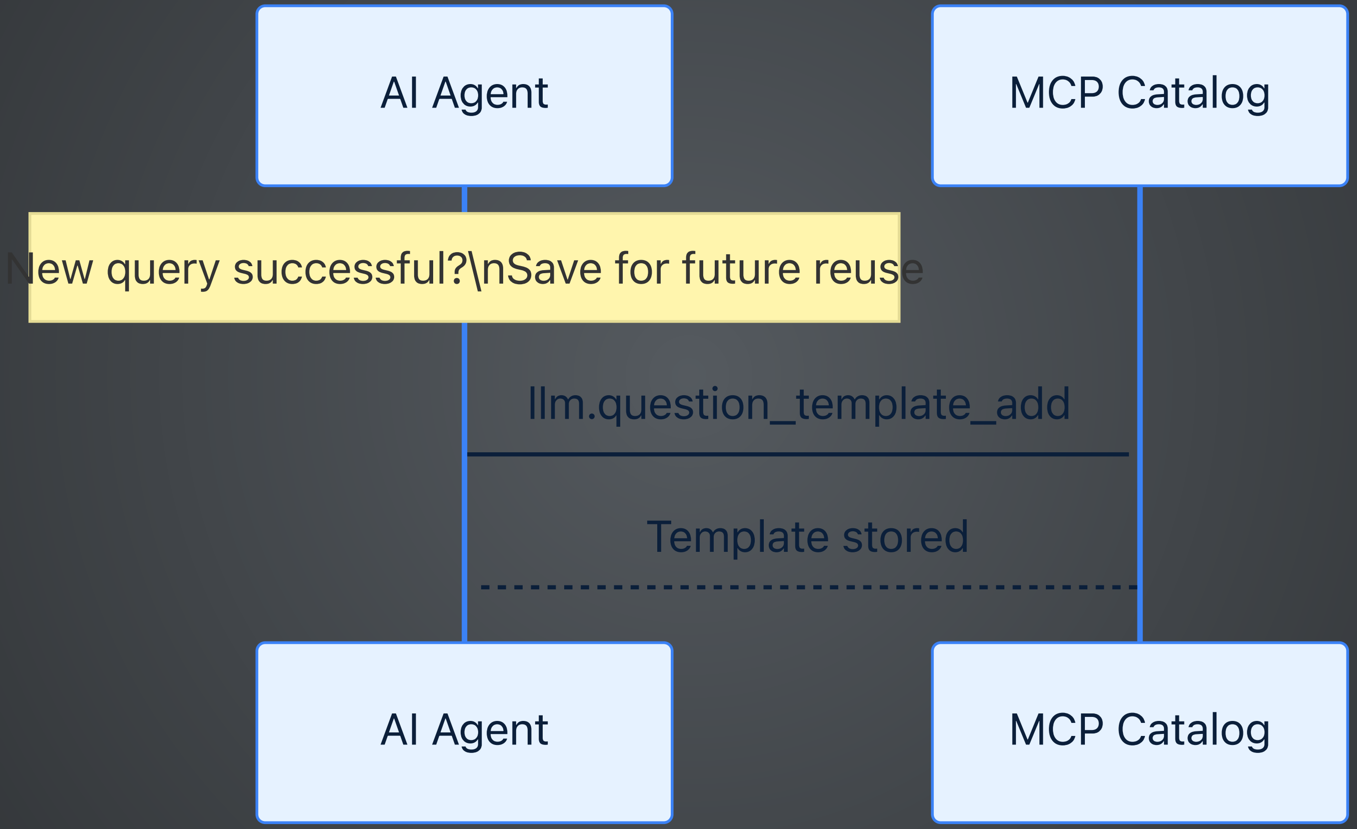
Step 3A: Execution Paths



Step 3B: Synthesis Path



Step 4: Learning (Optional)


AI Agent
MCP Catalog
New query successful?\nSave for future reuse
llm.question_template_add
Template stored
AI Agent
MCP Catalog

Agent Session Tracking & Learning

- **Session management**: track sessions, maintain context/state, audit trails
- **Continuous learning**: `llm.question_template_add`, `llm.summary_upsert`, `llm.note_add`, `llm.search`
- **Benefits**: higher NL2SQL accuracy, institutional knowledge, context-aware queries

Observability for agents: digests and counters

- MCP traffic stats to monitor behavior, performance, and rule effectiveness
- `stats_mcp_query_digest`: MCP equivalent of `stats_mysql_query_digest`
- `stats_mcp_query_tools_counters`: tool usage per schema

Deployment patterns and integration choices

- **Dev**: HTTP transport may be easiest (especially with certificate constraints)
- **Prod**: HTTPS with valid certificates + per-endpoint auth tokens
- **Integrations**: native HTTP/HTTPS clients or a stdio bridge (when needed)

Embeddings: where they fit (Roadmap / WIP)

- Automatic embedding generation for selected sources (tables, docs, logs)
- Model options: built-in local model **or** external embedding service
- Objective: avoid app-side embedding pipelines and "vector schema drift"

Real-time indexing via replication/binlog (Roadmap / WIP)

- Index without touching source tables (no `ALTER TABLE`)
- Observe changes through MySQL replication streams
- Maintain side indexes in the proxy layer (FTS + vector)

Hybrid search inside the proxy engine (Roadmap / WIP)

- Execute FTS + vector search close to metadata and (optionally) data replicas
- Return ranked candidates with citations (chunks/objects/rows)
- Use proxy rules to enforce limits (k, latency budgets, schemas, tables)

DISTANCE() semantics for AI-blind MySQL backends (Roadmap / WIP)

- Expose vector-distance functions through the proxy for older/managed MySQL
- Let apps/agents write familiar SQL while the proxy resolves vector work
- Preserve compatibility where extensions are limited

Technical Roadmap

- Harden MCP endpoints + auth/rules UX
- Expand `/mcp/rag` indexing + hybrid ranking quality
- Replication-based indexing and embedding pipelines
- Vector-distance functions and SQL surface design (DISTANCE())
- Complete `/mcp/config`, `/mcp/admin`, `/mcp/cache` endpoints

Community Roadmap

- Docs, examples, and end-to-end demos
- Performance benchmarks and optimization guides
- Agent integration samples
- Contributed question templates for NL2SQL

Wrap-up

- Proxy layer is the pragmatic place to add AI without DB migration
- MCP makes "Proxy-as-a-Tool" deterministic, auditable, and governable
- v4.0 prototypes: catalog, tools, FTS/vector/hybrid search, NL2SQL workflow
- Roadmap: replication indexing, embeddings, SQL surface (DISTANCE())

Resources:

- Docs: https://sysown.github.io/proxysql/ (Generative AI section)
- Code: https://github.com/sysown/proxysql/tree/v4.0

Q&A