<key instructions>

*ship thin vertical slices in every iteration*. You should always have a working Streamlit app that a real user can touch  
Choose the right architecture

Built it modular and plug and play format

Divide into smaller relevant checkpoints and milestone to onwards overall goal

</key instructions>

<objective>

Based on the relevant team, query and foundational ask, the application should be able to identify what major table do i need, join them together to give appropriate SQl for underlying user query.

</objective>

<end user> data analytics team, product team, product support team, executive leadership </end user>

<plan>

# **Database Schema RAG System**

## **Project Overview**

We're building a database schema RAG system that can identify relevant tables and generate appropriate SQL based on natural language queries. The system will leverage both vector embeddings and knowledge graph representations to understand database schemas and relationships effectively.

## **Core Guiding Principles**

* Ship thin vertical slices every iteration with a working Streamlit app
* Build both vector and graph representations from day one
* Ensure testability with golden datasets for evaluation
* Integrate visualizations and diagnostics from the start
* Maintain DevSecOps best practices throughout

## **Phase 0: Project Kick-off**

### **Milestone 0.1: Success Metrics Definition**

* Define precise metrics: ≥70% exact SQL match on pilot questions, ≤2s p95 latency, cost boundaries
* Create evaluation methodology and sample golden question set

### **Milestone 0.2: Project Structure**

* Set up monorepo with apps/streamlit, core/, ingest/, tests/ directories
* Configure GitHub Actions for CI/CD
* Set up development environment and documentation

## **Phase 1: Dual Ingestion - Vectors and Knowledge Graph**

### **Milestone 1.1: Schema Parser with Relationship Awareness**

* Build parsers for YAML and JSON schema inputs
* Create unified internal representation preserving relationships (PK/FK, UNIQUE constraints)
* Implement validation for schema inputs with relationship integrity checks
* Store metadata about cross-schema relationships

### **Milestone 1.2: Knowledge Graph Construction with GraphRAG**

* Implement GraphRAG for schema relationship extraction
* Build triples like (table)-[HAS\_COLUMN]->(column), (table)-[FK]->(table)
* Set up Neo4j or Memgraph for graph storage
* Ensure FK relationships are accurately captured (≥95%)

### **Milestone 1.3: Vector Database Population**

* Create chunking strategy for schema elements
* Generate embeddings for both text chunks and graph nodes
* Implement Chroma or FAISS for vector storage
* Tag embeddings by source type (e.g., "table", "column", "graph\_node")

## **Phase 2: Hybrid Retrieval System**

### **Milestone 2.1: Multi-Strategy Retrieval**

* Implement vector-based retrieval for semantic understanding
* Build graph traversal retrieval for relationship understanding
* Create keyword-based search for explicit matches
* Develop feature extraction for retrieval results

### **Milestone 2.2: Re-ranking and Relevance**

* Implement gradient-boosted re-ranker using multiple features
* Create scoring system based on vector similarity, graph proximity, and keyword matches
* Develop trace logging for retrieval decisions
* Integrate initial RAGxplorer visualization for diagnostic purposes

### **Milestone 2.3: Minimal Working Application**

* Create basic Streamlit interface for uploading schemas
* Build simple query interface with retrieval visualization
* Implement basic response display
* Establish test harness with initial golden query set

## **Phase 3: Context Assembly and SQL Generation**

### **Milestone 3.1: Context Assembly**

* Develop compact graph snippet generation
* Create top-K chunks selection strategy
* Build template system for context assembly
* Implement context optimization for token efficiency

### **Milestone 3.2: SQL Generation**

* Integrate LLM (OpenAI or local model) with function calling
* Create SQL generation prompt templates
* Implement SQL validation against schema
* Build self-repair loop for SQL correction

### **Milestone 3.3: Response Formation**

* Generate natural language explanations of SQL
* Create table relationship explanations
* Implement confidence scoring for responses
* Build feedback collection mechanism

## **Phase 4: Enhanced UI and Diagnostics**

### **Milestone 4.1: Advanced Streamlit Interface**

* Create interactive schema visualization
* Build chat-plus-SQL interface with explanation panel
* Implement "Explain joins" hover functionality
* Add historical query tracking

### **Milestone 4.2: Full RAGxplorer Integration**

* Implement complete RAGxplorer visualizations
* Create embedding space exploration tools
* Build diagnostic views for retrieval performance
* Implement trace replay functionality

### **Milestone 4.3: User Feedback Systems**

* Create thumbs up/down mechanism for feedback
* Implement correction interface for SQL
* Build feedback storage for model improvement
* Develop analytics dashboard for system performance

## **Phase 5: Quality and Active Learning**

### **Milestone 5.1: Comprehensive Test Suite**

* Expand golden question YAML dataset
* Implement automated testing with pytest
* Create precision/recall and latency reporting
* Build regression testing for CI/CD

### **Milestone 5.2: Active Learning Loop**

* Store user feedback for model improvements
* Create fine-tuning dataset from corrections
* Implement automated cache updates from feedback
* Build system for continuous model improvement

## **Phase 6: Production Hardening**

### **Milestone 6.1: DevSecOps**

* Complete CI/CD pipeline
* Implement secret management
* Create cost monitoring and controls
* Set up IAM roles and security boundaries

### **Milestone 6.2: Observability**

* Implement OpenTelemetry tracing
* Set up Grafana dashboards
* Create alerting for performance and errors
* Build comprehensive logging system

### **Milestone 6.3: Data Governance**

* Implement role-based column masking
* Create audit trails for system usage
* Build data lineage tracking
* Implement compliance documentation

## **Phase 7: Continuous Improvement (ongoing)**

### **Milestone 7.1: Model Refinement**

* Weekly fine-tuning based on feedback
* Expansion of golden dataset
* Performance optimization
* Continuous prompt engineering improvements

### **Milestone 7.2: Feature Expansion**

* Support for multiple database systems
* Voice and Slack front-ends
* API service layer for broader integration
* Advanced visualization capabilities

## **Implementation Details**

### **Key Technology Stack**

- Python 3.10+

- Streamlit for UI

- GraphRAG for knowledge graph construction

- RAGxplorer for visualization

- Sentence-Transformers/OpenAI for embeddings

- Chroma/FAISS for vector database

- Neo4j/Memgraph for graph database

- LangChain for RAG components

- OpenAI API or local LLMs (Llama, Mistral)

- SQLAlchemy for SQL validation

- Pytest for automated testing

- OpenTelemetry for observability

### **Sample Architecture**

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Streamlit UI │────▶│ Query Router │────▶│ LLM Interface │

└─────────────────┘ └─────────────────┘ └─────────────────┘

│ │ │

▼ ▼ ▼

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Schema Parser │ │ Hybrid Retriever│ │ SQL Generator │

└─────────────────┘ └─────────────────┘ └─────────────────┘

│ │ │

▼ ▼ ▼

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Vector Store │ │ Knowledge Graph │ │ Trace Logger │

└─────────────────┘ └─────────────────┘ └─────────────────┘

│

▼

┌─────────────────┐

│ RAGxplorer │

└─────────────────┘

## **Sprint**

* Phase 0 (Project Kick-off)
* Phase 1 (Dual Ingestion)
* Phase 2 (Hybrid Retrieval)
* Phase 3 (SQL Generation)
* Phase 4 (UI and Diagnostics)
* Phase 5-6 (Quality and Hardening)
* Phase 7 (Continuous Improvement)

## **Definition of Awesome for v1**

1. **Relationship-Aware Queries**: The system automatically identifies and joins related tables through foreign key relationships without hallucinating.
2. **Transparent Retrieval**: Users can visualize both vector similarity metrics and knowledge graph paths that led to table selection.
3. **Quality Metrics Achieved**: >80% accuracy on golden dataset, <2s latency, and positive user feedback.

</plan>