Technical Requirement Document (TRD)

Project Name: DataSphere

Author: Juhit Patil

Date: 29/09/2025

1. Introduction

1.1 Purpose

The purpose of this document is to outline the technical requirements for **DataSphere**, an intelligent query orchestrator I have designed. DataSphere enables users, including data analysts and business users, to query relational databases and access general knowledge through natural language, without the need to manually write SQL or perform separate searches. By integrating a Vector Database (VectorDB) for schema retrieval, Retrieval-Augmented Generation (RAG) for query refinement, and external search capabilities for unstructured information, the system ensures accurate, context-aware, and comprehensive responses.

1.2 Scope

DataSphere will:

- Accept natural language queries from users.
- Parse and convert queries into valid SQL statements using an LLM-based engine for database interactions.
- Retrieve relevant database schema, metadata, or sample data from VectorDB to ground SQL generation.
- Use RAG to enhance accuracy, resolve ambiguity, and improve schema alignment.
- Search external sources (e.g., the internet) to provide answers to general knowledge questions.
- Return outputs in user-friendly formats such as JSON, tables, or text for easy analysis and integration.

1.3 Objectives

The key objectives of DataSphere are:

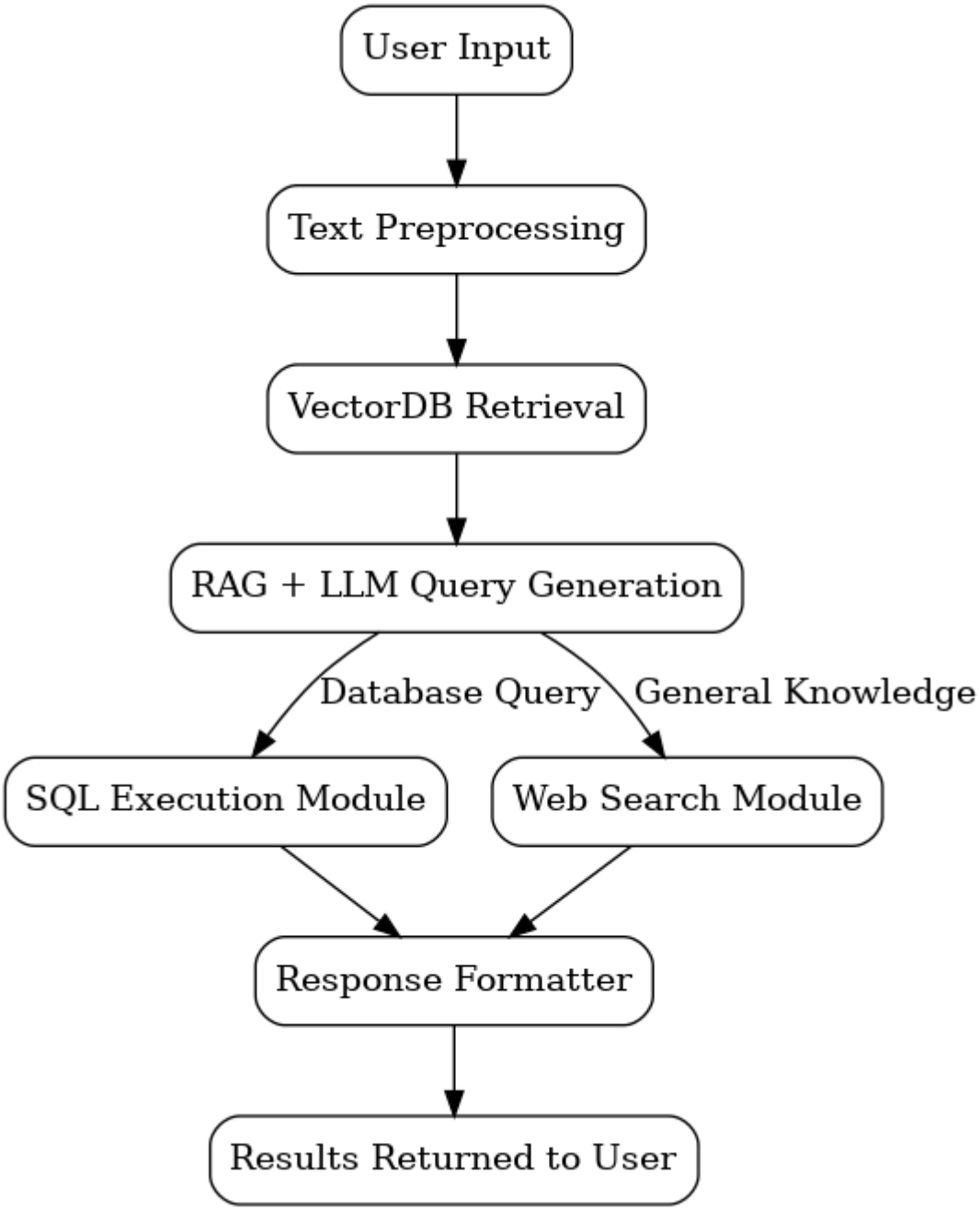
- To simplify database querying for non-technical users by enabling natural language input.
- To provide a unified interface for querying both structured data (relational databases) and unstructured information (external sources).
- To enhance query accuracy and reliability through the use of VectorDB and RAG.
- To deliver results in clear, user-friendly formats suitable for analysis and reporting.
- To ensure the system remains scalable, maintainable, and compatible across platforms.

1.4 Definitions & Acronyms

- **RAG**: Retrieval-Augmented Generation is a technique to improve query accuracy by retrieving context from VectorDB.
- **VectorDB**: A vector database used to store and retrieve embeddings of database schema and metadata.
- **LLM**: Large Language Model is the AI model responsible for transforming natural language into SQL.
- **SQL**: Structured Query Language is the standard language for querying relational databases.

2. System Overview

2.1 Architecture



2.2 Components

1. **Text Preprocessing Module** – Handles tokenization, normalization, and stop-word removal.
2. **VectorDB Module** – Stores schema embeddings and sample queries. Retrieves relevant context for the LLM.
3. **RAG Module** – Combines retrieved context with the user query and LLM to generate accurate SQL queries.
4. **SQL Execution Module** – Executes SQL on the target relational database (PostgreSQL/MySQL).
5. **Response Formatter** – Converts SQL results or search responses into structured JSON output.

Component Interactions (with respect to Endpoints):

- **For `/rag` (Database/Table Queries):**
    - User input → Text Preprocessing → VectorDB Retrieval → RAG Module → SQL Execution → Response Formatter.
- **For `/ask` (External Queries):**
    - User input → Text Preprocessing → External Search API/LLM → Response Formatter.

3. Functional Requirements

ID	Requirement	Description	Priority
1	User Query Input	Accept natural language queries from UI	High
2	Context Retrieval	Fetch relevant schema info from VectorDB	High
3	SQL Generation	Convert user query to SQL using RAG	High
4	SQL Execution	Run SQL safely against database	High
5	Result Formatting	Return results in JSON/CSV/table	Medium
6	Error Handling	Gracefully handle syntax errors or failed queries	High

4. Non-Functional Requirements

Category	Requirement	Priority
Performance	System should respond within 2-3 seconds for queries on a DB with <1M records	High
Scalability	Must support multiple concurrent users	Medium
Maintainability	Modular Python code with clear documentation	High
Compatibility	Python 3.10+, works on Linux & MacOS	Medium

5. Technology Stack

Layer	Technology
Backend	Python 3.10+, FastAPI
Database	PostgreSQL / ChromaDb
LLM	Generative AI
RAG Framework	Custom Python Module

6. Data Flow

The data flow of the DataSphere system follows a clear sequence of operations to transform natural language into meaningful results:

1. **User Query Submission** – The user enters a natural language query (e.g., *“What’s the role of Charlie”*).
2. **Preprocessing** – The system normalizes text, removes stop words, and resolves synonyms or abbreviations to improve query clarity.
3. **Schema Retrieval** – VectorDB retrieves the top-*k* relevant schema embeddings, metadata, or sample values related to the query.
4. **RAG Processing** – The RAG module combines the retrieved embeddings with the user query and refines it to produce accurate SQL statements or identify when the request should be redirected to a web search.
5. **SQL Execution / Web Search** – Depending on query type:
    - **Database queries** are executed safely against the connected relational database.
    - **General knowledge queries** are routed to the external search component for retrieval.
6. **Response Formatting** – Results are formatted into user-friendly outputs (JSON, CSV, or tables) and returned to the interface.
7. **Feedback Loop** – Optional feedback from the user is logged to continuously refine system accuracy.

7. Security Considerations

To ensure system reliability and protect sensitive data, the following security measures are applied:

- **Encryption** – Database credentials, API keys, and sensitive query data are encrypted both in transit (TLS/SSL) and at rest.
- **Authentication and Access Control** – Only authorized users and applications can submit queries or access results.
- **Input Validation** – User inputs are sanitized to prevent SQL injection, malicious payloads, or unauthorized system access.
- **Audit Logging** – All queries, results, and system interactions are logged for traceability and compliance monitoring.
- **Data Privacy** – Personally Identifiable Information (PII) and sensitive records are masked or anonymized where applicable.
- **Error Reporting** – Error logs avoid leaking sensitive schema or system details while still aiding in debugging.

8. Testing Strategy

A robust testing strategy ensures DataSphere functions as expected under a variety of scenarios:

- **Unit Tests**
  - Validate preprocessing logic such as tokenization, normalization, and stop-word removal.
  - Test VectorDB retrieval to confirm relevant schema embeddings are correctly matched.
  - Verify SQL generation and web search routing by the RAG module.
- **Integration Tests**
  - Full end-to-end validation from user input → preprocessing → schema retrieval → SQL execution or web search → response formatting.
  - Ensure smooth interaction between components (LLM, VectorDB, database, web search, and response formatter).
- **Error Handling Tests**
  - Invalid queries (e.g., malformed input or unsupported requests).
  - Missing or incomplete schema information.
  - Network failures (e.g., VectorDB or database connection issues).
  - Graceful fallback when web search APIs are unavailable.
- **Performance Tests**
  - Validate system response times (2–3 seconds for <1M records).
  - Test scalability under concurrent user queries.
- **Security Tests**
  - Penetration testing for SQL injection and input manipulation.
  - Verification of encryption, authentication, and access control policies.