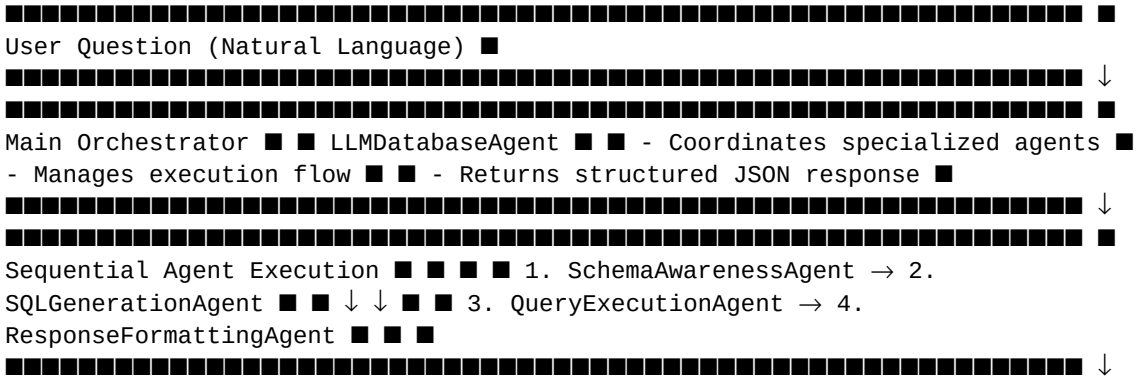
User Question (Natural Language)
Main Orchestrator
LLMDatabaseAgent
- Coordinates specialized agents
- Manages execution flow
- Returns structured JSON response
Sequential Agent Execution
1. SchemaAwarenessAgent → 2. SQLGenerationAgent
3. QueryExecutionAgent → 4. ResponseFormattingAgent

■■ ■
Natural Language Response + Results ■
■■

```
# Discovers all tables, columns, and data types schema = { "web_activity": {
"columns": ["id", "user_id", "website_name", "time_spent", "activity_date"],
"sample_data": [...] # Actual data samples }, "github_activity": {...} } #
Provides exact database values to LLM schema_info = "website_name: 'youtube.com',
'github.com', 'stackoverflow.com'..."
```

```
# Receives schema info and user question prompt = """ Schema: {database structure
with samples} Question: "How much time did I spend on YouTube today?" Rules: 1.
Use SELECT only (READ-ONLY) 2. Include WHERE user_id = %s 3. Use exact database
names from schema 4. Return JSON with sql_query Generate SQL query... """ #
Generates parameterized SQL sql = "SELECT website_name, SUM(time_spent) as total
FROM web_activity WHERE user_id = %s AND website_name = 'youtube.com' GROUP BY
website_name"
```

```
# Step 1: Security validation is_safe = security_guard.validate_query(sql, user_id) # Checks: dangerous keywords, user_id filtering, modification patterns # Step 2: Execute query results = db_manager.execute(sql, params=[user_id]) # Step 3: Return structured results return { "success": True, "results": [...], # Actual data "row_count": 5, "columns": ["website_name", "total"] }
```

```
# Receives raw results results = [{"website": "youtube.com", "total": 120}] # Generates natural language response prompt = """ Question: "How much time did I spend on YouTube today?" Results: {results} Provide a conversational response explaining the data. """ response = llm.generate(prompt) # Output: "You spent 120 minutes on YouTube today."
```

```
{ "response": "You spent 120 minutes on YouTube today.", "results": [...], "sql_query": "...", "timestamp": "...", "agents_used": [...] }
```

```
def process_question(question, user_id): # 1. Get Database Schema schema_info = schema_agent.get_database_schema() # Returns: Tables, columns, sample data # 2. Generate SQL Query sql_result = sql_agent.generate_sql_query( question=question, user_id=user_id, schema_info=schema_info ) # Returns: {"sql_query": "...", "reasoning": "...", "confidence": 0.95} # 3. Execute Query query_result = query_agent.execute_query( sql_query=sql_result["sql_query"], user_id=user_id ) # Returns: {"success": True, "results": [...], "row_count": 5} # 4. Format Response final_response = response_agent.format_response( question=question, query_results=query_result, sql_query=sql_result["sql_query"] ) # Returns: {"response": "...", "results": [...], "success": True} return final_response
```

```
# ■ BLOCKED Operations - DROP, DELETE, UPDATE, INSERT, ALTER - Multiple statements (; DROP TABLE...) - System table access - Data modification (arithmetic: +, -, *, /) - Missing user_id filtering # ■ ALLOWED Operations - SELECT queries only - Parameterized queries (WHERE user_id = %s) - UNION queries (with validation) - Aggregation functions (SUM, COUNT...) # Example Validation Result { "is_safe": True, "reason": "Query is safe" } # OR { "is_safe": False, "reason": "Dangerous keyword detected: DROP" }
```

```
# Removes: - Script injections (...) - Malicious JavaScript - SQL injection attempts - Excessive response length
```

Step 1: SchemaAwarenessAgent ■■■ ■
Returns: ■ ■ - web_activity table structure ■ ■ - Sample data showing
youtube.com ■ ■ - Column names and types ■
■■■ Step 2: SQLGenerationAgent
■■■ ■ Returns: ■ ■ SQL: SELECT
website_name, ■ ■ SUM(time_spent) as total ■ ■ FROM web_activity ■ ■ WHERE
user_id = %s ■ ■ AND website_name = 'youtube.com' ■
■■■ Step 3: QueryExecutionAgent
■■■ ■ Validates: ■ Safe query ■ ■
Executes: Query on database ■ ■ Returns: ■ ■ [{website_name: "youtube.com", ■ ■
total: 120}] ■ ■■■ Step 4:
ResponseFormattingAgent ■■■ ■
Returns: ■ ■ "You spent 120 minutes on ■ ■ YouTube today." ■
■■■

```
# Layer 1: Security Guards query_guard.validate_query(sql, user_id) # Layer 2: Query Execution Agent query_agent.check_for_modification(sql) # Layer 3: Database Manager db_manager.execute_with_validation(sql, params)
```

```
graph TB USER[User Question] --> ORCH[LLMDatabaseAgent Orchestrator] ORCH --> AG1[SchemaAwarenessAgent] AG1 --> AG2[SQLGenerationAgent] AG2 --> AG3[QueryExecutionAgent] AG3 --> AG4[ResponseFormattingAgent] AG1 --> DB[(MySQL Database)] AG3 --> DB AG2 --> LLM1[Gemini LLM] AG4 --> LLM2[Gemini LLM] AG3 --> SEC1[QuerySecurityGuard] AG4 --> SEC2[ResponseSecurityGuard] AG4 --> USER2[Natural Language Response] classDef agent fill:#e8f5e8 classDef tool fill:#e1f5fe classDef security fill:#ffebee class AG1,AG2,AG3,AG4 agent class DB,LLM1,LLM2 tool class SEC1,SEC2 security
```

User: "Add 1 to my commit count" ↓ System: Detects modification attempt ↓ Response: "I can only view your data, not modify it. Here's your current commit count: 5"

User: "Show all my activity for today" ↓ System: Uses UNION to combine web_activity + github_activity ↓ SQL: SELECT 'Web' as type, website_name as name, ... UNION ALL SELECT 'GitHub' as type, repo_name, ...

agents/ ■■■ core/ ■ ■■■ llm_agent.py # Main orchestrator ■ ■■■ schema_agent.py # Schema discovery ■ ■■■ sql_agent.py # SQL generation ■ ■■■ query_execution_agent.py # Query execution ■ ■■■ response_formatting_agent.py # Response formatting ■■■ guards/ ■ ■■■ security_guards.py # Security validation ■■■ schemas.py # Pydantic schemas backend/ ■■■ api/ ■ ■■■ agent_endpoints.py # API endpoints ■■■ database/ ■■■ db_manager.py # Database operations streamlit_app.py # Frontend UI main.py # Flask backend