**Expanded Subtasks for Key Sections**

**2. Structured Data Handling with Data Insights (DI)**

**Story:**  
As a user, I want the system to handle queries on structured data (e.g., CSV files) and return SQL-query-based insights accurately.

**Acceptance Criteria:**

* The **Data Insights (DI)** agent executes SQL queries against structured data files.
* SQL results are processed and formatted for clarity.
* **DI** agent connects to the shared SQL database to check session history.

**What Needs to Happen in the Code (Expanded into Subtasks)**

**2.1 Generate SQL Queries Dynamically Based on User Prompts**

**Goal:** Dynamically generate SQL queries based on user intent and prompt structure.

**Subtasks:**

1. **Analyze User Prompt:**
   * Parse user queries to identify table names, filters, and operations (e.g., SELECT, UPDATE).
   * Use natural language processing (NLP) to extract relevant SQL keywords.
2. **Build Query Templates:**
   * Develop reusable SQL templates for common query patterns.
   * Ensure templates support parameterized queries to prevent SQL injection.
3. **Dynamic Query Generation:**
   * Map user intent to predefined SQL templates.
   * Dynamically fill parameters in templates based on user input.
4. **Validation:**
   * Validate generated queries for syntax and logical errors.
   * Add unit tests for common query scenarios.

**2.2 Retrieve Relevant Chat History from SQL Database Before Generating a Query**

**Goal:** Retrieve relevant context from previous user interactions for continuity in SQL query generation.

**Subtasks:**

1. **Fetch Session History:**
   * Query SQL history using session\_id and chat\_id.
   * Retrieve the latest n interactions for contextual understanding.
2. **Parse and Filter Relevant Context:**
   * Extract critical parameters (e.g., table names, filters) from historical context.
   * Remove redundant or irrelevant information.
3. **Handle Edge Cases:**
   * Gracefully handle missing or incomplete session history.
   * Provide fallback mechanisms for history retrieval failures.

**2.3 Ensure SQL Results Are Converted into Readable Text**

**Goal:** Format SQL query results into a user-friendly and readable format.

**Subtasks:**

1. **Fetch Query Results:**
   * Execute SQL queries and fetch results in tabular format.
   * Handle errors like empty result sets or query failures.
2. **Format Results:**
   * Convert raw SQL data into a readable JSON or string-based response.
   * Ensure column names are human-readable.
3. **Add Metadata:**
   * Include metadata (e.g., number of rows returned, query execution time).
4. **Validation:**
   * Validate the final output for correctness and readability.

**2.4 Return Results to the Orchestrator Agent**

**Goal:** Send processed SQL results back to the **Orchestrator** agent for further handling.

**Subtasks:**

1. **Package Query Results:**
   * Package SQL results into a standardized response format.
   * Include source agent metadata.
2. **Error Handling:**
   * Add fallback mechanisms for partial or failed query execution.
   * Log all query failures for debugging.
3. **Send Results:**
   * Implement API or internal messaging calls to send the response back to the **Orchestrator**.

**Subtask Summary Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Subtask ID** | **Description** | **Assignee** | **Estimated Time** |
| **2.1.1** | Analyze User Prompt | Backend Developer | 4 hours |
| **2.1.2** | Build Query Templates | Backend Developer | 3 hours |
| **2.2.1** | Fetch Session History | Backend Developer | 2 hours |
| **2.3.1** | Format Query Results | Backend Developer | 3 hours |
| **2.4.1** | Package Query Results | Backend Developer | 2 hours |

**3. Unstructured Data Handling with Enterprise Search (ES)**

**Story:**  
As a user, I want the system to search through unstructured data and return insights, including information extracted from images.

**Acceptance Criteria:**

* The **Enterprise Search (ES)** agent processes user queries on unstructured data.
* **ES** handles image text extraction.
* **ES** connects to the shared SQL database for session history retrieval.
* Embeddings are fetched from the **Vector Database** for semantic search.

**What Needs to Happen in the Code (Expanded into Subtasks)**

**3.1 Implement Logic for Unstructured Data Searches**

**Subtasks:**

1. Build a document indexing pipeline.
2. Enable semantic search over indexed documents.
3. Implement ranking algorithms to prioritize search results.
4. Optimize searches for scalability.

**3.2 Support OCR or Image Text Extraction Mechanisms**

**Subtasks:**

1. Integrate OCR libraries (e.g., Tesseract).
2. Implement preprocessing techniques for image clarity.
3. Validate text extraction quality.

**3.3 Use Embeddings for Similarity-Based Querying**

**Subtasks:**

1. Generate embeddings using pre-trained models.
2. Store embeddings in the Vector Database.
3. Compare embeddings for semantic search.

**3.4 Validate Context from Shared Chat History Before Processing**

**Subtasks:**

1. Fetch historical context from SQL database.
2. Parse previous queries for relevant context.
3. Validate extracted context for follow-up accuracy.

**3.5 Return Results to the Orchestrator Agent**

**Subtasks:**

1. Package search results in a standardized format.
2. Annotate results with metadata.
3. Send results back to the Orchestrator.

**5. Follow-Up Query Handling Across Agents**

**Story:**  
As a user, I want follow-up queries to route correctly to the **DI** or **ES** agent based on the context of the original query.

**5.1 Implement Context-Aware Routing in Orchestrator**

**Subtasks:**

1. Build logic for analyzing follow-up context.
2. Route follow-up queries dynamically to the appropriate agent.
3. Handle ambiguity in routing decisions.

**5.2 Ensure Cross-Reference of Chat History**

**Subtasks:**

1. Fetch previous session context.
2. Validate references against historical queries.
3. Prevent mismatched agent routing.

**5.3 Annotate Follow-Up Responses**

**Subtasks:**

1. Add metadata to indicate follow-up nature.
2. Highlight historical references in the response.

**6. Report Generation Agent**

**Story:**  
As a user, I want aggregated insights from both DI and ES agents summarized into a report.

**6.1 Implement Response-Merging Logic**

**Subtasks:**

1. Define aggregation rules for multi-agent responses.
2. Handle conflicting information gracefully.

**6.2 Standardize Report Formatting**

**Subtasks:**

1. Create report templates.
2. Ensure clarity and readability.

**6.3 Store Aggregated Response with Metadata**

**Subtasks:**

1. Save final reports in the SQL database.
2. Include timestamps and agent metadata.

**7. Stateless Orchestrator Workflow**

**Story:**  
As a user, I want the **Orchestrator** agent to route and aggregate data without maintaining internal state.

**7.1 Avoid Session Persistence**

**Subtasks:**

1. Ensure each request starts fresh.
2. Rely on shared SQL database for session state.

**7.2 Retrieve Session Data from SQL History**

**Subtasks:**

1. Query session state on each interaction.
2. Validate retrieved context.

**7.3 Ensure Dynamic Query Routing**

**Subtasks:**

1. Implement routing logic for dynamic queries.
2. Validate routing rules.