## Data Model for Requirements Gathering Agent

This data model focuses on storing project information, generated documents, and analysis results for the Requirements Gathering Agent. We’ll use a relational database (SQL) for its ACID properties, strong data integrity, and widespread tooling support. A NoSQL database could be considered for specific aspects (like unstructured document storage) if the project scales significantly, but for now, a relational model offers the best balance of features and simplicity.

**I. Core Entities and Relationships:**

* **Projects:** Represents a single project.
  + project\_id (INT, Primary Key, Auto-Increment)
  + project\_name (VARCHAR(255), Not Null)
  + description (TEXT)
  + created\_at (TIMESTAMP)
  + updated\_at (TIMESTAMP)
  + user\_id (INT, Foreign Key referencing Users)
* **Users:** Represents users of the application.
  + user\_id (INT, Primary Key, Auto-Increment)
  + username (VARCHAR(255), Unique, Not Null)
  + email (VARCHAR(255), Unique, Not Null)
  + password\_hash (VARCHAR(255), Not Null) // Securely store password hashes
* **Documents:** Represents generated PMBOK documents.
  + document\_id (INT, Primary Key, Auto-Increment)
  + project\_id (INT, Foreign Key referencing Projects, Not Null)
  + document\_type (VARCHAR(255), Not Null, e.g., ‘Project Charter’, ‘Risk Management Plan’)
  + filename (VARCHAR(255), Not Null)
  + filepath (VARCHAR(255), Not Null)
  + content (MEDIUMTEXT) // Store the generated document content (consider LOB for very large documents)
  + format (VARCHAR(255), Not Null, e.g., ‘docx’, ‘markdown’, ‘json’)
  + created\_at (TIMESTAMP)
  + updated\_at (TIMESTAMP)
  + pmbok\_compliance\_score (INT) // Score from 0-100
  + quality\_score (INT) // Overall quality score from 0-100
* **Files:** Represents files discovered during project analysis.
  + file\_id (INT, Primary Key, Auto-Increment)
  + project\_id (INT, Foreign Key referencing Projects, Not Null)
  + filepath (VARCHAR(255), Not Null)
  + filename (VARCHAR(255), Not Null)
  + relevance\_score (INT) // Score from 0-100
  + category (VARCHAR(255), e.g., ‘Planning’, ‘Development’, ‘Documentation’)
  + content\_type (VARCHAR(255), e.g., ‘markdown’, ‘text’, ‘json’)
  + created\_at (TIMESTAMP)
* **AIProviders:** Stores information about supported AI providers.
  + provider\_id (INT, Primary Key, Auto-Increment)
  + provider\_name (VARCHAR(255), Unique, Not Null, e.g., ‘Azure OpenAI’, ‘Google Gemini’)
  + api\_key (VARCHAR(255)) // Consider secure storage solutions for API keys.
  + endpoint (VARCHAR(255))
* **AnalysisResults:** Stores results from the project analysis phase.
  + analysis\_id (INT, Primary Key, Auto-Increment)
  + project\_id (INT, Foreign Key referencing Projects, Not Null)
  + total\_files\_found (INT)
  + total\_relevant\_files (INT)
  + high\_value\_sources (TEXT) // JSON array of high-value file paths
  + context\_utilization (FLOAT)
  + created\_at (TIMESTAMP)
* **Context:** Stores the context used for document generation. This might need a separate table or a NoSQL solution for very large projects. For now:
  + context\_id (INT, Primary Key, Auto-Increment)
  + analysis\_id (INT, Foreign Key referencing AnalysisResults, Not Null)
  + context\_data (MEDIUMTEXT) // Consider JSON or a specialized format for structured data.

**II. Entity-Relationship Diagram (ERD):**

Projects \*---1 Users (one-to-many)  
Projects 1---\* Documents (one-to-many)  
Projects 1---\* Files (one-to-many)  
Projects 1---1 AnalysisResults (one-to-one)  
AnalysisResults 1---\* Context (one-to-many)

**III. Data Types and Constraints:**

All data types are suggested above. Constraints like NOT NULL and UNIQUE are crucial for data integrity. Foreign keys enforce referential integrity between tables.

**IV. Indexing:**

* **Projects:** Index project\_name and user\_id.
* **Documents:** Index project\_id, document\_type, and created\_at.
* **Files:** Index project\_id, relevance\_score, and category.
* **AnalysisResults:** Index project\_id.

**V. Normalization:**

The model is largely in 3NF (Third Normal Form). Further normalization might be needed depending on future requirements.

**VI. Database Technology:**

PostgreSQL or MySQL are recommended for their robustness, scalability, and extensive support for features like JSON storage and full-text search (if needed for searching document content).

**VII. Scalability and Performance:**

* **Database Sharding:** For extremely large numbers of projects, database sharding can improve performance.
* **Caching:** Implement caching mechanisms (e.g., Redis) to reduce database load for frequently accessed data.
* **Load Balancing:** Use a load balancer to distribute database traffic across multiple servers.
* **Database Optimization:** Regularly analyze query performance and optimize database indexes.

**VIII. Data Security and Privacy:**

* **Secure Password Storage:** Use strong hashing algorithms (like bcrypt) to store passwords securely.
* **API Key Management:** Employ secure methods for storing and managing API keys (e.g., dedicated secrets management service).
* **Data Encryption:** Encrypt sensitive data at rest and in transit.
* **Access Control:** Implement appropriate access control mechanisms to restrict data access based on user roles.
* **Compliance:** Ensure the data model and security practices comply with relevant regulations (e.g., GDPR, CCPA).

**IX. Future Considerations:**

* **Full-text search:** Add full-text indexing to the content column in the Documents table to enable efficient searching within generated documents.
* **Document versioning:** Implement a mechanism to track different versions of generated documents.
* **Collaboration features:** Add features to allow multiple users to collaborate on the same project.

This detailed data model provides a solid foundation for the Requirements Gathering Agent. Remember to adapt and refine it as the project evolves and new requirements emerge. The choice of specific database technology and security implementations should be based on a thorough risk assessment and the organization’s security policies.