Scalable Data Warehouse & Query Engine in Django

Overview

Your task is to build a prototype of a data warehouse system using Django. This system should be capable of ingesting large volumes of both structured and unstructured data—including nested or semi-structured formats—and support efficient querying. In addition, every change to the structured data must be tracked.

Requirements

1. **Data Ingestion and Storage**
    - **Input Types:**
        - Support ingestion of both structured data (e.g. CSV/JSON with defined schemas) and unstructured or semi-structured data (e.g. free-form text, nested JSON).
        - The system should be able to handle millions of rows of data.
    - **Data Warehouse Design:**
        - Architect a storage solution that scales horizontally. You may use the databases of your choice (e.g., PostgreSQL, MySQL, etc.) to achieve scalability and performance.
        - Demonstrate how you would store nested and unstructured data in a way that supports efficient querying.
    - **Change History Tracking:**
        - Implement functionality to maintain a complete change history of every fact in your structured data.
        - This historical data should be designed to facilitate plotting of graphs and charts showing trends and changes over time.
    - **Assumptions:**

- Clearly call out any assumptions regarding data formats, storage choices, scalability constraints, and historical data tracking in your documentation.

2. **API and Query Engine**
   - **Django API:**
     - Develop your solution using Django.
     - Expose RESTful (or GraphQL) API endpoints for data ingestion and querying.
     - The query endpoint should support filtering based on both structured fields (e.g., name, date) and contents of unstructured data (e.g., keywords or phrases).
   - **Query Capabilities:**
     - Implement advanced query features, such as full-text search for unstructured data and basic aggregation (e.g., counts, group by) for structured data.
   - **Performance:**
     - Your design should consider performance optimizations to handle large datasets efficiently.
3. **Bonus: Visualization Layer**
   - Provide a simple dashboard or visualization component that displays query results and historical trends (e.g., charts or graphs showing changes over time).
   - Note that the visual component is a bonus and will not be the primary focus during evaluation.
4. **Use of AI Tools**
   - The use of AI tools to assist in code generation is encouraged.
5. **Documentation**
   - Provide clear instructions on how to set up and run your Django solution.
   - Document any assumptions, trade-offs, and design decisions you make.

- Include a brief architecture overview explaining how your design addresses scalability, flexibility (to handle nested/unstructured data), and performance, along with your approach for tracking change history.

6. **Time Constraint**
    - This assignment must be completed within 48 hours. Plan your implementation accordingly, focusing on core backend functionality.
7. **Evaluation Criteria**
    - **Scalability:** How well does your system design handle increasing data volumes?
    - **Flexibility:** How effectively can your solution accommodate a wide variety of data types and structures?
    - **Historical Data Tracking:** How well does your implementation capture and store change history for structured data to enable future visualization of trends?
    - **Code Quality and Documentation:** Clear, maintainable code along with comprehensive documentation on assumptions and design decisions.
    - **API Design:** Usability and correctness of your Django API endpoints.
    - **Bonus:** Any visualization or interactive components provided.

Deliverables

- A working Django codebase with clear instructions for setup and execution.
    - If you can host it, even better. Just pass on the repository link
- A README (or similar document) detailing:
    - Your design approach.
    - Assumptions made.
    - How to run your code.
    - How your system tracks change history and potential improvements or considerations for a production environment.

This assignment challenges you to design and implement a scalable, flexible data warehouse solution using Django, ensuring that every change in structured data is recorded for future trend visualization. The use of AI tools to assist in coding is encouraged, provided that the final submission maintains high standards of code quality and clarity. Good luck, and we look forward to reviewing your solution!

Data examples: