Healthcare System for Patient Monitoring

NAME: RAMZ DAOUD & OMAR JABOUR

****

1- Requirements Analysis .

.Patient Data Storage: Documentary data such as name and medical history → Document DB

.Storing Real-Time Biometrics: Such as heart rate and blood pressure → Time-Series DB

.Managing Doctor-Patient Relationships: Complex Relationships → Graph DB

Analysis of Real-Time Patient Data: Storing and analyzing massive data organized by time and .region → Column-Family DB

.Emergency Alerts: Storing abnormal alert data quickly → Key-Value DB

.Partitioning Patients by Geographic Region

.Providing High Availability for Emergency Alerts through Replication

.Ensuring Strong Consistency of Patient Medical Records

.Eventual Consistency of Real-Time Sensor Data

.Fault tolerance (simulating node failures or network outages and recovering the system)

.Performance optimization: through indexing, caching, and replication settings adjustments

2. Database Selection and Justification

Type of DB	System	Justification	Trade-offs
Document DB	MongoDB	Flexible for unstructured patient records with strong consistency using writeConcern.	Needs replication management for availability.
Time-Series DB	InfluxDB	Optimized for high-frequency time-series data like heartbeats.	Eventual consistency; not suited for static records.
Graph DB	Neo4j	Ideal for complex doctor-patient relationship queries.	Can be less performant with very large data.
Column-family DB	Cassandra	Handles large-scale analytics distributed by region with partitioning and replication.	Eventual consistency by default; tuning needed for strong consistency.
Key-Value DB	Redis	Fast, temporary storage for real-time alerts with replication.	Short-lived data; requires cache management.

3. Data Model Design

MongoDB: Collections for patients and doctors, linking patients to doctors via doctor_id.

Neo4j: Nodes labeled Doctor and Patient connected by TREATS relationships.

InfluxDB: Time-stamped heartbeat measurements tagged with patient_id.

Cassandra: Table patient_analytics partitioned by region and patient_id, storing metrics ordered by measurement time descending.

Redis: Keys formatted as alert:{patient_id} to store alerts for abnormal readings.

4. Partitioning and Replication

-Patients are partitioned by region in Cassandra for efficient localized queries.

-MongoDB ensures strong consistency with majority write concern (w=majority).

-Cassandra replication factor can be configured to enhance availability.

-Redis supports replication for high availability of alert notifications.

-Eventual consistency applies to sensor data in InfluxDB and Cassandra.

5. Prototype Implementation

--Connects to all databases with proper authentication and configuration.

-Performs CRUD operations in Neo4j for doctors, patients, and their relationships.

-Reads patient and doctor data from MongoDB.

-Writes heartbeat measurement data to InfluxDB

-Inserts patient analytics data into Cassandra.

-Sets and clears alerts in Redis based on randomized heartbeat monitoring.

Conclusion

This healthcare monitoring system prototype demonstrates the effective integration of multiple specialized database technologies, specifically designed to meet the unique requirements of medical data management. Leveraging document, time series, graph, column families, and key-value databases, the system achieves a balance between strong consistency for critical patient records and high availability and performance for real-time analytics and alerts. The design supports scalability through partitioning and replication, while fault-tolerant mechanisms ensure resilience to failures. Continuous improvements, such as improved replication strategies, automated error mitigation, and rigorous error handling, will enhance the system's reliability and operational excellence in real-world .healthcare environments