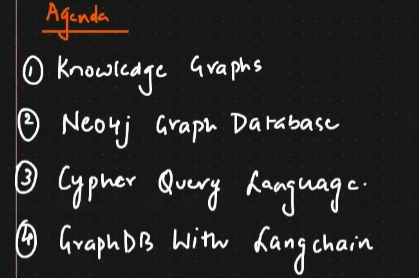
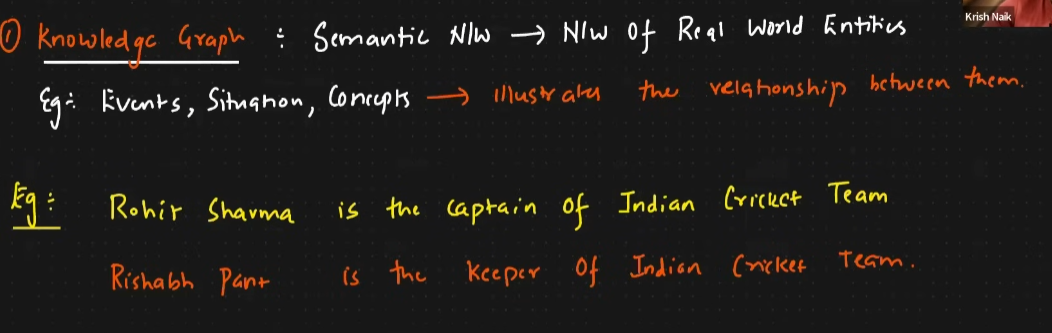
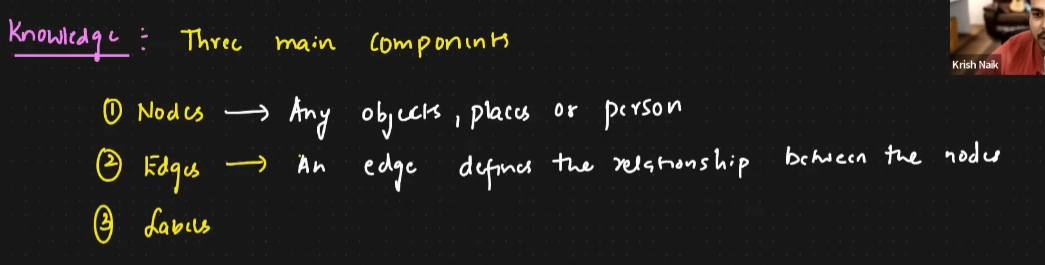
|  |  |
| --- | --- |
| GEN AI LEARNING  **Complete Generative AI Learning – New Year Challenge** | WRITTEN BY: ALOY |

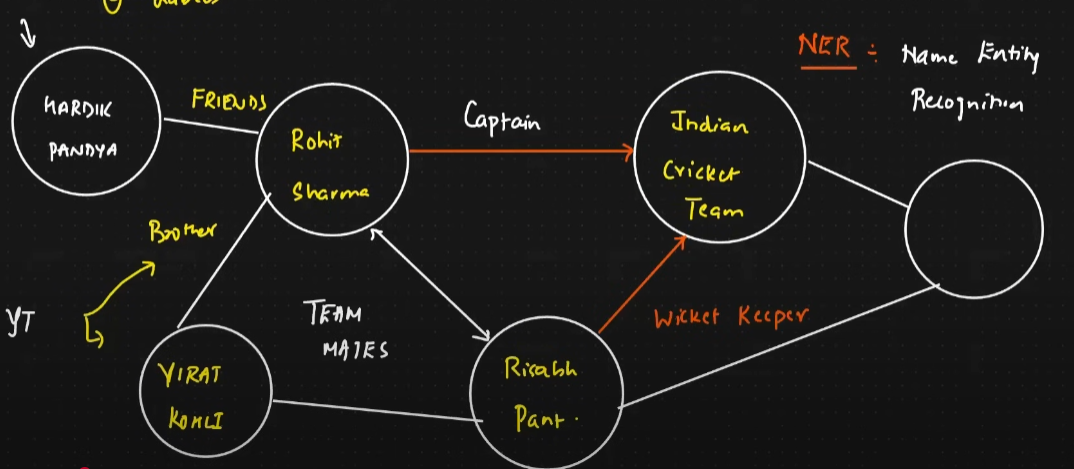
***Day – 15***

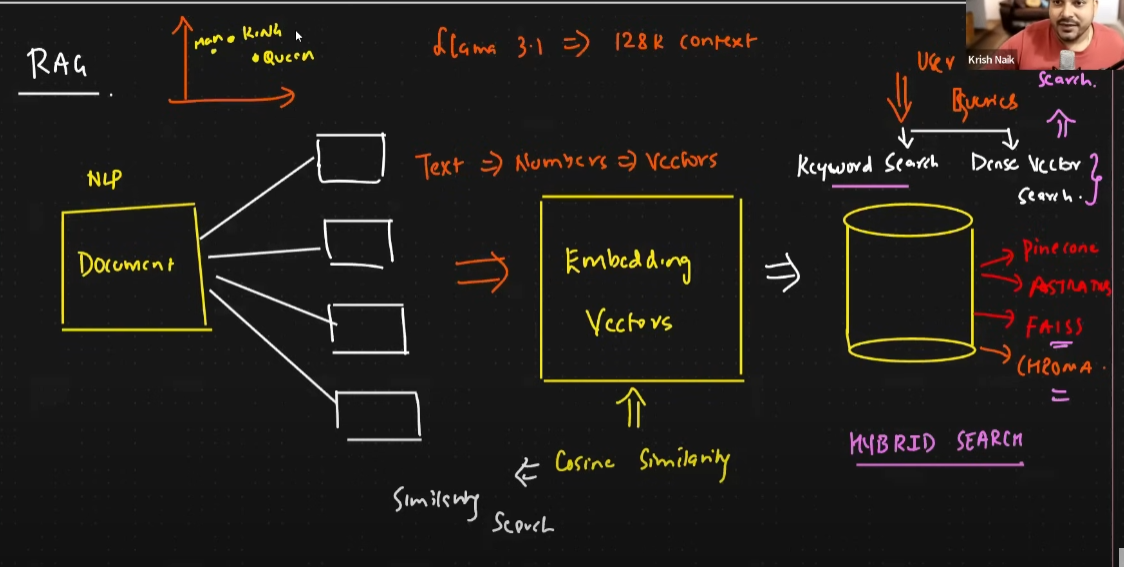
Knowledge Graph:

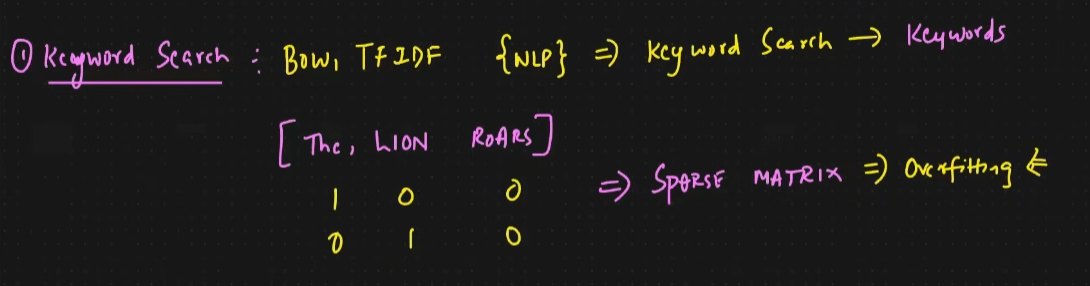


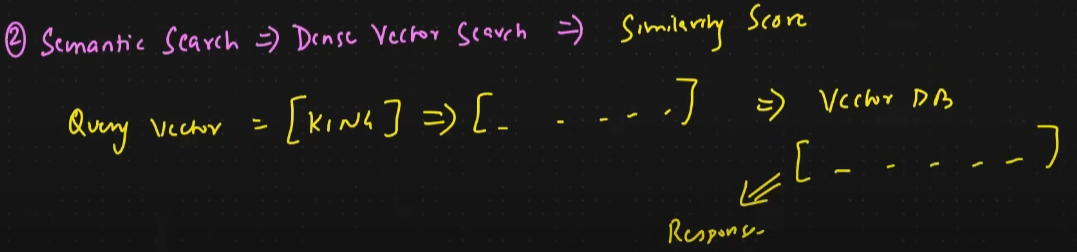


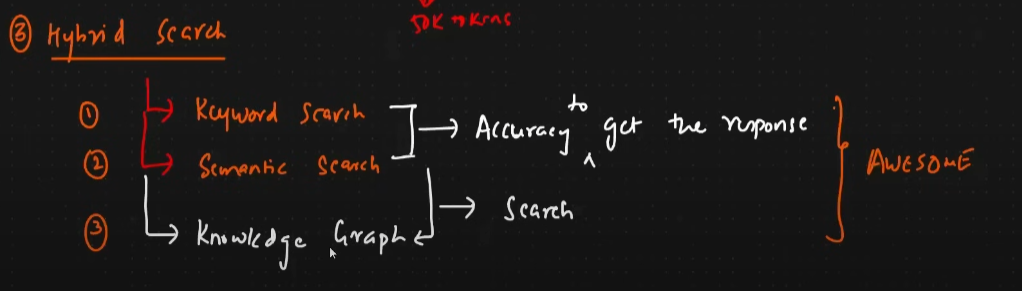












1. **Agenda**:
   * The session focuses on the following topics:
     1. Knowledge Graphs
     2. Neo4j Graph Database
     3. Cypher Query Language
     4. GraphDB with LangChain
2. **Introduction to Knowledge Graphs**:
   * **Definition**: A semantic network that represents real-world entities and their relationships.
   * **Purpose**: To illustrate relationships between events, situations, and concepts.
   * **Examples**:
     1. "Rohit Sharma is the captain of the Indian Cricket Team."
     2. "Rishabh Pant is the keeper of the Indian Cricket Team."
3. **Main Components of Knowledge Graphs**:
   * **Nodes**: Represent objects, places, or persons.
   * **Edges**: Define relationships between nodes.
   * **Labels**: Help categorize nodes and relationships.

The diagrams explain the concept of **Retrieval-Augmented Generation (RAG)** and the search methodologies it incorporates: **Keyword Search**, **Semantic Search**, and **Hybrid Search**.

**Key Concepts:**

1. **Document Processing**:
   * Natural Language Processing (NLP) converts documents into meaningful vectors (numerical representations).
   * Text data is transformed into **embedding vectors**, enabling similarity searches through cosine similarity.
2. **Keyword Search**:
   * Utilizes traditional techniques like Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF).
   * Creates a sparse matrix representation, which can lead to overfitting in high-dimensional spaces.
3. **Semantic Search**:
   * Employs dense vector representations for understanding context and semantics.
   * Queries like "King" generate vectors that are matched against a vector database to find similar entries based on **similarity scores**.
4. **Hybrid Search**:
   * Combines Keyword Search, Semantic Search, and Knowledge Graphs to improve accuracy.
   * Integrates multiple search techniques for a more **robust and precise retrieval** process.
5. **Storage and Infrastructure**:
   * Embedding vectors are stored in vector databases like Pinecone, FAISS, Chroma, etc., to support large-scale, efficient searches.
6. **Application**:
   * Hybrid search leverages the strengths of different techniques to provide optimized, accurate results, making the process highly **efficient and versatile** for various use cases.

This system enables **powerful retrieval** mechanisms combining traditional keyword-based methods with advanced semantic understanding, enhancing knowledge discovery and retrieval performance.

<https://neo4j.com/product/auradb/>

Relational Database Management Systems (RDBMS) and Graph Databases are two different types of databases designed for distinct purposes. Here's a comparison of their characteristics, strengths, and weaknesses:

**Relational Database Management Systems (RDBMS)**

**Characteristics:**

* **Structure**: Data is stored in **tables** with rows and columns.
* **Schema**: Strictly defined schema with relationships enforced using **primary and foreign keys**.
* **Query Language**: Uses SQL (Structured Query Language).
* **Relationships**: Handles relationships indirectly through **joins**.

**Strengths:**

1. **Mature Technology**: Decades of use, making it highly reliable.
2. **ACID Compliance**: Ensures strong **data integrity** and reliability.
3. **Efficient for Structured Data**: Excellent for scenarios with well-defined schema and relationships.
4. **Wide Adoption**: Extensive community support and compatibility with tools like MySQL, PostgreSQL, Oracle DB.

**Weaknesses:**

1. **Complex Queries for Relationships**: Performing **joins** across multiple tables for complex relationships can be slow and cumbersome.
2. **Less Flexible for Evolving Schema**: Schema changes can be costly.
3. **Not Optimized for Highly Connected Data**: Struggles with efficiently handling deep and dynamic relationships.

**Use Cases:**

* Financial systems.
* E-commerce platforms.
* Inventory and order management.
* Traditional enterprise applications.

**Graph Databases**

**Characteristics:**

* **Structure**: Data is stored as **nodes** (entities) and **edges** (relationships between entities).
* **Schema**: Schema is **flexible** and often schema-less.
* **Query Language**: Uses graph-specific query languages like **Cypher** (Neo4j), **Gremlin**, or **SPARQL**.
* **Relationships**: Relationships are first-class citizens, stored as **edges**, making them inherently more efficient to traverse.

**Strengths:**

1. **Optimized for Connected Data**: Excellent for handling deep and complex relationships without joins.
2. **Flexibility**: Can handle unstructured or semi-structured data easily.
3. **Performance for Relationship Queries**: Traversing nodes and edges is fast, even for deeply nested relationships.
4. **Evolving Schema**: Adapts easily to changes in data structure.
5. **Visual Representation**: Natural visualization of data as graphs.

**Weaknesses:**

1. **Lack of Standardization**: Fewer standardized tools and languages compared to RDBMS.
2. **Smaller Ecosystem**: Less mature than RDBMS, with fewer experts and tools.
3. **Not Ideal for Tabular Data**: Inefficient for scenarios requiring strict tabular data processing.
4. **Learning Curve**: Requires specialized knowledge of graph databases and query languages.

**Use Cases:**

* Social networks (e.g., user connections).
* Fraud detection (e.g., transaction patterns).
* Recommendation engines.
* Knowledge graphs and semantic queries.
* Network topology analysis.

**Comparison Table**

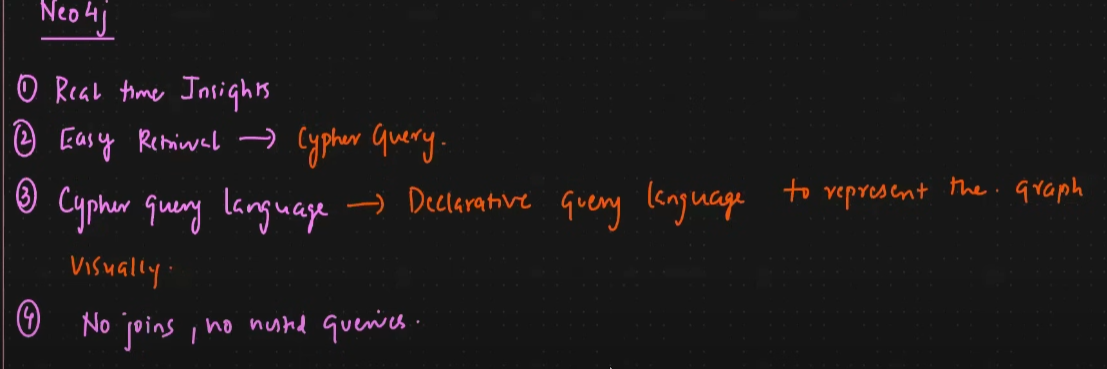
|  |  |  |
| --- | --- | --- |
| Feature | RDBMS | Graph Database |
| Data Structure | Tables (rows and columns) | Graph (nodes and edges) |
| Schema | Fixed, predefined schema | Flexible, schema-less |
| Query Language | SQL | Cypher, Gremlin, SPARQL |
| Relationship Handling | Indirect (joins) | Direct (edges) |
| Performance for Relationships | Slower for complex joins | Optimized for relationships |
| Flexibility | Low | High |
| Best Use Case | Structured data, transaction systems | Highly connected data |

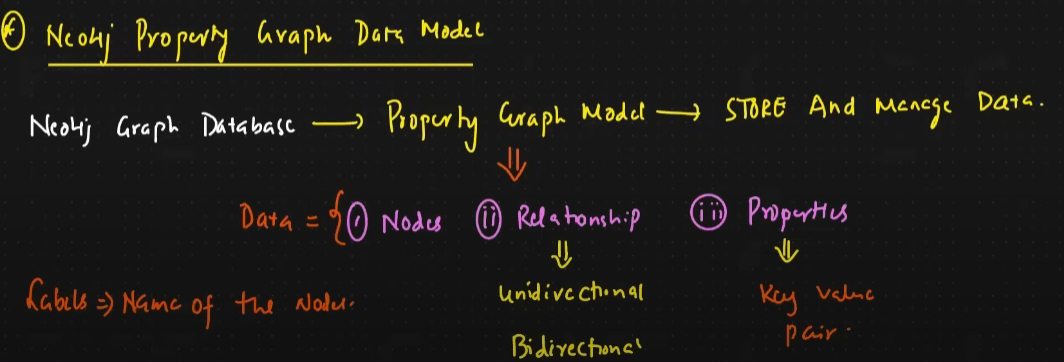
**When to Use Which?**

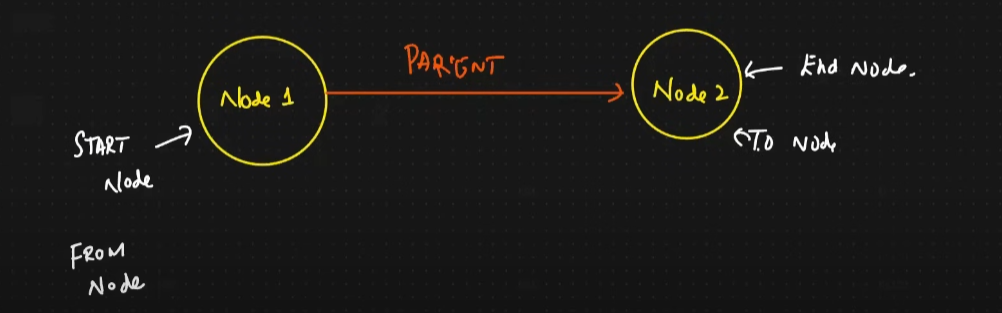
* **RDBMS**:
  + When your data is structured and fits a tabular format.
  + For applications requiring strict ACID compliance.
  + If relationships are simple or shallow.
* **Graph Databases**:
  + When your data is highly connected (e.g., social networks, fraud detection).
  + If relationships are dynamic, deep, or the schema evolves frequently.
  + For use cases requiring fast traversal of relationships.

Choosing between RDBMS and Graph DB depends on the nature of your data and the complexity of relationships you need to handle. In many modern systems, they are used together to leverage their respective strengths.

**Neo4j:**







The notes focus on **Neo4j**, a popular graph database, emphasizing its features, data model, and structure.

**Key Highlights of Neo4j**

1. **Real-time Insights**:
   * Neo4j provides **real-time data retrieval** and insights, making it efficient for connected data queries.
2. **Easy Retrieval with Cypher Query Language**:
   * Cypher, Neo4j's declarative query language, is intuitive and allows easy representation of graph structures visually.
3. **No Joins, No Nested Queries**:
   * Unlike traditional RDBMS, Neo4j eliminates the need for complex joins, making it faster and more efficient for relationship-heavy datasets.

**Neo4j Property Graph Data Model**

1. **Core Components**:
   * **Nodes**: Represent entities (e.g., "Person", "Location").
   * **Relationships**: Represent connections between nodes, which can be **unidirectional** or **bidirectional**.
   * **Properties**: Store additional data as **key-value pairs** for nodes and relationships.
2. **Representation**:
   * Nodes are connected by relationships, and properties provide additional context (e.g., a "Parent" relationship connecting Node1 and Node2).
3. **Start and End Nodes**:
   * Each relationship connects a **start node** (source) to an **end node** (destination), enabling efficient traversal.

**Use Cases of Neo4j:**

* Ideal for **knowledge graphs**, **social networks**, and **hierarchical relationships**.
* Handles highly connected and dynamic data efficiently without performance degradation.

In summary, Neo4j's graph-based model and Cypher language make it powerful for analyzing and querying complex, interconnected data structures.

