Introduction to Graph Database in System Design

Introduction

In various real-world scenarios, we need to understand relationships between data elements rather than individual data elements. To understand such relationships, graph databases provide an intuitive and efficient way to organize and analyze such data to get valuable insights.

- Graph databases help us to efficiently traverse complex hierarchies, identify hidden connections, and uncover inter-relationships between elements. That’s why they are used in various applications, where things are interconnected.
- In graph databases, we organize information into nodes to represent data entities, and edges to indicate the relationships between them. Edges can denote various types of relationships such as parent-child, ownership, and actions. There are no restrictions on the number and types of relationships that can be associated with a node.

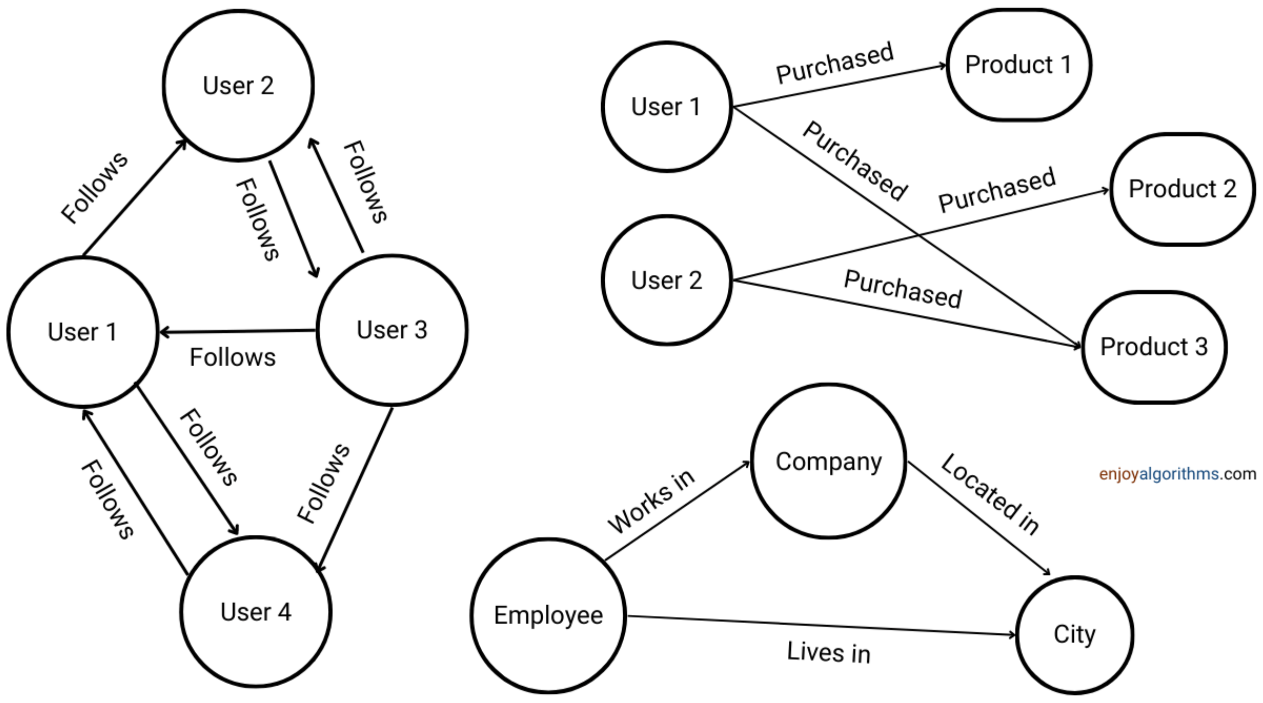
In system design, a graph database is classified as a NoSQL database because it does not use the tabular structure of a traditional SQL database. Instead, it stores data in a graph-like structure.

Graph database use cases in real-life applications

- In recommendation engines, a graph database is useful for storing the relationships between different categories of information, such as customer interests, friends, purchase history, and preferences. These systems analyze user behaviour and offer personalized product recommendations.
- To maintain secure transactions, fraud detection systems analyze the connections between individuals and their purchase activities. For example, these systems can detect patterns of relationships, like

multiple individuals linked to a single personal email address or various people sharing an IP address despite living in different locations.

- Using social media, we can easily find people (nodes) and their relationships (edges). For example, we can find out who the "friends of friends" of a particular person are.
- In life sciences, graph databases help us understand the complex relationships between genes, proteins, and diseases. This can lead to advancements in drug discovery and personalized medicine.
- Knowledge graphs help us manage complex relationships between entities such as people, places, and events. This can improve semantic search and information retrieval.



Why do we prefer graph databases over relational databases?

Let's take an example of social media. If we use traditional relational databases, we need to create a “users” table that contains data about each user (name, email, password, etc.). We also need to create a “connections” table to store data about relationships between users (date they connected and type of relationship like friend, family, etc.).

In such a scenario, we might need to perform a JOIN operation between the "users" and "connections" tables to analyze relationships. This can be time-consuming and resource-intensive if there are large amounts of data and relationships.

For example, if we want to find all friends of a user, we might need to perform a JOIN operation that matches the user's ID with the IDs of all their friends in the "connections" table. This operation can be slow with the increase in the number of relationships and users, which could lead to long wait times and poor user experience.

On another side, the rigid schema of traditional relational databases can limit the flexibility and scalability of the database. For example, if we want to add a new type of relationship, we might need to modify the schema of the "connections" table. This can be a complex and time-consuming process. In other words, this can make it difficult for us to adapt to changing requirements and increase latency.

To solve the above issues, graph databases offer a better solution. Graph databases provide a flexible way of storing relationships between data elements.

- Graph databases eliminate the need for time-consuming JOIN operations or cross-lookups.
- In a graph database, moving through the connections or links between nodes is done quickly because the relationships are stored in the database, instead of being calculated each time a query is made.
- Graph databases have a flexible schema, which helps us to easily add new types of relationships without modifying the schema. This makes it easier to adapt to changing requirements.

For example, if we want to find all friends of a user on social media, we can simply start from the node and follow edges (friend relationship) to reach all the friends. This will be much more efficient process than performing a JOIN operation in a relational database.

Some popular graph databases

- **Neo4j** is a highly efficient and scalable graph database with powerful performance.
- **Amazon Neptune** is a fully managed graph database service offered by Amazon Web Services.
- **ArangoDB** is an open-source multi-model database.
- **TigerGraph** is a fast and scalable graph database that is specifically designed for use cases such as real-time fraud detection, recommendation systems, and network analysis.
- **RedisGraph** is an open-source graph database built on top of popular in-memory data store, Redis.
- **GraphQL** is a powerful query language for APIs, which provides a complete and understandable description of the data and gives clients the power to ask for exactly what they need.

Advantages of graph database

- Provide a flexible data model that simplifies the representation of complex relationships.
- Process and analyze large amounts of data in real time (useful for real-time applications).
- Offer fast performance for complex queries (useful for fast and efficient data analysis).
- Easily integrate with other data sources (useful for analyzing data from multiple sources).
- Handle dynamic and changing data (useful for applications where data is constantly changing).

How graph database is implemented under the hood?

At a high level, a graph database is composed of a collection of nodes and edges, which are stored in a data structure such as an adjacency list or matrix. The nodes represent entities in the data, while the edges represent relationships between the entities. But in a typical implementation at low level:

- Graph database will use an index to efficiently store and retrieve the nodes and edges in the graph. For example, a hash table or B-tree can be used to index nodes based on some unique identifier. When performing a query, database will use indexes to quickly locate relevant nodes, and then traverse the edges to find related nodes.
- In addition to indexing, graph databases may also use various optimization techniques to improve performance, such as caching frequently-used data, using algorithms such as breadth-first or depth-first search to traverse the graph, or partitioning the graph into smaller sub-graphs to reduce the size of the data that needs to be processed.

Overall, implementation of a graph database will depend on the specific requirements of the application, such as size and complexity of data, type of queries that need to be performed, and performance and scalability requirements.

Some common queries on graph database

- **Neighbourhood queries:** Used to find all nodes and edges that are directly connected to a specific node. For example, finding all friends of a user on a social network.
- **Pathfinding queries:** Used to find the shortest path between two nodes in the graph. For example, finding the shortest path between two cities in a transportation network.
- **Pattern matching queries:** Used to find all instances of a specific pattern in the graph. For example, finding all triangles in a social network (three users who are friends with each other).
- **Centrality queries:** Used to find the most important nodes in the graph based on some measure of centrality. For example, finding the most influential users in a social network based on the number of connections they have.
- **Clustering queries:** Used to find groups of nodes that are densely

connected to each other, but less connected to other nodes in the graph. For example, finding clusters of users who are friends with each other but not friends with users outside the cluster.

These are just a few examples of the types of queries that can be performed on a graph database. The specific queries will depend on the nature of the data and the requirements of the application.

If you have any queries/doubts/feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design!