Graph Databases For Beginners

Ch1: Graphs are the Future

Graph databases can handle increasing volumes of data and complex relationships between data points with consistent performance. Additionally, the structure and schema of a graph database can be flexibly adapted to changing business needs without affecting existing functionality. Finally, graph databases align with agile development practices and allow for evolutionary changes to applications.

Graph databases are intuitive and simpler than relational database management systems. A graph consists of nodes that represent entities and relationships that represent how two nodes are associated. For example, "cake" and "dessert" would have the relationship "is a type of" pointing from "cake" to "dessert."

Relationships are given priority in graph databases, unlike other database management systems. This simplifies data models and makes them more expressive. There are two important properties of graph database technologies to understand: graph storage and graph processing engine. Some graph databases use "native" graph storage, while others use relational or object-oriented databases, which can be slower. Native graph processing is the most efficient means of processing data in a graph, and Neo4j is currently the industry leader in both graph storage and processing.

The interconnected nature of the real world can be mimicked by graph databases, making them useful for understanding big datasets in a variety of scenarios, such as logistics route optimization, retail suggestion engines, fraud detection, and social network monitoring. Graph databases are becoming increasingly popular, while big data is also growing. By harnessing the power of graph technology, businesses can gain a competitive advantage over their competitors who may not be using graph databases. However, the window of opportunity to gain this advantage is narrow as competitors may soon learn to use graphs as well.

Ch2: Why Data Relationships Matter

Relational databases (RDBMS) were designed to codify paper forms and tabular structures, which they still do well. However, they are not effective at handling data relationships, especially when those relationships are added or adjusted on an ad hoc basis. The inflexibility of their schema is their greatest weakness. As business needs are constantly changing and evolving, the schema of a relational database cannot efficiently keep up with those dynamic variables. To compensate, development teams may try to leave certain columns empty, but this approach requires more code to handle exceptions in the data, which can lead to performance disruptions and hinder further development as the data multiplies in complexity and diversity. Relational databases require developers to write several JOIN tables to discover what products a customer bought, which can significantly slow down the application's performance. Moreover, asking reciprocal questions like "Which customers bought this product?" or "Which customers buying this product also bought that product?" becomes prohibitively expensive. However, such questions are essential to building a proper recommendation engine for transactional applications. As a business's needs grow, their database schema may become inadequate. However, migrating data to a new schema is incredibly effort-intensive.

NoSQL databases store sets of disconnected documents, values, and columns, which gives them a performance advantage over relational databases. However, this disconnected construction makes it harder to harness data relationships properly. Some developers add data relationships to NoSQL databases by embedding aggregate identifying information inside the field of another aggregate, using foreign keys. But this approach becomes prohibitively expensive later when joining aggregates at the application level. Foreign keys only point in one direction, making reciprocal queries too time-consuming to run. In contrast, graph databases store data relationships as relationships, making them more efficient and flexible when it comes to query speeds, even for deep and complex queries. The explicit storage of relationship data in graph databases means fewer disconnects between evolving schema and the actual database. Adding new nodes and relationships in graph databases is easier and does not compromise the existing network or require expensive data migration.

Ch3: Data Modeling Basics

Data modeling is an abstraction process that starts with business and user needs and maps them into a structure for storing and organizing data. With traditional database management systems, modeling is far from simple. After whiteboarding initial ideas, relational databases require the creation of a logical model, which is then forced into a tabular, physical model. By the time a working database is created, it may look nothing like the original sketch, making it difficult to tell if it meets user needs. In contrast, modeling data for a graph database is simpler. A structure of circles and boxes connected by arrows and lines is already a graph. Creating a graph database from there is just a matter of running a few lines of code.

Data modeling is not a one-off activity that is completed once at the beginning of application development. Instead, over the lifetime of an application, the data model constantly shifts and evolves to meet changing business and user needs. Relational databases, with their rigid schemas and complex modeling process, are not well-suited for rapid change. What is needed is a data modeling approach that does not sacrifice performance and supports ongoing evolution while maintaining data integrity. The agile approach offered by a graph database can create data models more quickly and adapt them to the changing needs of an uncertain future.

Ch4: Data Modeling Pitfalls to Avoid

Graph databases are highly expressive when it comes to data modeling for complex problems, however, expressivity does not guarantee that a data model will be correct on the first try. Even experts make mistakes.