Rasmus

What are the advantages and disadvantages of using graph databases and which are the best and worse scenarios for it?

Advantages:

Graph databases offer great performance when it handles data which focuses on relationships between entities. This means that it would be a great option for social media systems. Furthermore, it ensures great performance even when the data grows, which means that it is a good option for analysing big data. Graph databases makes it very easy to just add new nodes, properties, and relationships without any concerns of changing a schema.

Because graph databases are structured as graphs, it makes it easier to use algorithms that find patterns in the data, which may be important.

Disadvantages:

Graph databases have not been around for as long as relational databases, which means there may not be as much support, or it may be harder to find help you are looking for. Graph databases are also not optimized for performing transactions which makes them inefficient in this department. Lastly, the query language various between the different providers of graph databases which can be confusing since it isn't just 1 standardized query language.

This means that it wouldn't be preferrable to use graph databases when the system you want to develop revolves around performing transactions and not as many relationships and analysis of these.

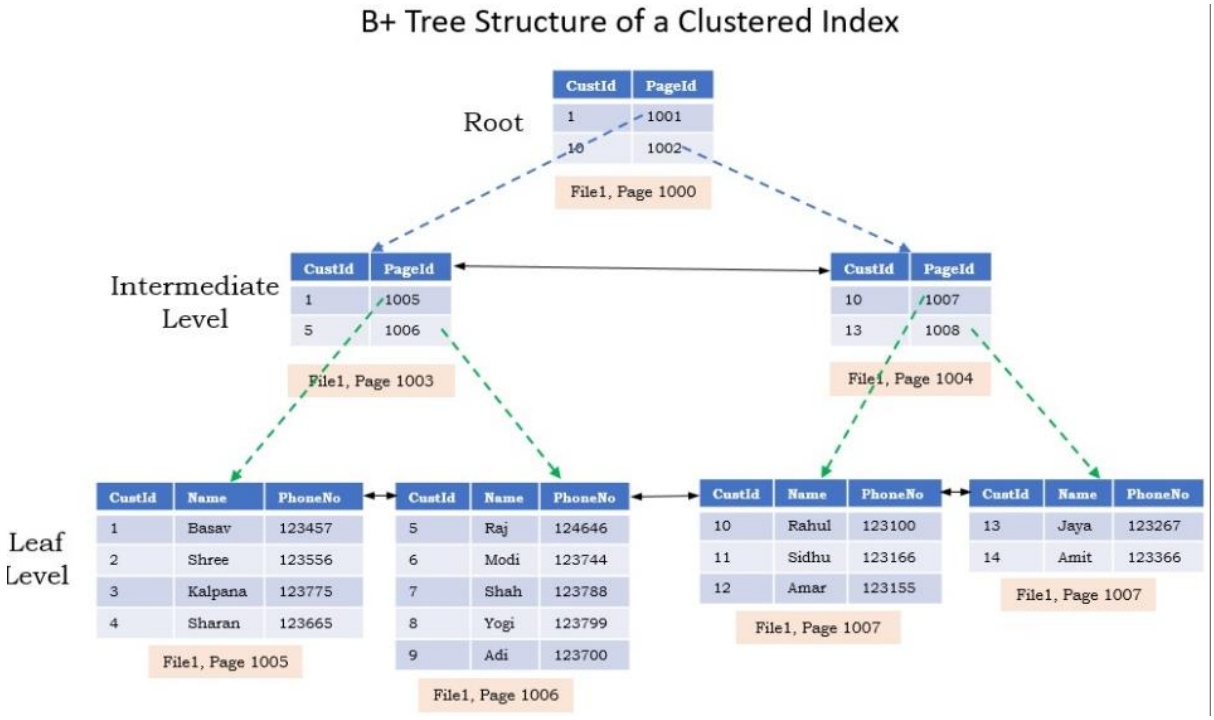
How would you code in SQL the Cypher statements you developed for your graph- algorithms-based query, if the same data was stored in a relational database?

With the current knowledge that I have of graph databases and relational databases, I would not be able to recreate SQL equivalent to the cypher statements used for the graph-algorithm-based query. I do know that I would probably have to represent nodes and edges from the graph database as tables in the relational database.

To then further perform a graph-algorithm on this would probably require a lot of conditional statements.

How does the DBMS you work with organizes the data storage and the execution of the queries?

If we look at SQL Server, we know that data is stored in rows and columns, but physically it is stored in data pages. A data page is a unit of data which has a size of 8KB. Any data inserted into a table will be saved to a series of data pages. The data pages are structured as a B-tree.



On the Leaf Level we have the actual data pages which contain the data. The number of rows that are stored in each data page depends on the size of the row. These data pages are sorted by the primary and clustered key CustId. The Root and intermediate Level have index nodes, which pint further down the tree so that there is a path all the way down to the Leaf Level. This means that the entire B-tree consists of pointers, so that the database engine will be able to look the data up efficiently.

Which methods for scaling and clustering of databases you are familiar with so far?

When it comes to scaling there are in simple terms two ways of doing so: Vertical Scaling and Horizontal Scaling.

Vertical scaling is when you improve/add more resources to an existing machine/server.

Horizontal scaling is when you add more machines/servers which are then able to share the load.

Clustering can also be a way to scale because the request from clients can be spread out between the database nodes. Each node in a cluster contains a copy of the data from primary node. Only the primary node can be written to.

Another way of clustering/scaling is sharding which is the action of separating databases into smaller parts called “shards”. This makes it easier to manage. Essentially, sharding distributes data from a database between multiple nodes in a cluster. Each shard is only responsible for the data that it stores which increases the processing power. However, this means that when the database handles a request, it must discover which node contains the relevant data.