WHITE PAPER

The Graph Technology Buyer's Guide

What You Should Know before Selecting a Graph Technology Solution

Amit Chaudhry, Vice President, Neo4j

White Paper

TABLE OF CONTENTS

The Graph Technology Buyer's Guide

What You Should Know before Selecting a Graph Technology Solution

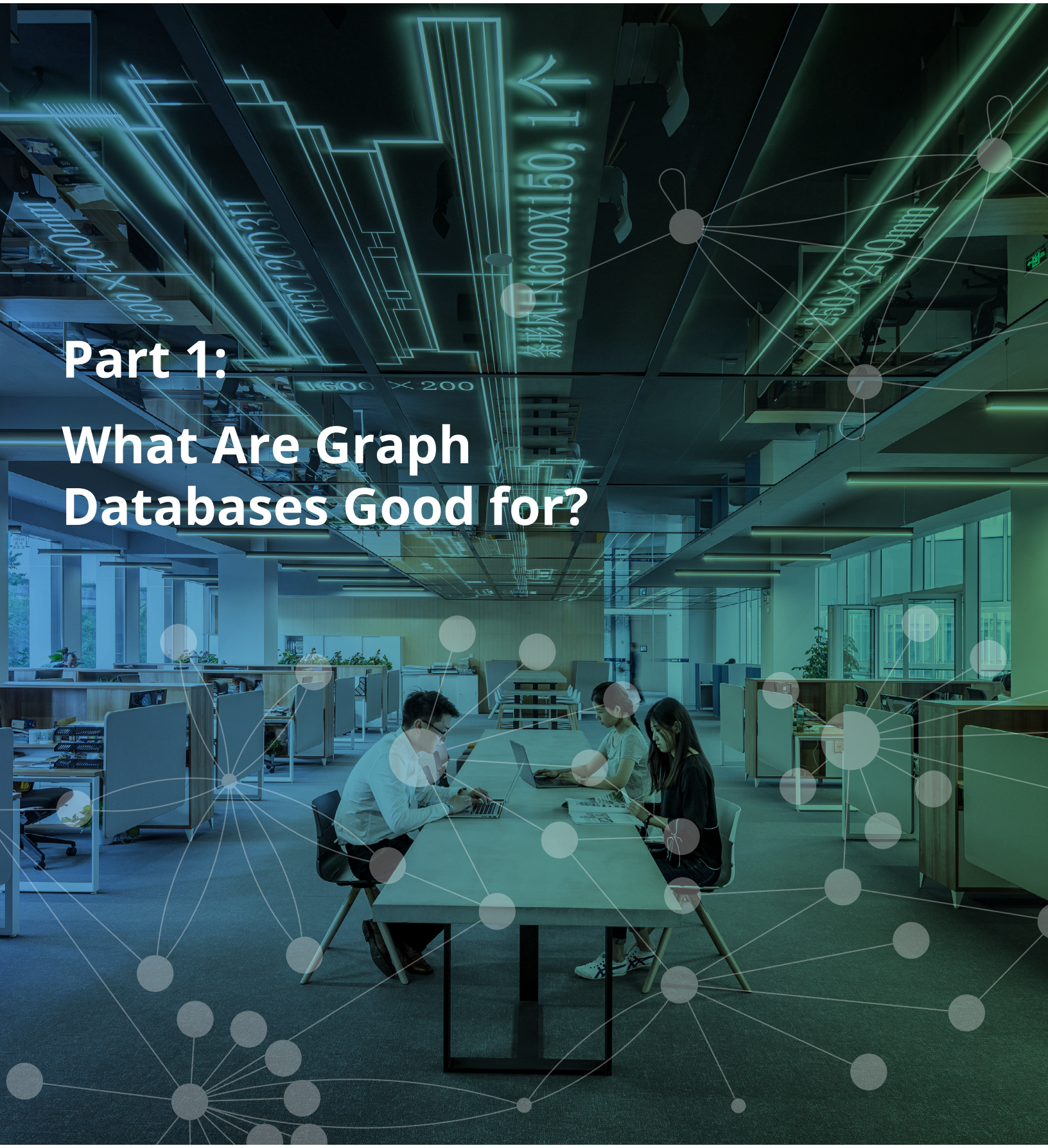
Amit Chaudhry, Vice President, Neo4j

Introduction

The graph technology market is heating up with a wide variety of database and analytics vendors staking claim to supporting graph (connected data) capabilities.

The challenge for the modern technology buyer is weeding through mountains of material in order to make an informed buying and implementation decision. This challenge is compounded because graph technology is **a new way of managing data**. Data consistency, performance, scalability and other important characteristics of any database technology are all much different with graph databases than with traditional data management platforms like relational databases (RDBMS).

This buyer's guide is designed to help expedite those decisions and explain what makes purchasing this type of technology so different from other database purchase decisions. For the full list of buying criteria and questions, skip to Part 2 on page XX. For readers who aren't yet familiar with the basics of graph technology, we'll start with a brief introduction.



Part 1:

What Are Graph Databases Good for?

Viewed through a technological lens, graph databases tackle the most harrowing of data problems – ones that often linger at the root of project failures and delays.

What Are Graph Databases Good for?

Graph database technology is specifically designed and optimized for highly interconnected datasets to identify patterns and hidden connections.

Graph data stores are intuitive because they mirror the way the human brain thinks and maps associations via neurons (nodes) and synapses (relationships). A graph database efficiently stores and queries connected data in a node-and-relationships format. As a result, graph technology excels at problems where there is no a priori knowledge of path length or shape by using graph storage and infrastructure to find neighboring data efficiently.

The most common graph use cases and solutions include:

- **Fraud Detection & Analytics:** Real-time analysis of data relationships is essential to uncovering fraud rings and other sophisticated scams.
- **Artificial Intelligence & Machine Learning:** Artificial intelligence (AI) winners and losers will be decided based on who harnesses context within data for a true competitive advantage.
- **Real-Time Recommendation Engines:** Graph-powered recommendation engines help companies personalize products, content and services by building a contextualized map of offers using both historical and real-time data.
- **Knowledge Graphs:** Graph-based search tools tap into your organization's institutional memory. They are also used for better digital asset management. Moreover, knowledge graphs are the basis for many natural language processing (NLP) and AI solutions.
- **Network & Database Infrastructure Monitoring:** Graph databases are inherently more suitable than RDBMS for making sense of complex interdependencies central to managing networks, data centers, cybersecurity and IT infrastructure.
- **Master Data Management (MDM):** The schema-optional graph database model allows you to organize and manage your master data with flexibility. It also lets you harness real-time insights and a 360° view of your customers, products and employees.

What these use cases have in common is that their success requires solving complex problems with dynamic and interconnected datasets. To this end, we should reframe the question, "What are graph databases good for?" as a technical one.

Viewed through a technological lens, graph databases tackle the most harrowing of data problems – ones that often linger at the root of project failures and delays. These include:

- Vastly different views of the data model between business and technology teams, which result in misunderstanding and miscommunication.
- Lack of schema flexibility and adaptability, making it hard to respond to changing business requirements both during a project and after a system has gone live.
- The "JOIN problem" which occurs when queries become so tangled that even powerful databases with massive amounts of hardware resources grind to a halt in their attempts to bring the resulting data together.

With the right choice of technologies, graph databases introduce a new way of looking at data by promising to significantly address all of these issues. They seek to take you beyond

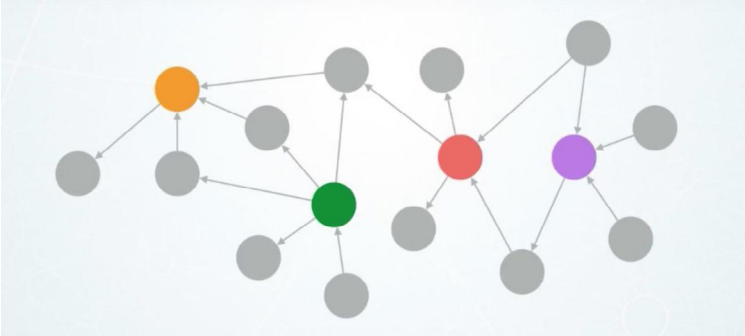
Well-established entities such as NASA, eBay, UBS and many of their peers use graph technology to improve their customer experiences and increase their competitiveness.

handling sheer volumes of relatively simple data and towards revealing the interconnected complexities latent in your data – and deriving bottom-line value from them.

Traditional Technology Choices Do Not Consider How Data Is Interrelated

Today's data and applications require elasticity, agility, speed and interconnectivity. Despite the name, relational databases are not well-suited for modeling and storing today's highly connected and agile datasets. RDBMS demand slow and expensive schema redesigns that hurt agile software development processes and hinder your ability to scale and innovate quickly.

Traditional RDBMS technology has a difficult time expressing and revealing how real and virtual entities are related. Columns and rows aren't how data exists in the real world. Rather, data exists as rich objects and the relationships between those different objects.



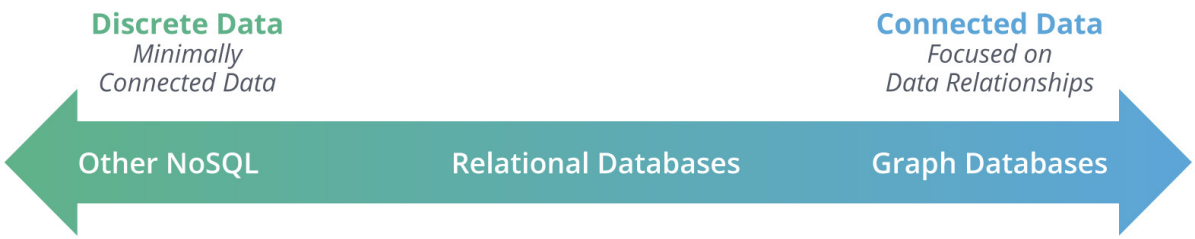
The theory of connected networks (graph theory) has been a mathematical discipline for nearly three centuries, but few technologies have harnessed these theoretical models for the purpose of data storage and analysis.

Early on, a handful of companies have truly tapped into the power of graph technology as the driver of their businesses, including Google, Facebook, LinkedIn and Microsoft as well as upstarts like Airbnb and Uber. Well-established entities such as NASA, eBay, UBS and many of their peers use graph technology to improve their customer experiences and increase their competitiveness.

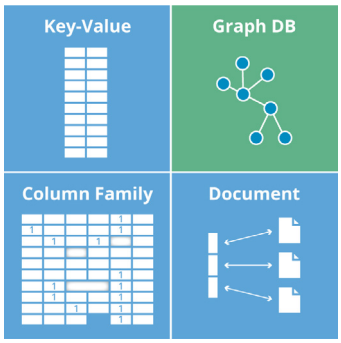
Today, graph-powered applications are used by more than 75% of the Fortune 500, including:

- 7 of the world's top 10 retailers
- 3 of the top 5 aircraft manufacturers
- 8 of the top 10 insurance companies
- All of North America's top 20 banks
- 8 of the top 10 automakers
- 3 of the world's top 5 hotels
- 7 of the top 10 telcos

These successes serve as a strong indicator of graph technology's impact on both innovation and the bottom line.



By eschewing data relationships and providing simple programmatic APIs, NoSQL systems make it easy for developers and administrators to work with simple data in a way that can easily scale.



Collections vs. Connections

SQL & NoSQL Systems Focus on Data Aggregation & Collection

Collection-centric storage designs as implemented by SQL and Not only SQL (NoSQL) databases are designed to **efficiently divide and store data.**

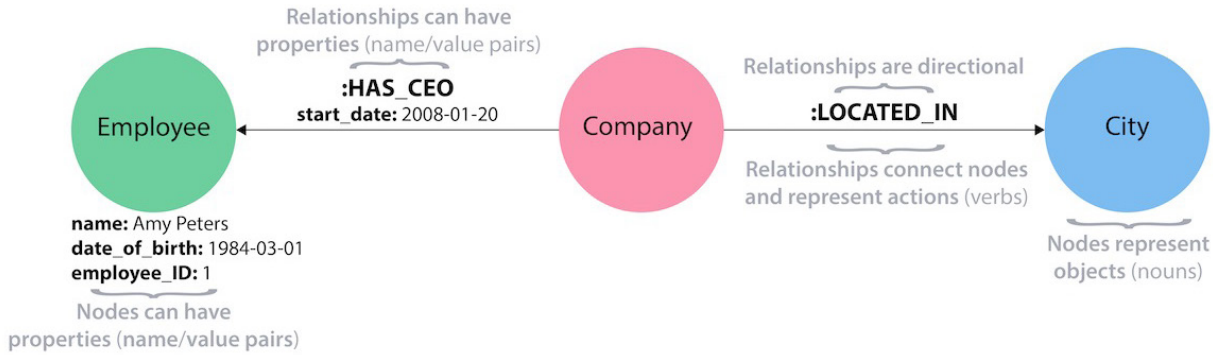
In SQL's case, the normalization of data into a tabular schema aims to minimize storage of duplicate data objects, types and values. These systems were born during the era of scarce physical memory and expensive disk-based storage, designed to avoid managing often-redundant data objects such as physical location addresses for shipping, billing, homes, offices, destinations, businesses, etc. For example, all of these data objects included common, redundant data such as a country and its provinces or states, or telephone area codes and postal codes.

The original relational databases were designed to minimize storage of duplicative data values because disk space was costly. (Ironically, each normalization incurs a cost in relationship storage in the form of JOIN tables, which native graph databases have managed to eliminate through the use of pointers.) The RDBMS design achieved this consolidated, normalized goal by linking tables of data via foreign keys to their associated records from other tables. This is why a relational dataset modeled into a graph often shrinks by an order of magnitude or more, maintaining the full richness of the data without redundant data storage.

NoSQL systems like document, wide column and key-value data stores carry those concepts forward (and backward) by simplifying their models in exchange for higher levels of scale and simplicity. By eschewing data relationships and providing simple programmatic APIs, NoSQL systems make it easy for developers and administrators to work with simple data in a way that can easily scale.

A lack of concern about relationships leads to looser data guarantees, plain APIs and straightforward scaling schemes. Data is easily spread out and just as easily retrieved, without the need to maintain the integrity of related data that's written across a distributed storage or a cluster of machines and without needing to concern itself with the performance of distributed JOINs across those machines.

These systems take on the "store and retrieve" problem at scale for simple data, and their architectures reflect this, as does the set of problems they are equipped to address. However, none of these systems focus on interrelated, contextualized data or how that data might be traversed to reveal unobvious relationships, as explained below.



A property graph is a data model designed to express data connectedness as nodes connected via relationships to other nodes, where both nodes and relationships can have properties attached to them (which in turn can be indexed).

Graph Systems Focus on Data Connections

By contrast, graph database technologies focus on how data elements are interrelated and contextualized as connected data.

Connected data is the materialization and harnessing of relationships between data elements, which is modeled as a property graph.

A property graph is a data model designed to express data connectedness as nodes connected via relationships to other nodes, where both nodes and relationships can have properties attached to them (which in turn can be indexed). For example, devices on an enterprise network might be modeled as nodes, with properties for their attributes (such as throughput) and relationships pointing to other adjacent devices.

In the graph model, **data relationships are persisted** so they can be navigated or traversed along connected paths to gain context. Relationships are both typed and directional.

The context provided by these data connections is essential to identifying friendships, making relevant real-time recommendations, attaching adjacent ideas and detecting fraud by following money trails. Without relationships as first-class data entities, all of these use cases become extremely difficult to execute.

Property Graphs Are Intentionally Simple

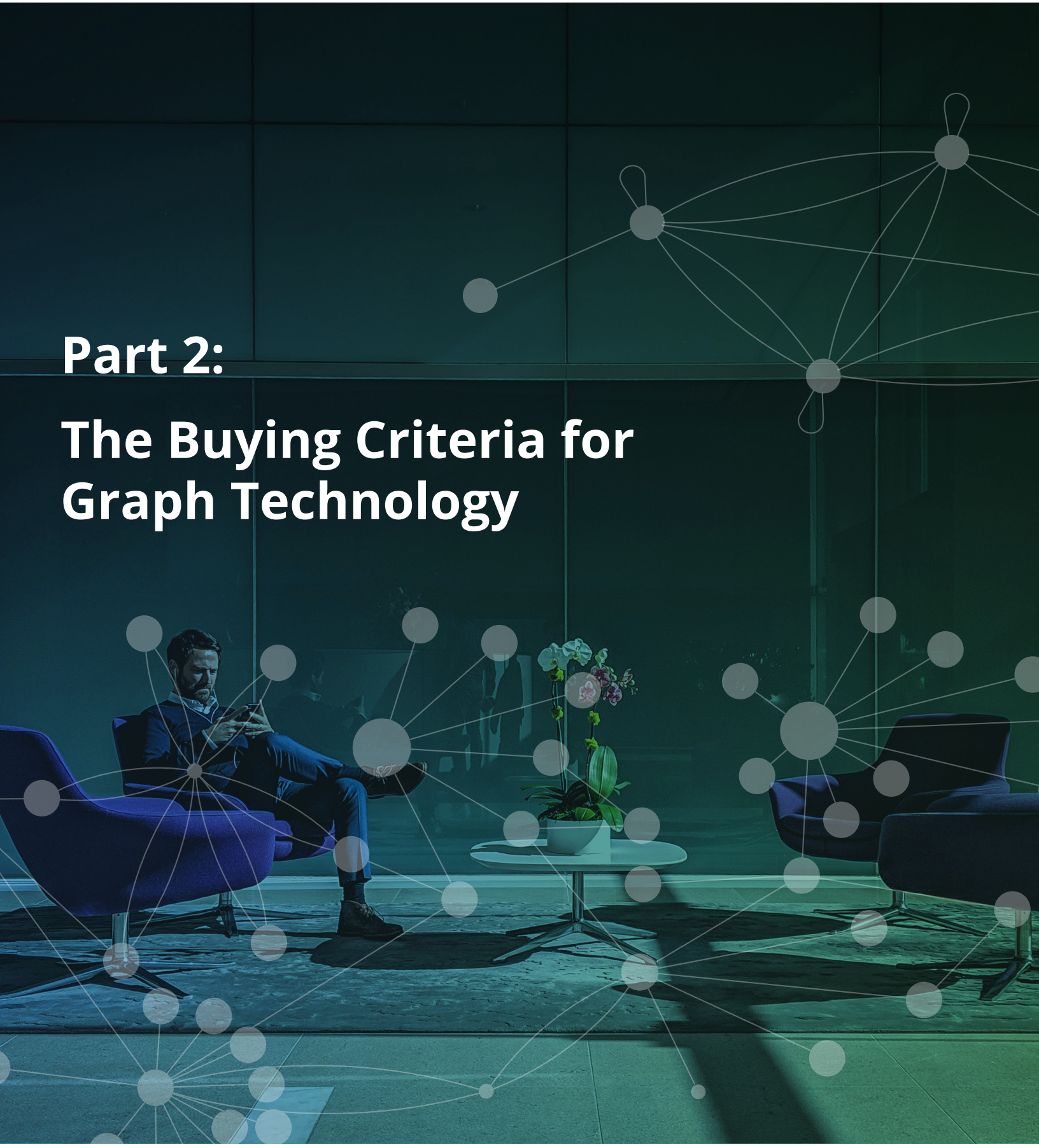
- You can draw property graphs on whiteboards and map that design directly into a graph database.
- You can change or update a property graph easily, because its agile design eliminates most of the structural overhead of traditional database schemas.
- You can quickly program property graphs because their query language expresses and follows relationships.
- You can visualize and navigate property graphs efficiently by following the relationships on their paths to context.
- You can rapidly determine data context when property graph queries are executed in hyper-fast native graph platforms built on reliable, scalable database architectures.

Connected data in property graphs enables you to illustrate and traverse many relationships and find context for your next breakthrough application or analysis.

Benefits of Graph Databases

- **Simple and natural data modeling:** Graph databases provide flexibility for data modeling, depending on relationship types. Since the graph model comes with no inherent rules, graph data stores add as much or as little semantic meaning as the domain requires. This occurs without any constraints like normalization or restructuring of the data using denormalization.
- **Flexibility for evolving data structures:** Graph technology provides flexible schema evolution. In a constantly changing data environment, you need the option to add or drop data entities or relationships as well as extend or modify your data model. Graph databases allow for evolving data structures that match today's agile development environments.
- **Simultaneous support for real-time updates and queries:** A graph data store and its model allow real-time updates on graph data while supporting queries concurrently.
- **Better, faster and more powerful querying and analytics:** Graph data stores provide superior query performance with connected data using native storage and native indexed data structure.

Connected data in property graphs enables you to illustrate and traverse many relationships and find context for your next breakthrough application or analysis.



Part 2:

The Buying Criteria for Graph Technology

Open source technology helps drive adoption, ensure quality and assist market fit as the software matures version by version with the feedback – and sometimes contribution – of a global community.

Structure

The structure of this document allows the reader to understand the buying criteria to consider, given their desired use case. Each vendor you consider should be able to robustly answer your questions according to these criteria.

Graph DBMS buying criteria is often mischaracterized as being akin to purchasing any traditional database. To some degree that is true, with shared questions such as:

- ❒ Does it store data safely and redundantly?
- ❒ Does it have security features?
- ❒ Does it have an easy-to-use query language?
- ❒ Does it scale?
- ❒ Is it embeddable?
- ❒ Does it run in the cloud?
- ❒ Does it work with my existing applications?

All of these questions are indeed valid. There are, however, important requirements related to supporting and persisting data relationships that make graph database management systems and platforms unique, therefore requiring specific buying criteria. These criteria are explained below.

Open Source Foundation & Community

The most successful infrastructure software contains a strong foundation of open source code that provides power and transparency, often catalyzed by a community of users around that software.

Open source technology helps drive adoption, ensure quality and assist market fit as the software matures version by version with the feedback – and sometimes contribution – of a global community. Beyond Oracle, Microsoft SQL Server and a handful of other early RDBMS, very few proprietary database technologies survive when compared to their open source counterparts like Postgres, MySQL, MariaDB, MongoDB and many others.

The premise of using open source as a means to validate and move a market forward is no different in the case of graph databases. Therefore, buyers should consider the size and enthusiasm of a graph technology's open source community as a proxy for multiple items, including the market impact of that technology, the likely availability of skill sets in the market, and the flexibility of the software to evolve beyond the capabilities and personnel of a single software vendor.

What to Look for: Open Source Community

There are a number of graph technologies that are born from open source projects. The most notable is Neo4j, which was the first property graph database released as open source in 2010. After nearly a decade, Neo4j's community is thriving with millions of downloads and hundreds of thousands of deployments.

Apache TinkerPop is another open source graph project. The TinkerPop graph traversal engine's v1 release was in 2011, and it became an Apache-incubated project in 2015. Given

Long considered the "gold standard" for database transactional reliability, ACID (Atomicity, Consistency, Isolation and Durability) compliance ensures that a DBMS safely and reliably stores transactional data as it enters and is updated within the system – especially as the data and system grow.

its ease of integration, flexible storage options and permissive-use license, it has become a common choice for NoSQL vendors when adding graph interfaces to their products.

There are a number of other proprietary graph vendors as well. However, their ability to establish market share and adoption is severely limited when compared to the Neo4j and TinkerPop open source projects.

Independent sites like DB-Engines prove out this market reality. As of the publication of this guide, Neo4j's score accounts for roughly half of the total graph market popularity score. Meanwhile, TinkerPop-based products make up approximately 40 percent of that score. The remaining 10 percent are divided among over 20 additional vendors.

Native Graph Storage

Native graph storage ensures that relationship information – the entity that connects one data node to another – is persisted as a primary data element.

Without native graph storage, relationship information may be lost, misconnected or abandoned, all of which are unique symptoms of graph data corruption. Many non-native graph solutions allow for (and sometimes create) graph data corruption, which is unacceptable for mission-critical enterprise applications.

What to Look for: Transposition vs. Persisted Relationships

When considering non-native graph technology, a buyer should ask: Is there a graph-to-document or graph-to-column transposition step buried in the software?

Document databases often store relationships as special document collections. Columnar databases may store relationships as column families. Neither is native, thereby increasing complexity and slowing processing times, especially if a graph traversal query requires multiple hops where each hop needs to be individually translated from the underlying system.

What to Look for: Index-Free Adjacency

Native graph storage with index-free adjacency supports high-performance graph traversals across deep graph datasets.

Index-free adjacency avoids expensive, low-level transpositions that are supported by non-native graph storage and retrieval architectures. Index-free adjacency creates fixed-size pointers to and from any node to another; native graph technology is therefore both fast and predictable in its performance costs.

ACID Compliance

Long considered the "gold standard" for database transactional reliability, ACID (Atomicity, Consistency, Isolation and Durability) compliance ensures that a DBMS safely and reliably stores transactional data as it enters and is updated within the system – especially as the data and system grow.

While some (NoSQL) databases eschew ACID and can get away with doing so for certain use cases, the impact of non-ACID compliance with graph datasets is grave. Because graphs are inherently connected, proven risks arise with non-ACID transactions, where parts of a graph transaction successfully commit while other parts fail, leaving data in a corrupted state.

Unlike the ISO/ANSI SQL that has served RDBMS users as the standard query language for 40 years, there is not yet an industry-standard query language for graphs.

For example, corrupted graph data can result in dangling relationships that point nowhere, phantom properties, and nodes that can only be reached from one direction. The ripple effect of these situations is quite staggering when you consider what happens when subsequent reads against that corrupt data influence new updates or writes, which further spread corruption because they are based on the original corrupt, untrustable values. Ultimately, significant portions of the database become unusable.

For non-native graph databases built on top of other data models, ACID compliance can be equally complex and concerning. For example, key-value stores are optimized for high throughput and low latency on single-key lookups (reads) and writes. They support ACID semantics only at a single-key level. As a result, a graph data store that needs transactional and data consistency capabilities cannot be implemented directly over native key-value stores efficiently.

What to Look for: Durability

Database systems should be evaluated and rigorously tested to ensure that they lose no data when introducing a systemic hardware failure, such as a "kill -9" process interruption, power failure or kernel panic.

It has been observed that some native graph and multi-model database systems are unable to recover gracefully here, both by losing transactions that the system had previously accepted and/or failing consistency checks after recovering from the failure itself.

What to Look for: Consistency Checking for Graphs

When you're evaluating vendors, here are the questions you should consider when it comes to data consistency and corruption:

- ❒ Does the database offer any level of consistency checking for graphs?
- ❒ What safety measures are available to ensure writes are written correctly?
- ❒ Do these measures work as expected in a clustered, High Availability (HA) environment?
- ❒ Does the database vendor's documentation contain warnings, such as beware of ghost vertices or floating or untethered edges?

All of these questions should serve as warnings of data corruption.

Today, during the emergence of NoSQL and graph databases, it is not as easy to find trained developers who know each vendor's proprietary query language.

Graph Query Languages

Unlike the ISO/ANSI SQL that has served RDBMS users as the standard query language for 40 years, there is not yet an industry-standard query language for graphs.

Consequently, graph query language support varies widely across products. A number of graph-specific query languages have therefore been developed. Some of these support only the vendor's own product, whereas others – Cypher and Gremlin – support a number of DBMS implementations as well as third-party graph visualization and integration tools.

A subclass of the single-product languages use their own SQL-like dialects adapted by the vendor in an appeal to users' existing experience and comfort with SQL. Such approaches – while appearing familiar on the surface – tend to elude the desired benefits.

These SQL-like dialects are often supported by only one database vendor, lack a community of skilled users together with a third-party tooling ecosystem, and also tend to miss out on the technical benefits of query languages designed from the ground up for the property graph model.

Below is a review of more detailed considerations, but here is a summary of where most users end up: nearly all graph database vendors offer either direct Cypher support, direct Gremlin support or both. All graph databases that support Gremlin can now also run Cypher, thanks to openCypher's "Cypher for Gremlin" project, which has been donated to the Apache Foundation under the official TinkerPop umbrella, making Cypher support part and parcel of TinkerPop.

Also worth noting is that the vast majority of graph database users (which encompasses developers as well as projects) are using Cypher. There is an active and concerted effort inside of the ISO and the openCypher community to create a new standard language under the International Standards Organization umbrella as a sibling language to SQL. This language is inspired by and would be highly compatible with Cypher and its own sibling graph query language derivatives.

What to Look for: Declarative vs. Imperative Languages

Declarative languages such as SQL – as opposed to imperative languages like Java – tend to be advantageous as database query languages for a variety of reasons.

Declarative query languages are more easily learned, as the query author needs only to instruct the database about what to retrieve and not be concerned about the low-level details of how the database should go about obtaining it.

Declarative languages are also easier to write, read and debug for end users than imperative languages. This declarative approach has been a key factor in SQL's popularity over the years. Today, the most popular, fully declarative graph query language is Cypher.

Imperative languages offer fine-grained control over every behavior of the query by instructing the system where to go as it touches every node, almost like when one learns hopscotch. The issue with imperative languages is that the developer must be intimately familiar with the graph schema they are querying in order to tell the system how to execute each traversal, which increases both the learning curve for the language as well as the complexity of the code being written.

The most popular imperative graph query language is Gremlin, though Gremlin also combines some declarative attributes.

The most successful graph applications help organizations innovate more quickly, generate and optimize revenue streams, and improve the customer experience by preventing unexpected maintenance.

Declarative languages require far more work on the part of the database vendor to build and optimize, as they effectively render what a user is asking for into a set of low-level imperative commands that the database carries out. TinkerPop is thus relatively easy to port to any database backend. On the other hand, supporting Cypher requires a sophisticated combination of parsing, planning and runtime work, as well as database statistics and cost-based optimization techniques, all of which require substantial effort to write and maintain.

What to Look for: Compiled vs. Interpreted Instructions

Some graph query languages must be compiled into binary packages prior to execution within the database. While this technique may be advantageous for optimizing the performance of a single query, it undoubtedly increases debugging complexity and extends the time it takes to write, build and execute queries.

Compiled queries are also a major obstacle when it comes to ad-hoc querying by end users and end-user tools. This effectively eliminates the ability to change a query on the fly – perhaps to just correct a typo – or to modify an ad-hoc query parameter.

What to Look for: Graph Query Language Standardization

The relational database market of the 1980s grew much more rapidly after multiple vendors adopted ANSI SQL as their standard query language. This made it easier to find and hire skilled resources for nearly any project and created more application portability across platforms.

Today, during the emergence of NoSQL and graph databases, it is not as easy to find trained developers who know each vendor's proprietary query language. This will remain true until another standard language takes shape.

Developing a new standard not only requires an abundant user community, but also cooperation among vendors, large and small. This cooperation has not happened in the NoSQL space, as the SQL Standards oversight team chose to simply adapt the ANSI SQL standard to accommodate new NoSQL-style syntaxes and data types.

Graph Query Language (GQL) standards development is underway. This new standard is emerging both from the usage and adoption preferences of the graph tech user community, as well as the adoption and implementation of open source language toolkits within the vendor community.

What to Look for: GQL vs. Gremlin

Apache TinkerPop enjoys popularity across a variety of vendor implementations. Gremlin is TinkerPop's query language and is supported by many vendors including Neo4j, who co-authored Gremlin's original TinkerPop engine. Gremlin's drawbacks, however, are explained above.

Contrast this to the openCypher project with growing support from SAP HANA, Databricks and the Apache Spark project, Redis, Memgraph, Bitnine and Cambridge Semantics. There is also a swell of support for GQL, as the ISO W3C (the SQL Standards body) held a discussion on endorsing both a SQL-compatible, yet independent, Graph Query Language based on inputs from the Linked Data Benchmark Council (LDBC), Neo4j's Cypher and Oracle's Property Graph Query Language (PGQL).

As the software industry has evolved, most IT organizations have become wise to the configuration and execution games played among vendors, giving rise to the term "benchmarketing."

Hybrid Transactional-Analytic Platforms (HTAP)

Database use cases are generally divided into two top-level categories – those that support online transaction processing (OLTP) applications and those that support analytic activities (OLAP), including data warehouses, business intelligence, data discovery, data mining, etc.

While graph database use cases can fall into these general buckets of OLTP and OLAP, they often layer or blend analytic and transactional information in an approach known as Hybrid Transactional-Analytic Platforms (HTAP).

For example, a graph-powered metadata management application can act as both a knowledge graph for data lineage as well as a recommendation application. Or, a data scientist can develop AI instructions using graph analytic algorithms and then deploy those algorithms to a transactional AI application that makes personalized recommendations to each user.

OLTP Applications

Common OLTP use cases for graphs include identity and access management, real-time recommendations, social network publishing, fraud detection, network systems operations, cybersecurity, Internet of Things (IoT), artificial intelligence, supply chain, institutional knowledge recall, customer 360, digital transformation and metadata management.

Often, these are combined or layered to feed one another. Moreover, these transactional applications must integrate a variety of technologies beyond the underlying data management system. Therefore, the graph technology must support integrations across use cases, data interfaces, deployment environments, programming languages, user experience and more.

What to Look for: Verified Use Cases vs. Website Messaging

Ask each supplier for example implementations and references for each of your use cases.

If any vendor is unable to supply solid customer references, then there's a high likelihood that the implementations only exist in theory. The greater number of example successes for your use case, the higher the probability of your own success in implementing a graph solution.

Be sure to ask any vendor that provides use case examples whether the referenced company has a separate commercial relationship with the graph provider – for example, as an investor or business partner.

The most successful graph applications help organizations innovate more quickly, generate and optimize revenue streams, and improve the customer experience by preventing unexpected maintenance. Make sure your graph technology vendor knows how and why their customers and users chose graphs for their application. Beware of vendor inexperience and "foilware."

Verified community and customer experience should provide higher degrees of assurance in your search for the right vendor.

When data is imported into a graph, the data's relationships must be materialized. Furthermore, due to the schema-optional nature of graph systems, data is easily changed, added and removed.

What to Look for: Geospatial Support

Transactional graph applications are frequently designed to describe the interaction activities among people with a variety of digital and analog objects in the context of their location, time, space and the digital networks within which they operate.

If these applications, filled with graphs of people, objects, locations, events and networks sound familiar, then the system within which you store and query this information should support advanced data types, such as geospatial and temporal types.

What to Look for: Temporal Support

Many graphs have temporal requirements since they contain information about when activities occurred, and therefore how they are sequenced in the graph.

- ❒ Does the graph system support time formats and calculations?

What to Look for: Full-Text Search & Indexing

Many graph applications parse and search documents and large parcels of text such as maintenance orders, parts assemblies, research documents and even web pages. In these cases, full-text search and indexing are critical for optimal performance.

- ❒ Does the graph system support mapping, indexing and searching for specific text within the graph? How is that performed?

A graph platform should include features for all of these users, offering extensions, apps, libraries, APIs and SDKs, as well as integration with a variety of complementary technologies.

Query Traversal Performance Benchmark Comparisons

Benchmarks are a popular product comparison vehicle, but you should observe graph benchmark results with a skeptical eye. As the software industry has evolved, most IT organizations have become wise to the configuration and execution games played among vendors, giving rise to the term "benchmarketing."

When evaluating vendor-sponsored benchmarks, look for verifiable and reproducible tests that include data, documentation, configuration and execution methods as well as results. Look for a standard toolkit and dataset – for graph applications this is the Linked Data Benchmark Council's (LDBC) array of tests.

Look for independent third parties that have performed comparisons – even when working with the graph vendor. These comparisons are preferred over highly biased, vendor-developed and staged tests.

What to Look for: Query Traversal Performance Tricks Played by Vendors

When creating performance benchmarks, vendors often use tricks to improve their product performance while simultaneously causing their competitors to perform poorly. Here are a set of key questions to ask while evaluating any vendor benchmark:

- ❒ Does performance deteriorate exponentially when the number of traversal hops across the graph increases? (This is very likely due to the transposition issues mentioned above, or whether the queries are run as executables or within a runtime interpreter.)
- ❒ Does the benchmark compare a pre-compiled binary query against an interpreted query? Is that fair, or should both systems run pre-compiled queries?
- ❒ Does the vendor fully exercise the system or simply highlight its most favorable tasks and results? For example, are workloads mixed as both transactional writes and analytic reads, or is only one type of activity executed?
- ❒ Are caching strategies equal? Are both graphs fully loaded into memory (where they will be extra fast), or is only one product fully loaded into memory while the other swaps between cache and disk?
- ❒ Are one-time data loading operations overly emphasized, versus multi-user or multi-session tests, which better reflect regular usage?
- ❒ Are benchmark sessions multi-user, or are they single sessions that utilize all available CPU resources to parallelize query execution? How realistic is it that one user will use the database at a time?
- ❒ More broadly, do the benchmarks consider real-world, mixed read-write workloads?
- ❒ Are the systems configured optimally and fairly: system sizing, memory, tuning config parameters?

Look out for benchmarks that claim fairness by using database defaults, which are often extremely minimal in order to support small-scale functional evaluation rather than performance at scale.

A visual development environment should be included with the database to help developers write clean, clear queries and also understand what those queries return.

Graph Analytics

Graph analytics are unlike traditional aggregation-based analysis.

Graph analytics tasks search for pathways, clusters, similarities and contextualization. Given the newness of graph analytic exercises, you should place a higher degree of confidence in vendors who offer a variety of educational, implementation and integration materials to assist data scientists and other data analysts.

What to Look For: Publications & Educational Material

- ❒ Has the vendor not only published content about graph systems, but also published information about how to perform graph analytics exercises using graph algorithms?
- ❒ If not, then how do they teach users how to use any graph algorithm libraries they might offer?

What to Look for: Robust Library of Graph Algorithms

Graph algorithms help users readily identify patterns, paths, clusters and similarities among graph data. These algorithms are used to support AI applications and reveal interesting patterns during analysis. Vendors should offer support for popular graph algorithms, including documentation explaining their appropriate use and application.

Data Integration & Ingestion

Data ingestion and integration are just as important for graphs as they are for SQL and NoSQL systems.

When data is imported into a graph, the data's relationships must be materialized. Furthermore, due to the schema-optional nature of graph systems, data is easily changed, added and removed. Vendors will outline their ability to ingest high volumes of data rapidly, which is primarily useful when a dataset first enters the graph. Enterprises who have invested in other big data technologies will need integration to their Hadoop installations, ideally using already-deployed data processing technologies such as Apache Spark.

It also may become necessary for data to be streamed or imported into the graph system on a regular basis. Therefore, the inclusion or integration with data processing tools such as Kettle or Kafka (open source) or commercial tools from Informatica, iWay, Trifacta or others is desirable from any graph technology vendor.

What to Look for: Data Lake Integrations

Data lakes have become commonplace across enterprises. This mega-warehouse strategy is designed to centralize data storage in order to feed traditional data warehouses, support a new generation of transactional and analytic applications and ensure consistency of how data is managed, maintained and used.

Organizations who have invested in data lakes – and especially those who use Apache Spark as their analytic processing engine – should consider how easily the graph system integrates with that source data.

A well-built graph visualization tool allows you to explore the graph in a variety of ways, such as filtering nodes and relationships, lassoing parts of the graph, reviewing and editing properties, highlighting paths, coloring nodes categories and including graphic icons.

Some questions to consider for data lake integrations with graph technology:

- ❒ Can the graph system operate within the Apache Spark ecosystem?
- ❒ Can graphs be materialized from Spark Data Frames?
- ❒ Can graph structures be directly imported into the graph database system?

What to Look for: Data Integration & Ingestion Technologies

Data integration and data streaming capabilities are very desirable add-ons for graph systems. Without them, the DBA is left with simple comma-separated value (CSV) import functions, which could be sub-optimal.

Look for libraries, SDKs and data integration adapters to popular data management tools. Open source tools like Talend and Kettle should be available for a given graph system, while commercial product support from Informatica, Ab Initio, iWay, Trifacta or Tamr is desirable for organizations that have made investments in these tools.

What to Look for: CSV Import & High-Speed Ingestion

At a minimum, most products import CSV text files. Any given graph technology vendor should also support additional functions such as high-speed and bulk ingestion.

More advanced data population strategies include the ability to defer index building and consistency checking until after all the data is ingested. However, development teams should be careful not to over-consider the importance of high-speed ingestion. For most use cases, high-volume ingestion is usually more of a setup exercise when compared to ongoing incremental updates, which tend to be much smaller than initial data loads.

Graph Platform with Tools & Support for All Types of Users

Most use cases require more than just a database. Therefore, buyers should also look for capabilities and features that support the entire software engineering lifecycle as well as a wide user base that includes developers, data scientists, DBAs, IT operations and business users.

A graph platform should include features for all of these users, offering extensions, apps, libraries, APIs and SDKs, as well as integration with a variety of complementary technologies.

Traditional RDBMS systems support the ability to host and manage multiple databases within an instance of the software. For graphs, this is not yet the case.

Developer Tooling

Technical user tools are essential for graph developers.

In this area, the vendor should offer a visual programming environment that allows the developer to not only write queries, but to also see the graph upon which they are working.

Developers need a variety of drivers and APIs (beyond just JDBC) with which to integrate application-level functionality. Furthermore, the database should be embeddable within a larger application as is often necessary in modern deployments.

What to Look for: Developer Launchpad Application

An ideal developer experience is one that assembles all developer tools in one application wrapper such that the developer always enjoys the same launchpad from which to work with their graph database system.

Neo4j is the only graph database system to offer the Neo4j Desktop: an integrated developer launchpad for graph projects. Neo4j Desktop provides access to the Enterprise Edition of the Neo4j Graph Database along with its accompanying algorithm and procedure libraries, as well as graph application examples. The Neo4j Desktop is free for development use.

What to Look for: Visual Development Environment

A visual development environment should be included with the database to help developers write clean, clear queries and also understand what those queries return.

Features of this environment should include keyword color-coding and auto-completed values drawn from the database. It should also include a storyboard-like guide that allows developers to teach and share step-by-step instructions. A visual graph drawing tool should be included to help visualize the result set.

What to Look for: Application Drivers

Developers choose to write applications in a variety of programming languages, from Java, Python or JavaScript to newer languages such as Go. The database should offer multiple language-specific drivers to ease the burden for developers in designing systems that include graph databases.

Neo4j offers commercial support for Java, JavaScript, .NET, Python and Go, while the Neo4j community offers drivers for even more languages. TinkerPop also provides a number of community-supported language integration drivers.

What to Look for: API Support

Programming interfaces are also important for feeding data into the database using common protocols.

Interfaces such as Java or the database's native language are desirable, as are interfaces and exchange formats for other common occurrences. Any graph technology vendor should also offer support for GraphQL (Facebook's data exchange format) as a way of interacting with applications above the database querying, as well as support for SPARQL, the query language for RDF-based systems (Resource Description Framework), REST and others.

Graph data isn't typically "big data" in the classical sense, and graph databases are not usually swamped with the size of indexes and JOIN tables in the same way as an RDBMS.

- ❒ Does the vendor support GraphQL?
- ❒ Does the vendor support SPARQL?
- ❒ Does the vendor support REST APIs and frameworks?
- ❒ Does the vendor actively support multiple emerging standards such as the GRANDstack or similar?

What to Look for: Embeddability

Database embeddability is desirable for a number of reasons, including the ability for OEMs to package the graph database along with application logic, user interfaces, cloud containers like Kubernetes and more. More embeddability questions to consider include:

- ❒ Can the database be embedded within a surrounding application?
- ❒ How easily can the graph system embed itself in larger applications?
- ❒ Does the vendor support a robust OEM network and/or startup program?

Visualization Capabilities

Graph data visualization tools are extremely helpful in comprehending graph data behaviors. These environments allow users to quickly see shapes within connected data and determine where to look further.

What to Look for: Codeless Search, Business User Support

Business users benefit from a Google-like search and query experience, and a graph visualization environment should therefore support natural language search functions. In addition, visualization tools should help the user learn the graph as they use the environment by offering built-in query suggestions and auto-completion.

Business user questions to consider include:

- ❒ Is the graph data visualization tool codeless for non-technical users?
- ❒ Does the visualization tool support natural language search?
- ❒ Does the graph visualization tool offer built-in query suggestions?
- ❒ Is the visualization tool web-based for ease of access?

What to Look for: Graph Exploration

A well-built graph visualization tool allows you to explore the graph in a variety of ways, such as filtering nodes and relationships, lassoing parts of the graph, reviewing and editing properties, highlighting paths, coloring nodes categories and including graphic icons.

What to Look for: Linking & Embedding

A vendor's graph visualization tool should allow users to link to views of the graph via URL or external application for data illustrations on other mediums. These views should also be fully embeddable for easy sharing and integration.

Database administration is greatly improved with the inclusion of administrative tools not only for DBAs, but also for programmers and analysts.

What to Look for: Developer Code Pasting & Viewing

A robust graph visualization tool should supply fully-formatted query information to developers or other technical users so that they can adjust the first view or perspective presented to other non-technical users.

What to Look for: Graph Schema Perspectives for Data Security

Any data visualization tool should include advanced security features that exploit the graph's ability to show different views of the data from the same graph.

Specifically, the tool should support presenting different views of the data or the structure of the data (the schema of the graph) to different groups of users. Filters should also have the option to be based on group-level security permissions.

What to Look for: Printing, Sharing & Storyboarding

Every graph visualization tool should offer multi-user sharing of graph views so that a storyboard might be created to show multiple steps through a graph dataset. Furthermore, the visualization tool should be able to print or otherwise present graph content.

Deployment, Scaling & Administration

What to Look for: High Availability

Does the graph database platform offer a highly available solution designed to gracefully survive one or more system failures? Often, this requires built-in application and data redundancy. Some systems offer data redundancy upon a single, writable graph, and then provide scaling for reads by cascading updates to replica systems deployed around the world.

Other systems offer redundancy of multiple, duplicate writable graphs, which adds fault tolerances to applications, networks and operations that lie above the storage system. For these systems, it is important that their strength of consistency be carefully evaluated. For graphs, consistency is vitally important, and when writing to multiple duplicates, consistency must be coordinated carefully.

What to Look for: Write Scaling

Scaling graph writes is arguably the most difficult challenge faced by all graph database vendors.

Current market expectations are that graph systems should horizontally scale for writes as easily as they might for read optimization. The reality, however, is that creating a horizontally scalable graph system requires the user to understand and plan their graph schema carefully. For example, a user would need to design their schema so that the graph could be partitioned in some form and therefore fit within any hardware configuration.

Once this takes place, the user must then recognize how each partition will be identified and how queries will be directed to the proper partition. Finally, the system will ultimately need a means to leap queries from one partition to an adjacent one, which no graph database product currently supports.

Even within the almost wholly online world of data technology, face-to-face events, gatherings and conferences are just as important since they establish and strengthen relationships among users, customers, partners and employees.

Due to every vendor's inability to easily break apart a graph, write scaling must scale vertically, requiring beefier hardware or a cloud environment. Such large, vertically scaled systems can scale writes in the following ways:

- ❒ By offering more CPU-core backed sessions to accommodate concurrent connections
- ❒ By increasing RAM in which to cache most of the graph
- ❒ By providing enough CPU power to minimize transaction cycle times

What to Look for: Multi-Graph Support

Some graph vendors have begun to solve the above-mentioned writing challenge. These vendors have identified means by which to isolate, view and operate upon one or many graph clusters.

Vendors are also identifying ways in which to extend application drivers to support directing queries to different graphs using a multi-graph routing table. While these graphs are distinct, this is an illustration of the first stage in supporting a partitioned graph solution.

What to Look for: Multi-Database & Multi-Tenancy

Traditional RDBMS systems support the ability to host and manage multiple databases within an instance of the software. For graphs, this is not yet the case. Nearly all vendors are single graph per software instance. Be sure to ask graph vendors these questions:

- ❒ What does their product roadmap include for multiple databases?
- ❒ Will their planned system support multiple tenants, hosting multiple databases?

What to Look for: Read Scaling

Read scaling is significantly easier for graphs than scaling writes.

Many graph database vendors support the ability to offer readable replicas of the main graph. Some systems scale these replicas hierarchically. It is important to understand how replicas stay consistent with the main writable-core instance to ensure that an application or client can read their own writes (RYOW).

Now more than ever, enterprise organizations are deploying database instances to the cloud in addition to on-premises.

What to Look for: Caching Strategies

Caching all or most of a graph in RAM offers a notable performance boost. In-memory graphs significantly outperform those that are not cached.

Other questions a vendor should be able to answer about their graph-caching strategies include:

- ❒ If only sections of a graph fit in memory, then what functions does the vendor offer?
- ❒ Can a language driver point queries to pre-cached information?
- ❒ Can the system reinstate or reheat the cache after system restart?

What to Look for: Scaling Data Volumes

Graph data isn't typically "big data" in the classical sense, and graph databases are not usually swamped with the size of indexes and JOIN tables in the same way as an RDBMS. In fact, both JOIN tables and large indexes fall away when moving your data into a native graph database.

Scaling large volumes of data is accomplished via big data processing technologies such as Apache Spark for analytic activities or Apache Kafka for data streaming.

Ask vendors about their ability to execute graph queries upon Apache Spark, or whether they can materialize graphs from Spark DataFrames. Look for mature clustering technologies that use tools such as the Raft protocol instead of a simpler master-slave architecture.

What to Look for: Cloud Support

Now more than ever, enterprise organizations are deploying database instances to the cloud in addition to on-premises. Ask these questions when considering a graph database vendor who claims to have cloud support:

- ❒ Does the vendor offer both on-premises and cloud-based versions of its software?
- ❒ Does the vendor offer support for multiple cloud vendors beyond Amazon Web Services, such as Microsoft Azure, Google Cloud Platform or Alibaba Cloud?
- ❒ Are pricing and consumption costs transparent to the customer?
- ❒ Does the cloud offering mirror the vendors' on-premises functionality?

What to Look for: DBaaS Support

Analysts have predicted that a fully functional Graph Database-as-a-Service will become very popular in the years to come. Ask vendors these questions in relation to their DBaaS plans:

- ❒ What is the vendor's plan to offer this type of functionality?
- ❒ What subscription tiers will they offer?
- ❒ Is there a multi-tenant offering allowing a value-added software vendor to host multiple downstream customers?

What to Look for: Container Support & Management

Using containers benefits the organization in multiple ways, including simplified configurations, easier upgrade transitions and scaled replicas.

- ❒ What container-based operating environment is supported by the graph system – Docker, Kubernetes or others?
- ❒ Is this a common deployment model for the vendor?

What to Look for: Security

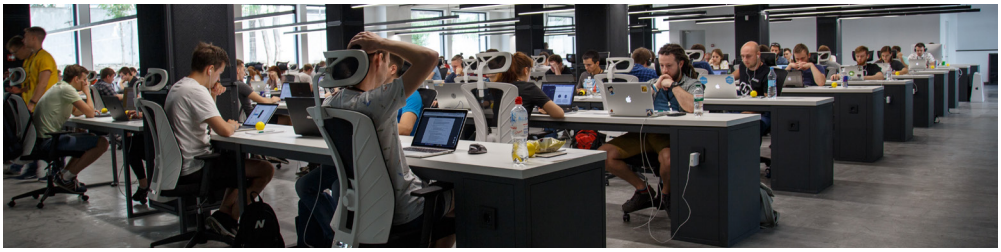
Database integrity and security are extremely important in enterprise data management systems. Ask vendors about these most common database security features and integrations:

- ❒ How does the environment secure users' authentication and authorization rights?
- ❒ Does the system integrate with popular directories such as LDAP or Active Directory?
- ❒ Are security groups supported?
- ❒ Is Kerberos supported?
- ❒ Are client-server traffic and server-to-server traffic encrypted inside of the cluster?
- ❒ Is event security logging supported and can administrators disable and re-enable users?
- ❒ Is security transparent to the developer, or must the developer themselves manage security within the applications they build?
- ❒ Is the data encrypted on disk? (This is often done at the operating system and file system level.)
- ❒ Is the data encrypted in transit in server-to-server activities, in administrative functions and to users' browsers?

What to Look for: Database Administration

Database administration is greatly improved with the inclusion of administrative tools not only for DBAs, but also for programmers and analysts.

- ❒ Does the system offer the ability to check logging activity or query executions?
- ❒ Can users terminate runaway queries?
- ❒ Can users view query execution plans?
- ❒ Can administrators review usage statistics that may be used for billing or departmental chargeback?



Business Model, Focus & Staying Power

Many vendors offer graph extensions and add-ons, which may not provide the optimal graph experience. They may also unintentionally introduce corruptions, as detailed in the earlier section on native graph storage.

- ❒ To what degree is the vendor focused specifically on graph support?
- ❒ What level of operations and activity are dedicated to graph systems?

Native graph systems and vendors have and will dedicate significantly more resources to the success of graphs than multi-model vendors, whose attention will be distracted by the other database models they claim to support.

What to Look for: Partnerships

Ask who the vendor considers to be their key partners in the market. Partnerships should include cloud vendors, consulting and implementation providers, software OEMs, value-added resellers, territorial resellers and education centers.

- ❒ Does the vendor have a wide variety of partnerships throughout both business and technology landscapes?
- ❒ How closely does the vendor manage and work with their partners? Are they partners in name only?

What to Look for: Ecosystem

You should also consider a vendor's customer and user ecosystem. Questions that you should ask a vendor include:

- ❒ How large, vocal and active is the vendor's ecosystem?
- ❒ What are their most frequent questions, issues and ideas?
- ❒ Where do they share ideas and information – a community forum, news group or developer website?

What to Look for: Events & Conferences

Even within the almost wholly online world of data technology, face-to-face events, gatherings and conferences are just as important since they establish and strengthen relationships among users, customers, partners and employees. Questions you should ask a vendor include:

- ❒ What events does the vendor host to develop its community?
- ❒ How many regular meetups does it participate in?
- ❒ What breadth of geography does a vendor cover in their events and in-person outreach?
- ❒ Does the vendor hold a regular conference dedicated to graph technology?

What to Look for: Customer Feedback

Any vendor can claim to have top-name customers who are happy with their product, but not every vendor can back up those claims with solid references and case studies. Questions you should ask a graph technology vendor include:

- ❒ Who are the vendor's referenceable customers?
- ❒ Do they have a public listing of customers available on their website?
- ❒ Will the vendor's sales team allow you to speak to some of their current customers?

Beware of signs that may indicate a conflict of interest, such as a corporate investor also posing as a customer reference.

What to Look for: Licensing Model

Not all customers' software needs are the same. A mature vendor offers a variety of licensing models for their graph database software to match the needs of different customer requirements. Questions that you should ask a vendor include:

- ❒ Does the vendor offer a variety of license models to suit the buyer's preferred deployment and feature combinations?
- ❒ Does the vendor publish their price list?

What to Look for: Mature Technology vs. Technology Past its Prime

Database analyst Curt Monash famously observed that "the first rule of databases is that **it takes at least five to seven years** to build a good, stable one. The second rule of databases is that there is no exception to rule number 1." This maturity requirement rule is intended to reinforce the importance of most of the earlier requirements.

Likewise, look out for aging technology that is losing its place in the market. This is often referred to as the "Goldilocks effect," whereby mature technologies grow stale, paving the way for newer ones. The aging technology is then slowly put to pasture as a cash cow or retrofitted with bolt-on updates that simply add features with little regard to their effectiveness. The result is a Swiss Army Knife – a tool that does a little of everything but none of it well.

This phenomenon is currently happening in the multi-model database space, especially for graphs. Graph add-ons are becoming the saw in a Swiss Army Knife – you can't build a robust house with it.

What to Look for: Staying Power & Customer Support to Minimize Technology Risk

The staying power requirement acts as a reminder that software must be maintained and enhanced, or else it will atrophy and cease to keep pace. Beware of small software teams or teams of consultants charged with writing and maintaining the software. Additionally, be cautious of companies nested inside of companies serving as embedded subsidiaries of a conglomerate.

In addition, a mature graph technology vendor should have the resources necessary to support your graph database deployment should any issue or problems arise. No technology is ever deployed perfectly (no matter whether the vendor or the buyer are at

As the graph technology market continues to grow and mature, purchasing decisions may become more clear as vendors coalesce around shared standards (such as query languages, data formats, etc.).

fault), but given the newness of the graph paradigm, a vendor **must** have a dedicated team of support engineers who can help with your critical issues or questions as they arise. This team should be geographically spread out and available for emergency support around the clock.

Below are questions to ask vendors:

- ❒ Is the vendor well-backed financially?
- ❒ Does the vendor operate as a fiscally responsible business?
- ❒ Is the vendor's product team properly staffed and supported?
- ❒ Does the vendor have a team of professional support engineers to help with critical issues around the globe and around the clock?

Conclusion

Making an informed buying decision for graph technology isn't always easy.

Because the graph paradigm is still such a new way of working with data, purchasing teams are often held back by RDBMS norms when considering how to vet a graph database vendor. But as has been shown above, factors such as data consistency, performance, scalability and other important characteristics all have different applications when it comes to graph databases versus their relational counterparts.

As the graph technology market continues to grow and mature, purchasing decisions may become more clear as vendors coalesce around shared standards (such as query languages, data formats, etc.). But until then, we hope that this buyer's guide has been helpful in clarifying which factors are most important when considering a graph database product – and which factors might just be marketing fluff with no substance behind them.

For your convenience, we've included a blank checklist in Appendix A that lists all of the factors outlined in the sections above.

If you'd like to talk to a graph expert about the advantages of the Neo4j Graph Platform, please contact us today.

This Appendix includes a blank checklist for the reader in order to independently evaluate and compare graph database choices.

Appendix A

The Buying Criteria for Graph Technology

Open Source Foundation & Community

- ❒ Is the graph database built on an open source foundation?
- ❒ Does the graph database vendor have an active open source community?

Native Graph Storage

- ❒ Transposition vs. Persisted Relationships
  - ❒ Is the offering a native or non-native graph database?
  - ❒ Is there a graph-to-document or graph-to-column transposition step buried in the software?
- ❒ Index-Free Adjacency
  - ❒ Does the graph database support index-free adjacency for high-performance graph traversals across deep graph datasets?

ACID Compliance

- ❒ Durability
  - ❒ Is the graph database systems rigorously tested for data durability?
  - ❒ Does the graph database lose data during systemic hardware failures?
- ❒ Consistency Checks
  - ❒ Does the database offer any level of consistency checking for graphs?
  - ❒ What safety measures are available to ensure writes are written correctly?
  - ❒ Do these measures work as expected in a clustered, High Availability (HA) environment?
  - ❒ Does the database vendor's documentation contain warnings, such as beware of ghost vertices or floating or untethered edges?

Graph Query Languages

- ❒ Declarative vs. Imperative Query Language
  - ❒ Does the graph database system use a declarative query language, imperative query language or a mixture of both?
- ❒ Compiled vs. Interpreted Instructions
  - ❒ Are queries compiled into binary packages prior to execution within the database (optimization for a single query)?
  - ❒ Or, are queries interpreted by the database in a way that allows for ad-hoc querying and changes on the fly (optimization for user agility)?
- ❒ Graph Query Language Standardization
  - ❒ Is the database vendor part of the ISO W3C standardization movement around Graph Query Language (GQL)?
  - ❒ Does the vendor have plans to be compatible with GQL in the future?

Hybrid Transactional-Analytic Platforms (HTAP)

- ❒ OLTP Applications
  - ❒ Does the graph database vendor have verified example implementations and customer references for a variety of OLTP use cases?
  - ❒ Does the graph database support geospatial data types?
  - ❒ Does the graph system support temporal formats and calculations?
  - ❒ Does the graph system support mapping, indexing and searching for specific text within the graph? (full-text search and indexing)
- ❒ Query Traversal Performance Benchmark Comparisons
  - ❒ If the vendor has published a benchmark comparison, does it provide verifiable and reproducible tests that include data, documentation, configuration and execution methods as well as results?
  - ❒ Did the vendor use the Linked Data Benchmark Council (LDBC) array of tests?
  - ❒ Did the vendor work with independent third parties that have performed other vendor-neutral comparisons?
  - ❒ Does database performance deteriorate exponentially when the number of traversal hops across the graph increases?
  - ❒ Does the benchmark compare a pre-compiled binary query against an interpreted query?
  - ❒ Does the vendor fully exercise the system or simply highlight its most favorable tasks and results?
    - ❒ For example, are workloads mixed as both transactional writes and analytic reads, or is only one type of activity executed?
  - ❒ Are caching strategies equal?
    - ❒ For example, are both graphs fully loaded into memory, or is only one product fully loaded into memory while the other swaps between cache and disk?
  - ❒ Are one-time data loading operations overly emphasized – versus multi-user or multi-session tests – which better reflect regular usage?
  - ❒ Are benchmark sessions multi-user, or are they single sessions that utilize all available CPU resources to parallelize query execution?
  - ❒ Does the benchmark consider real-world, mixed read-write workloads?
  - ❒ Are the systems configured optimally and fairly along criteria such as system sizing, memory and/or tuning config parameters?
  - ❒ Does the benchmark only use database defaults, rather than being optimized for performance at scale?
- ❒ Graph Analytics (OLAP Applications)
  - ❒ Has the vendor published information about how to perform graph analytics exercises using graph algorithms?
  - ❒ Does the vendor offer support for popular graph algorithms?
  - ❒ Does the vendor include documentation explaining the appropriate use and application of various graph algorithms?
- ❒ Data Integration & Ingestion
  - ❒ Does the graph system operate within the Apache Spark ecosystem?
  - ❒ Can graphs be materialized from Spark DataFrames?

- ❒ Can graph structures be directly imported into the graph database system?
- ❒ Can graph algorithms be run against data in Apache Spark?
- ❒ Does the vendor offer libraries, SDKs and data integration adapters to popular data management tools?
  - ❒ Open source tools: Talend and Kettle
  - ❒ Commercial tools: Informatica, Ab Initio, iWay, Trifacta and Tamr
- ❒ Does the graph database support CSV data import?
- ❒ Does the graph database offer high-speed and bulk data ingestion?
- ❒ Does the graph solution include the ability to defer index building and consistency checking until after the data is ingested?

Graph Platform with Tools & Support for All Types of Users

- ❒ Developer Tooling
  - ❒ Does the platform include a launchpad application wrapper for all developer tools related to the graph database system?
  - ❒ Does the platform offer a visual development environment to help developers write clean, clear queries and understand what those queries return?
    - ❒ Does the visual development tool include keyword color-coding and auto-completed values drawn from the database?
  - ❒ Does the platform offer commercial support for multiple language drivers for the most popular programming languages?
    - ❒ Does the vendor's wider user community offer drivers for other popular programming languages?
  - ❒ Does the graph platform offer API support for Java or the database's native language?
  - ❒ Does the platform include support for GraphQL?
  - ❒ Does the platform support integration with SPARQL?
  - ❒ Does the platform support REST APIs and frameworks?
  - ❒ Does the vendor actively support multiple emerging standards such as the GRANDstack or similar?
  - ❒ Can the database be embedded within a surrounding application?
  - ❒ How easily can the graph system embed itself in larger applications?
  - ❒ Does the vendor support a robust OEM network and/or startup program?
- ❒ Data Visualization Capabilities
  - ❒ Does the platform include a native graph data visualization tool?
  - ❒ Is the graph data visualization tool codeless for non-technical users?
  - ❒ Does the visualization tool support natural language search?
  - ❒ Does the graph visualization tool offer built-in query suggestions?
  - ❒ Is the visualization tool web-based for ease of access?
  - ❒ Does the visualization tool allow non-technical users to easily navigate and explore a given graph dataset?
  - ❒ Does the graph visualization tool allow users to link to (or embed) views of the graph?
  - ❒ Does the visualization tool let technical users execute Cypher queries or adjust perspectives for other non-technical users?
  - ❒ Does the data visualization tool include advanced security features such as user- or group-level viewing permissions?

- ❒ Does the visualization tool offer multi-user sharing of graph views, storyboarding, printing or other presentation methods?

❒ Deployment, Scaling & Administration

- ❒ Does the graph database platform offer a highly available solution designed to gracefully survive one or more system failures?
- ❒ Does the graph platform vertically scale for your expected write load?
- ❒ Does the graph platform include multi-graph support for both writes and reads?
- ❒ Are multi-database features included in the vendor's product roadmap?
  - ❒ Will the vendor's planned system support multiple tenants, hosting multiple databases?
- ❒ Does the graph platform support robust read scaling?
- ❒ Does the graph database offer readable replicas of the main graph?
- ❒ How do replicas stay consistent with the main writable-core instance to ensure that an application or client can read their own writes (RYOW)?
- ❒ What are the vendor's graph-caching strategies?
- ❒ If only sections of a graph fit in memory, then what functions does the vendor offer?
- ❒ Can a language driver point queries to pre-cached information?
- ❒ Can the system reinstate or reheat the cache after system restart?

❒ Can the graph platform execute graph queries upon Apache Spark?

- ❒ Can the graph platform materialize graphs from Spark DataFrames?

❒ Does the graph platform utilize mature clustering technologies such as the Raft protocol, or does it rely on a simpler master-slave architecture?

❒ Does the vendor offer both on-premises and cloud-based versions of its software?

- ❒ Does the vendor offer support for multiple cloud vendors beyond Amazon Web Services, such as Microsoft Azure, Google Cloud Platform or Alibaba Cloud?
- ❒ Are pricing and consumption costs transparent to the customer?
- ❒ Does the cloud offering mirror the vendors' on-premises functionality?

❒ Does the vendor plan to offer a Graph Database-as-a-Service (DBaaS) in the future?

- ❒ What is the vendor's plan to offer this type of functionality?
- ❒ What subscription tiers will they offer?
- ❒ Is there a multi-tenant offering allowing a value-added software vendor to host multiple downstream customers?

❒ What container-based operating environment is supported by the graph system: Docker, Kubernetes or others?

- ❒ Is this a common deployment model for the vendor?

❒ What are the graph vendor's most common database security features and integrations?

- ❒ How does the environment secure users' authentication and authorization rights?
- ❒ Does the system integrate with popular directories such as LDAP or Active Directory?
- ❒ Are security groups supported?
- ❒ Is Kerberos supported?

- ❒ Are client-server traffic and server-to-server traffic encrypted inside of the cluster?
- ❒ Is event security logging supported?
- ❒ Can administrators disable and re-enable users?
- ❒ Is security transparent to the developer, or must the developer themselves manage security within the applications they build?
- ❒ Is the data encrypted on disk?
- ❒ Is the data encrypted in transit in server-to-server activities, in administrative functions and to users' browsers?

❒ What database administrative tools are included in the graph platform for DBAs, programmers and analysts?

- ❒ Does the system offer the ability to check logging activity or query executions?
- ❒ Can users terminate runaway queries?
- ❒ Can users view query execution plans?
- ❒ Can administrators review usage statistics that may be used for billing or departmental chargeback?

Business Model, Focus & Staying Power

❒ To what degree is the vendor focused specifically on graph support?

- ❒ What level of operations and activity are dedicated to graph systems?

❒ Who are the vendor's key partners in the market?

- ❒ Does the vendor have a wide variety of partnerships throughout both business and technology landscapes?
  - ❒ Cloud vendors?
  - ❒ Consulting and implementation providers?
  - ❒ Software OEMs?
  - ❒ Value-added resellers?
  - ❒ Territorial resellers?
  - ❒ Training and education centers?
- ❒ How closely does the vendor manage and work with their partners?
  - ❒ Are they partners in name only?

❒ What is the vendor's customer and user ecosystem like?

- ❒ How large, vocal and active is the vendor's ecosystem?
- ❒ What are their most frequent questions, issues and ideas?
- ❒ Where do they share ideas and information – a community forum, news group or developer website?

❒ What sorts of face-to-face events and conferences does the vendor participate in?

- ❒ What events does the vendor host to develop its community?
- ❒ How many regular meetups does it participate in?
- ❒ What breadth of geography does a vendor cover in their events and in-person outreach?
- ❒ Does the vendor hold a regular conference dedicated to graph technology?

- ❒ Who are the vendor's reference customers?
  - ❒ Do they have a listing of public customers available on their website?
  - ❒ Will the vendor's sales team allow you to speak to some of their current customers?
  - ❒ Do any of the vendor's reference customers hold a conflict of interest, such as also being a corporate investor?
- ❒ What is the vendor's licensing model for their graph database software?
  - ❒ Does the vendor offer a variety of license models to suit the buyer's preferred deployment and feature combinations?
  - ❒ Does the vendor publish their price list?
- ❒ How long has the vendor been working on their graph database product?
  - ❒ Does is meet the Monash Rule of 5-7 years of minimum development for a stable database product?
  - ❒ Are the graph capabilities merely an add-on of an older, legacy product?
- ❒ Is the graph technology vendor have the business staying power to build, update and support the product for years to come?
  - ❒ Is the vendor well-backed financially?
  - ❒ Does the vendor operate as a fiscally responsible business?
  - ❒ Is the vendor's product team properly staffed and supported?
  - ❒ Does the vendor have a team of professional support engineers to help with critical issues around the globe and around the clock?

Neo4j is the leading graph database platform that drives innovation and competitive advantage at Airbus, Comcast, eBay, NASA, UBS, Walmart and more. Hundreds of thousands of community deployments and more than 300 customers harness connected data with Neo4j to reveal how people, processes, locations and systems are interrelated.

Using this relationships-first approach, applications built using Neo4j tackle connected data challenges including artificial intelligence, fraud detection, real-time recommendations and master data. Find out more at Neo4j.com.

Questions about Neo4j?

Contact us around the globe:
info@neo4j.com
neo4j.com/contact-us