neo4j

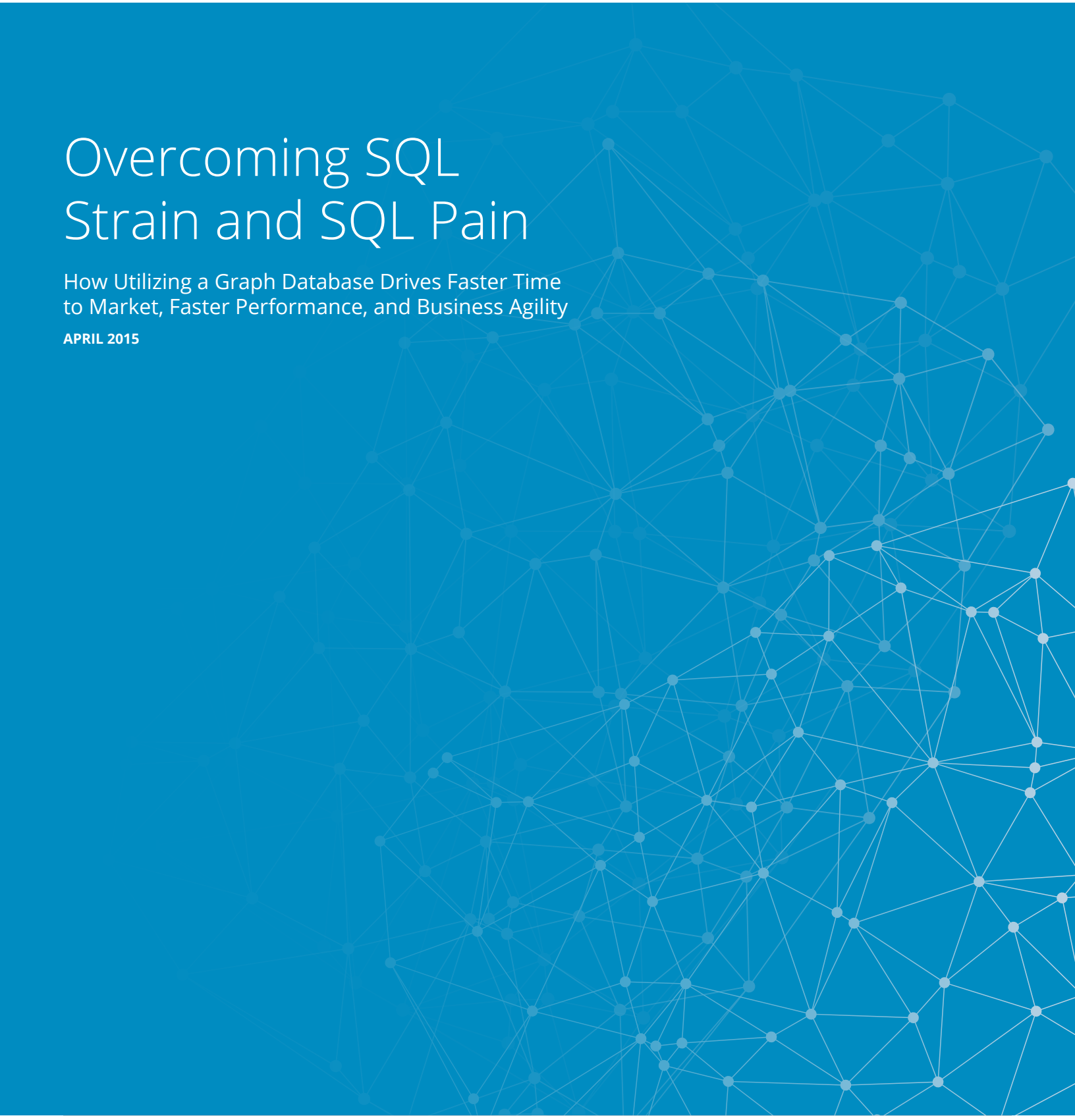
The World's Leading Graph Database

White Paper

Overcoming SQL Strain and SQL Pain

How Utilizing a Graph Database Drives Faster Time to Market, Faster Performance, and Business Agility

APRIL 2015



White Paper

TABLE OF CONTENTS

Signs of SQL Strain 2

The Impact of the Graph Data Model 2

Solving a Connected Data Problem 3

Data Modeling in a Relational Database versus a Graph Database 3

Writing Queries with a Relational Database versus a Graph Database 3

Query Performance with a Relational Database versus a Graph Database 5

Evolution of the Application with a Relational Database versus a Graph Database 5

Top Use Cases for Neo4j 5

Conclusion 6

Overcoming SQL Strain and SQL Pain

How Utilizing a Graph Database Drives Faster Time to Market, Faster Performance, and Business Agility

APRIL 2015

Despite advances in computing, faster processors, and high-speed networks, the performance of some database applications is becoming slower and slower. We can see the problem in lengthening query times and in workarounds such as precomputing query results in advance to serve application needs instead of performing real-time queries. At a time when business agility is at a premium, business requests for changes are put off because the schema of relational databases isn't designed for frequent changes and pivots.

The problems just described are happening because of an overall evolution not only in the volume and velocity of data, but in its variety, complexity, and interconnectedness, that is, the data relationships inherent in the data. The tidal wave of today's data can be characterized as densely connected, semi-structured, and with a high degree of data model volatility. And as the volume, velocity and variety of data are increasing, the data relationships are growing even faster.

Relational databases were designed for tabular data, with a consistent structure and a fixed schema. They work best for problems that are well defined at the outset. Attempting to answer questions about data relationships (consider a product recommendations engine, a social graph, or the connections involved in uncovering patterns of fraud) with a relational database involves numerous JOINs between database tables. Despite their name, relational databases do not store relationships between data elements; they are not well suited for today's highly connected data.

Despite their name, relational databases do not store relationships between data elements; they are not well suited for today's highly connected data.

Relational databases do not adapt well to changes; they have a fixed schema. DBAs and developers face a steady stream of business requests to add elements or attributes to meet changing business requirements, such as storing information about the latest social platform. Such schema changes are problematic and take a great deal of time.

This guide is designed to help you recognize problems resulting from using relational databases to manage highly connected data and see how graph databases can be more effective in solving these problems. For DBAs and developers, graph database expertise represents an intuitive new skill that offers important business and career benefits. For business stakeholders, adopting a graph database means faster performance and faster time to market.

Signs of SQL Strain

Many relational database applications are working fine within their limits. Some, however, may be showing significant signs of strain induced by the database, especially when an RDBMS is being used to handle highly connected data. Signs you may be trying to solve a connected data problem with a relational database include:

• Large number of JOINs
When you do a large number of JOINs as a part of a single query, there is an explosion of computing resource consumption and complexity and a corresponding increase in query response times.

• Numerous self-joins (recursive joins), which are common for hierarchy and tree representations
Some of the longest SQL queries in the world involve recursive joins. Traversing relationships by repeatedly joining tables to themselves is inefficient.

• Frequent schema changes
This can be an indication that the data or requirements are rapidly evolving, calling for a more flexible model.

• Slow running queries despite extensive tuning
DBAs may use every trick they know to speed up query times, but the queries still aren't fast enough to support the application's needs. Denormalizing data models for performance impacts data quality and update behavior.

• Precomputing results
Because queries run so slowly, results are precomputed. In effect, you are using yesterday's data for queries that should be handled in real-time. This often entails precomputing 100% of the data even if only 1-2% will be accessed at any given time.

These symptoms may indicate that you are using a relational database to solve a graph problem where the value is derived from data relationships. A graph database is purpose built to store highly connected data and derive value from data relationships in real-time.

The Impact of the Graph Data Model

Relational databases such as Oracle and MySQL excel when it comes to capturing repetitive, tabular data. Despite the word "relational" in their name, relational databases are much less effective at storing or expressing relationships between stored data elements.[1]

Consider the impact that using a graph data model can have in three important areas:

1) Modeling data with a high number of data relationships
2) Flexibly expanding the model to add new data or data relationships
3) Querying data relationships in real-time

In discussions, we draw on a whiteboard and sketch connections between data elements, creating a natural and intuitive data model. Attempting to take a data model based on relationships and forcing it into a tabular framework creates a mental disconnect between the way business stakeholders think about data and processes and the way the database model is implemented. Developer productivity also suffers because the tabular data model is complex, hard to understand, and does not match the developer's mental model of the application (referred to as "object relational impedance mismatch").

The mismatch between the intuitive, related data model from our whiteboard and the tables that will be created in the relational database leads to longer development time, higher project costs, and significant delays in getting to market, as the logical model is painstakingly crafted into a physical model.

The value of the graph data model becomes even clearer when it's time to flexibly expand the model to add new data or data relationships. Projects with rapidly evolving requirements or data sources (which are often the most business critical) are hit hardest by the rigidity

1. The word "relational" in relational databases comes from relating columns in a table, not relating information in different tables. Relationships between columns exist to support set operations. This is very different from the real world where relationships exist between individual data elements.

of relational database models, as changes to the model often require reworking the application, and (if data has already been loaded) migrating the data itself. With a graph data model, changes to the data model can be made with little or no impact to the application.

Unlike a relational database, a graph database is structured entirely around data relationships. Graph databases treat relationships not as a schema structure but as data, like other values. From a relational database standpoint, you could think of this as pre-materializing JOINs once at insertion time instead of computing them for every query. Because the data is structured entirely around data relationships, real-time query performance can be achieved no matter how large or connected the dataset gets.

The best way to understand the difference between relational databases and graph databases is to walk through a sample use case, as we do in the following section.

Solving a Connected Data Problem: Two Approaches

How do relational and graph databases compare from a project standpoint? In order to contrast how you approach development with a relational versus a graph database, let's look at a specific example: a simple product recommendations engine.

The data in this case is highly connected: Customers relate to products and brands, products relate to other products and brands and finally customers relate to other customers. Almost every online retail organization is interested in building a recommendations engine where value can be derived from data relationships. For a recommendations engine there are three key requirements:

1) Model data and the data relationships to understand how recommendations can be made
2) Make recommendations in real-time by querying the data relationships
3) Continually make the model richer by adding more data and more relationships

In the following sections, we'll compare how to approach creating a recommendations engine with an RDBMS and with a graph database, covering the topics of modeling data, writing queries, query performance, and evolving the application.

Data Modeling in a Relational Database versus a Graph Database

In a relational database, data modeling for a very simple recommendations engine requires creating multiple tables. You'd create separate tables for customers, orders, and products, as well as separate intermediate JOIN tables that represents which customers purchased which products in which order.

Data models for a graph database can be easily sketched on the back of a napkin. A customer purchased a certain product. The relationship (customer to product) is stored as part of storing the order. Other customers purchased the same product. What other products did those people purchase?

Modeling with a graph database is whiteboard-friendly; it is intuitive not just for developers but for everyone familiar with the domain. In other words, with a graph database, the logical model, the way we think about the problem, corresponds to the physical model, the way the data is stored, queried, and visualized by the database.

For many of the really interesting problems people are trying to solve today, it's not enough to know that two things are connected. It's also important to know something about that connection: its meaning, its significance, and its strength or weight or quality. For example, who is friends with whom and how strong is that friendship? How are two locations in a logistics network connected and how far apart are they? How many trains per minute follow that line?

A fully featured graph database will move beyond defining data relationships between entities and enable you to incorporate relevant information about the characteristics or strength of those connections.

Writing Queries with a Relational Database versus a Graph Database

Graph data queries are straightforward to write and to understand. Graph databases have their own syntax for such queries. Graph database Neo4j includes a simple but expressive language called Cypher that is purpose built for traversing data relationships. Cypher queries are much simpler than SQL queries; very frequently a long SQL statement can be compressed to many fewer lines in Cypher.

Here's a sample Cypher query for the example just described:

```
MATCH (u:Customer {customer_id:'customer-one'})-[:BOUGHT]->(p:Product)<-
[:BOUGHT]-(peer:Customer)-[:BOUGHT]->(reco:Product)
WHERE not (u)-[:BOUGHT]->(reco)
RETURN reco as Recommendation, count(*) as Frequency
ORDER BY Frequency DESC LIMIT 5;
```

This Cypher query says that for each customer who bought a product, look at the products that peer customers have purchased and add them as recommendations. The WHERE clause removes products that the customer has already purchased, since we don't want to recommend something the customer already bought.

Each of the arrows in the MATCH clause of the Cypher query represents a relationship that would be modeled as a many-to-many JOIN table in a relational model with two JOINs each. So even this simple query encompasses six JOINs across tables.

Here's the equivalent query in SQL:

```
SELECT product.product_name as Recommendation, count(1) as Frequency
FROM product, customer_product_mapping, (SELECT cpm3.product_id, cpm3.customer_id
           FROM Customer_product_mapping cpm, Customer_product_mapping cpm2,
Customer_product_mapping cpm3
           WHERE cpm.customer_id = 'customer-one'
           and cpm.product_id = cpm2.product_id
           and cpm2.customer_id != 'customer-one'
         and cpm3.customer_id = cpm2.customer_id
         and cpm3.product_id not in (select distinct product_id
            FROM  Customer_product_mapping cpm
            WHERE cpm.customer_id = 'customer-one')
      ) recommended_products
WHERE customer_product_mapping.product_id = product.product_id
and customer_product_mapping.product_id in recommended_products.product_id
and customer_product_mapping.customer_id = recommended_products.customer_id
GROUP BY product.product_name
ORDER BY Frequency desc
```

This SQL statement will not only suffer from performance issues due to the JOIN complexity but will also degrade in performance as the dataset gets larger.

Query Performance with a Relational Database versus a Graph Database

Query performance in a relational database is impacted by data growth and the number of JOINs. As tables get bigger, so do indexes, which means that joining the same number of entities requires more and more work. As questions get more challenging, the number of entities you have to join increases. Even if the data volume stays constant, your computational complexity explodes, which impacts query performance. Problems with query performance are driving application developers to use denormalized results rather than returning results from live data.

A graph database is scalable and shows very small increases in query times as data grows. This is because it doesn't compute relationships at query time but stores them at insertion time. In addition, graph queries look at the neighborhood of starting points, so regardless of the total amount of data in the database, the amount of data that is examined remains roughly the same. Organizations that use graph databases report significant decreases in query time.

Evolution of the Application with a Relational Database versus a Graph Database

Change is a certainty with today's applications, and nothing is changing faster than the number of data elements and attributes we want to store. There are good reasons that DBAs resist schema changes if possible. Schema changes can have a massive impact on the database and on other applications. They introduce development and operational overhead as well as risk. Adding a table, a foreign key, or even just a column into a table can affect or even break the applications using that database. Depending on the database, downtime may be required to carry out the schema change. Given these realities, schema changes are often less frequent than business requirements dictate.

Graph databases are schema-optional. This means that there's no need to define schemas in advance. New data elements, new relationships, and new attributes for those relationships can be added at any time. With a relational database, you have to prep the system to accept new types of relationships and that prep takes time. With a graph database, you can inject new data elements and new relationships almost instantaneously.

Top Use Cases for Neo4j

USE CASE	DESCRIPTION	CUSTOMER EXAMPLES
Network and IT Operations	Plan, predict and monitor network behavior by storing physical and virtual device information together with their data relationships	HP, Earthlink
Recommendations	Personalize product, content and service offers by leveraging data relationships	Wal-Mart, eBay, Nomura
Identity and Access Management	Enable authorization and access to resources by storing complex relationships between users, roles, permissions, content and resources	UBS, Telenor
Graph-Based Search	Improve search, retrieval, and analytics of digital content by storing rich content metadata and data relationships	Scripps Networks, National Geographic Society, Beamly, Decibel
Master Data Management	Improve business outcome through storage, merging, and retrieval of complex, connected and hierarchical business information	Cisco, Zephyr, Pitney Bowes, InfoJobs

"We found Neo4j to be literally thousands of times faster than our prior MySQL solution, with queries that require 10-100 times less code."

– Volker Pacher, Senior Developer, eBay

Conclusion

The recent proliferation of database technologies is a testament to the fact that relational databases are not the right tool for every job. A relational database can be complemented effectively by applications running on graph databases.

Supplement your landscape or replace your relational database: the use case is up to you. Because of the natural fit of the graph database model to business domains and processes, Forrester Research estimates that at least 25% of enterprises worldwide will use graph databases by 2017.[2]

Neo4j is an ACID compliant, enterprise grade OLTP graph database that is being widely used for production applications as a replacement for relational databases. It takes time to build a robust database for production use. Work on Neo4j started in 2000, and by 2003, it was the first graph database in 24/7 production. Neo Technology continues to lead the industry. It is notable that Neo4j is the only graph database included in Gartner's 2014 Magic Quadrant for Operational Databases.

The business won't wait. Users won't wait. It's time to choose the right tool for the job, moving applications that show SQL strain to a graph database that reduces query times from minutes to milliseconds. And with so many ways to get started quickly, adding graph to your skillset is the best investment of time you can make.

Add Graph to Your Skillset

For DBAs and developers, it's easy to start experimenting with Neo4j. You can download it and get started right away.

Download Neo4j: neo4j.com/download

From SQL to Cypher: neo4j.com/developer/guide-sql-to-cypher/

GraphAcademy Online Courses: neo4j.com/graphacademy/online-course

Neo4j Community: neo4j.com/community

2. "TechRadar: Enterprise DBMS," Forrester Research, February 13, 2014.

Neo Technology is the creator of Neo4j, the world's leading graph database, that brings data relationships to the fore. From companies offering personalized product and service recommendations; to websites adding social capabilities; to telcos diagnosing network issues; to enterprises reimagining master data, identity, and access models; organizations adopt graph databases as the best way to model, store and query both data and its relationships. Neo Technology researchers pioneered the modern graph database and have been instrumental in bringing the power of the graph to numerous organizations worldwide. Large enterprises like Walmart, eBay, UBS, Nomura, The InterContinental Exchange, Cisco, CenturyLink, HP, Pitney Bowes, Telenor, TomTom, Lufthansa, and The National Geographic Society, as well as startups like CrunchBase, Medium, Polyvore, Zephyr Health, and Elementum use Neo4j to unlock business value from data relationships.

Neo Technology is a privately-held company funded by Fidelity Growth Partners Europe, Sunstone Capital, Conor Venture Partners, Creandum and Dawn Capital, and is headquartered in San Mateo, CA, with offices in Sweden, UK, Germany, France, and Malaysia. For more information, please visit Neo4j.com.

For more information on Neo4j, contact us via email or phone:
1-855-636-4532
info@neotechnology.com