neo4j

Accelerate Fraud Detection With Graph Databases

How Graph Design Patterns Help You Identify and Investigate Suspicious Activity

aws

PARTNER

Foreword

This ebook describes how developers and data scientists can use Neo4j Graph Database to complement their current fraud-detection solutions and improve their results. It explains six common techniques, called graph design patterns, which you can use to rapidly match complex patterns in data and relationships, reveal hidden fraud connections and intermediaries, and find duplicate and suspicious profiles.

Table of Contents

Why Current Fraud Solutions Fall Short

Fraud is increasingly costly and sophisticated. The Association of Certified Fraud Examiners estimates that fraud typically goes undetected for 12 months, costing an average of $8,300 per month. That translates to 5% of revenue ($4.7 trillion) lost to fraud each year.

Companies in various industries, from banking and financial services to arts and entertainment, struggle with fraud detection. Current solutions lack the speed, capabilities, data sources, and insights needed to keep up with new techniques. Plus, sophisticated syndicates use AI, generative AI, and deep-fake technology to circumvent detection and protection tools.

Common fraud-detection methods take reactive approaches. Either someone contacts a company to flag suspicious charges, or predetermined rules built into a fraud software solution trigger an alert. The company then investigates the situation on a case-by-case basis. This method can be effective; however, many alerts turn out to be "false positives," flagging transactions and activity that aren't fraudulent. They often miss more complex fraud schemes, such as creating a false identity that uses other people's (or fake) information to appear authentic.

Ineffective fraud solutions open a company's virtual doors to big losses and put confidential data at risk. Here are some reasons why current fraud solutions can't keep up — and what you can do about it.

Relational Databases Lack Speed and Flexibility

Many of today's fraud-detection solutions are based on relational databases. These databases only provide a superficial understanding of unusual behaviors and patterns across customers, devices, and companies.

Finding connections requires manual and complex joins between tables, which reduces query performance. Traditional relational databases also lack the flexibility necessary to easily investigate hops or jumps that connect different entities. This can result in missing new cases of fraud associated with a bad entity.

Machine Learning Data Models Rely on Historical Data

Machine learning can address some challenges of more established solutions. It uses models trained to seek anomalous, potentially malicious activity. The more data the models have, the more they improve fraud detection. While this sophisticated approach can identify patterns humans can't spot, there are still issues. The datasets that feed into the models can make them brittle, so their algorithms can't adapt to updated or real-time data. These models also rely on historical data that includes past fraudulent behavior. If certain fraud examples aren't part of the dataset, models can't detect them.

Is there a solution that addresses these fraud-detection shortcomings? The answer is yes.

Graph Databases Enable More Accurate Fraud Detection

Graph databases natively store complex networks of transactions, accounts, people, and related data. With this approach, applications can detect fraud more accurately and in real time. Graph nodes represent different data points, such as phone numbers, IP addresses, and devices. "Relationships," also called "edges," connect these nodes through activities such as transactions and shared information.

Graph enhances fraud detection by enabling you to analyze connections between many different data points. It supports semantically rich queries at scale to reveal suspicious patterns.

For instance, running a projection to identify nodes with the highest number of connections helps flag potential fraud indicators. However, a high number of connections alone doesn't necessarily indicate fraud. Collecting additional contextual information and analysis can confirm suspicious activity.

Graph databases also allow you to score current data to represent similarities and risks. Scoring helps rank results based on their resemblance to known fraud communities. Node-connection analysis, other relevant factors, and scoring contribute to a more comprehensive, effective fraud-detection system.

With a graph database, you can use dynamic fraud-detection techniques that provide more accurate, faster results.

Enhance Detection With Graph Design Patterns

A design pattern is an abstract technique that solves a specific technical challenge. In this paper, we'll focus on six graph design patterns that simplify understanding, connecting, and uncovering activities and behaviors that could indicate fraud.

Pattern matching identifies and matches complex patterns that link entities.

Entity resolution identifies separate entities that are really the same.

Knowledge graph stores, organizes, and accesses interrelated data entities and their semantic relationships.

Machine learning uses data and algorithms to make predictions, such as whether an event relates to fraud.

Pathfinding and **anomaly detection** are well-defined strategies for finding paths and unusual patterns. Pathfinding helps uncover paths between entities and fraud intermediaries, while anomaly detection helps identify unusual behavior that needs investigation.

When you use graph design patterns to enhance existing fraud-detection solutions or build new ones, you strengthen the ability to find and stop threats. This reduces false positives, giving you an opportunity to develop new ways to find and approach fraud-indicative patterns. To understand how to use graph design patterns in your fraud applications, learn what each one can do and how they complement one another.

Pattern Matching

Implementing pattern matching on a graph database yields comprehensive results faster. Graph databases handle queries that cover complex relationships over large datasets, so they're well-suited for pattern matching across millions of transactions or user accounts. Pattern matching involves finding shapes and relationships in your data like trees, rings, paths, or chains.

The pattern-matching graph design pattern can uncover fraudulent behavior by displaying the connections and transactions between thousands of entities. First, you specify the pattern you're seeking in the graph: for example, a sequence of nodes connected by relationships that meet defined conditions for account takeovers, payment card fraud, and identity theft. You then use Cypher to express this pattern in a query that finds all matching instances.

Deciphering Cypher

Neo4j created Cypher, a graph query language, to interact with graph databases. When someone writes a query in Cypher, they code a pattern for what they're searching. Developers and data scientists can construct expressive and efficient queries that create, read, update, or delete (CRUD) the data in their graph.

Cypher represents patterns with a syntax similar to ASCII art: (nodes)-[:ARE_CONNECTED_TO]->(otherNodes)

using rounded brackets for (nodes), and -[:ARROWS]-> for relationships.

Your Fraud-Detection Applications — Amplified

You can use pattern matching to add more depth to your fraud-detection solutions. Here are a few examples of capabilities the graph design pattern supports:

Unusual pattern detection

Pattern matching reveals transactions or activities that significantly deviate from a user's usual behavior — an indicator of potential fraud.

Ring detection

You can use pattern matching to find shapes in data and relationships that suggest a fraud ring, such as circular transaction patterns.
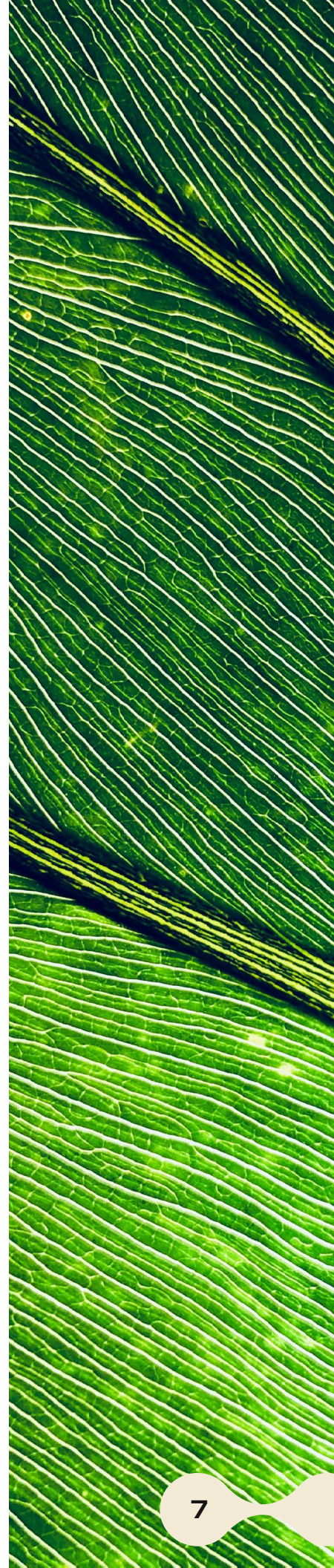
Hidden relationship discovery

With pattern matching, you can find hidden relationships between seemingly unrelated accounts or entities that may indicate fraudulent activity.

Pattern-Matching Use Case: Credit Card Fraud

Consider this hypothetical scenario. A financial institution uses a graph database and the pattern-matching graph design pattern in a solution that monitors and analyzes credit card transactions. Each transaction is a node with edges indicating the relationships between transactions, account holders, merchants, and geographical locations. Node and edge properties store detailed information about each transaction, such as amount, timestamp, and merchant category.

Using Cypher queries, the system searches for predefined patterns that signal fraud, such as constant changes to home and IP addresses or rapid departures from typical credit-card behavior.

Entity Resolution

Entity resolution is a popular starting point for graph databases. It enhances the graph's data quality, making it easier to see, understand, and match patterns and relationships. You can use entity resolution to merge user profiles, accounts, and other entities that appear different but are actually the same. By removing redundant and irrelevant entities, you can find fakes and trace them back to the fraudster.

Exposing Fakes and Fraudsters

With the entity-resolution design pattern, you aggregate and reconcile data from multiple sources to create comprehensive profiles for individuals, organizations, and other entities. These profiles then become part of your application. For example, when you use the entity-resolution design pattern with Neo4j Graph Database, you unify all transactions, interactions, and relationships associated with a single profile. By resolving these entities, you can more effectively map a network of suspicious activities and connections.

Entity-Resolution Use Case: When a Claim Isn't a Claim

A global insurance company developed a fraud detection system that uses Neo4j Graph Database to analyze claims, policyholder information, and network connections. Its system uses entity resolution to identify multiple policyholder nodes with slight variations in names or addresses but with overlapping contact information or claims history. The system marks these nodes as individuals who might be trying to defraud the company by filing claims under different identities. (See Figure 1.)
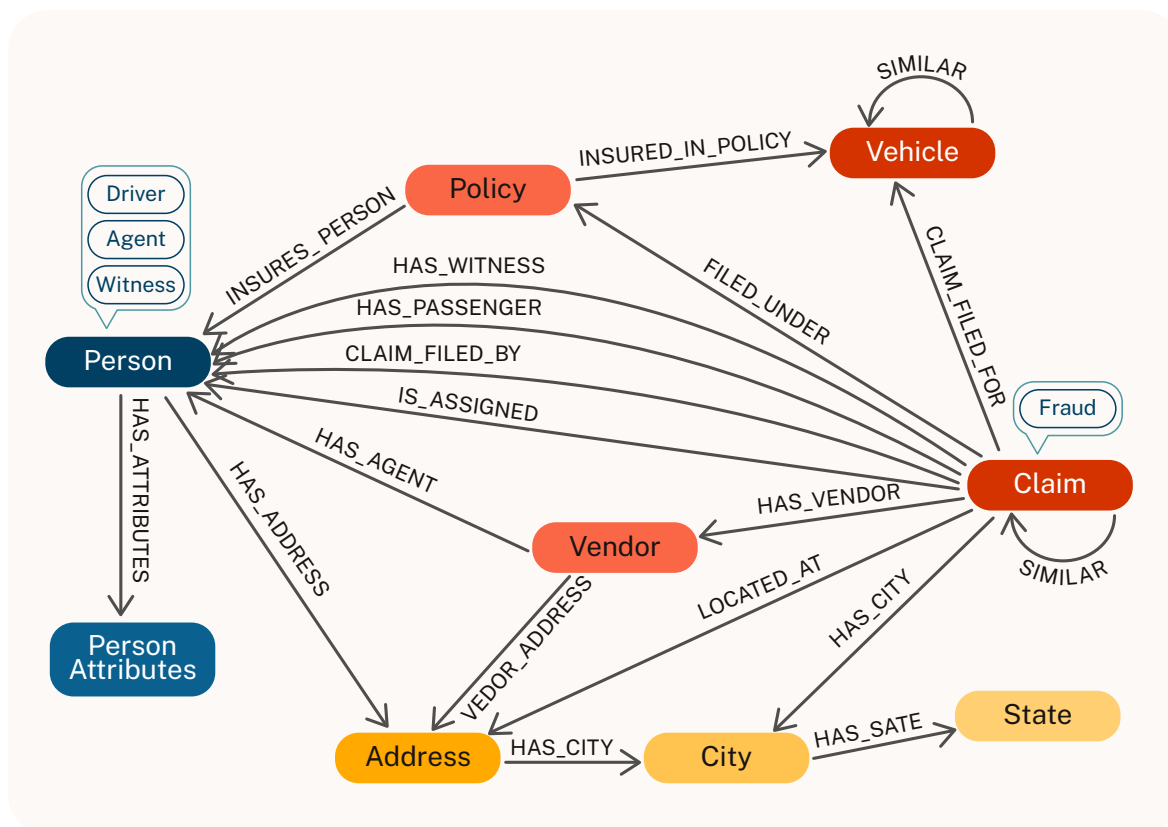

Figure 1: Entity resolution

In another scenario, the system reveals that common entities connect seemingly unrelated claims with the same policy numbers and types. The insurance company constructs a clear network of incidents that shows how different claims are interrelated. For example, it could uncover a pattern where a particular group of individuals is involved in a series of accidents under suspicious circumstances, suggesting staged incidents.

Knowledge Graph

Knowledge graph is a design pattern used to store, organize, and access interrelated data entities and semantic relationships between different pieces of information. It enables a more sophisticated understanding of data as well as better reasoning across the data. Knowledge graphs incorporate organizing principles that include your institutional knowledge and domain expertise. The model describes every aspect of your business which you can use to better detect and investigate fraud. You start building a knowledge graph the first day you begin storing and organizing data in a graph database.

Identifying Sophisticated Fraud: It's Layered

Organizing your data into a knowledge graph helps you view organizational and data layers within a single data model. You can traverse the data layer and uncover new relationships by examining the organizational layer. By centralizing the organizing principles in your knowledge graph, you can reuse them across multiple applications and services instead of implementing them separately.

With knowledge graphs storing and organizing data and relationships, applications can better identify sophisticated fraud schemes. By providing context through explicit relationships, the knowledge-graph design pattern enriches data. This is crucial for fraud-detection solutions because it considers a broader network of interactions.

Enriched data simplifies identifying differences between usual and suspicious behaviors in fraud-detection applications. Building knowledge graphs using Neo4j Graph Database enables real-time performance, which helps you prevent fraud.

Knowledge-Graph Use Case: Don't Give Just Anyone Credit

A fraud-detection solution using the knowledge-graph design pattern can more easily identify credit fraud. The knowledge-graph design pattern aggregates and organizes complex data into a comprehensive view of a customer's behavior and history. (See Figure 2.)
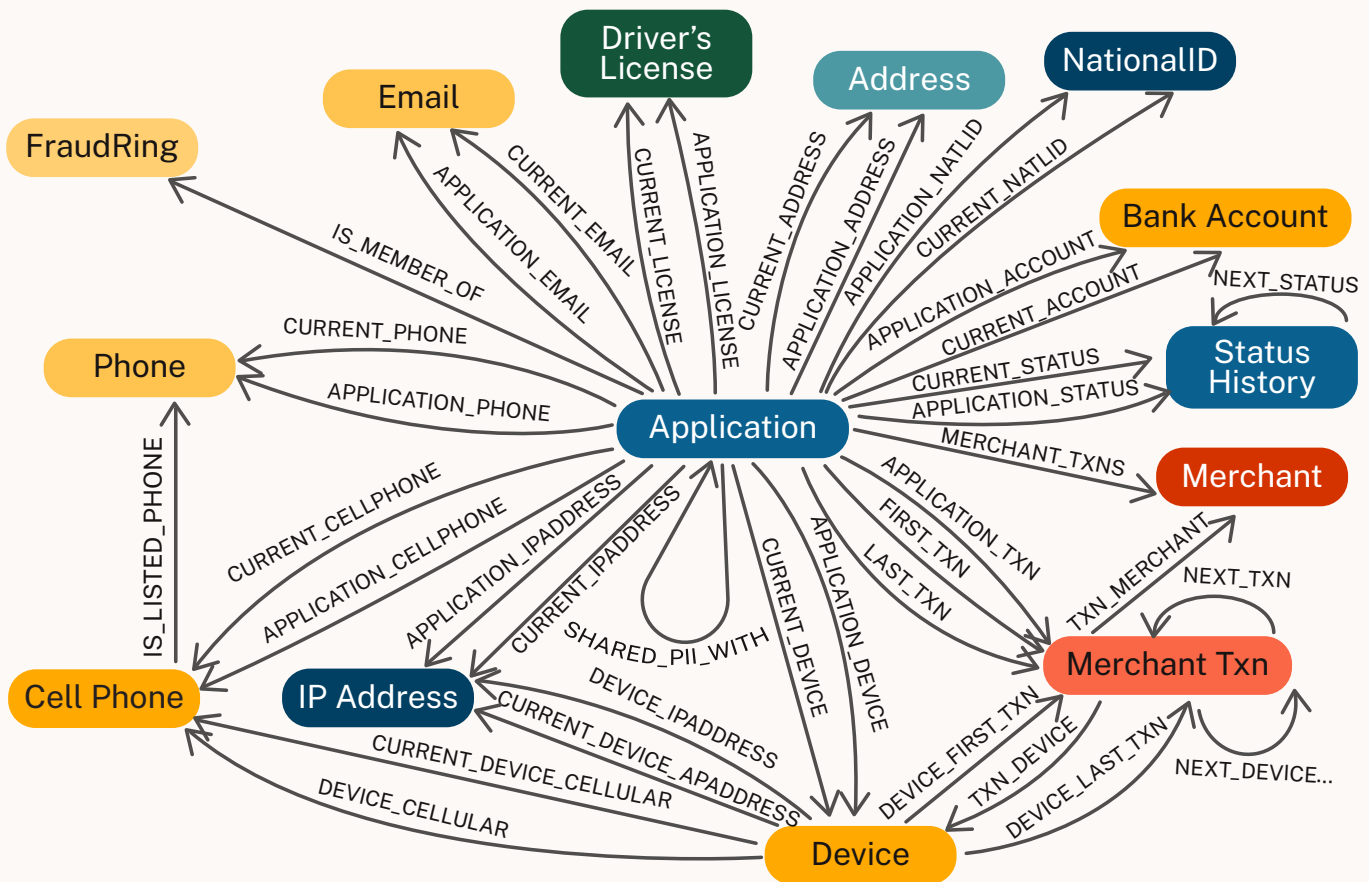

Figure 2: Knowledge graph

With this comprehensive view, the credit issuer can evaluate transactions beyond individual instances in a web of related activities. This view identifies sophisticated fraud schemes easier. For example, it can reveal multiple unrelated accounts with identical deposits, or a person with a low fraud risk score who has connections to a community with a high-risk score. The credit issuer can flag these credit applications for further investigation. As you start identifying fraud, you can incorporate that information into the organizing principles comprising your knowledge graph. Then you can use that metadata to identify the same pattern in other areas.

Machine Learning

You may already use machine learning in your fraud-detection pipeline. However, if you're not using a graph database with your machine learning, you're missing an opportunity to bring valuable relationships into your models.

By combining machine learning with [Neo4j Graph Database](), you gain access to relationships and signals encoded in the data that you wouldn't otherwise get. This helps you identify new features you can add to your existing fraud-detection models. The machine-learning design pattern finds contextual information in the patterns and the signals encoded in those relationships. It contributes to feature engineering by discovering signals that improve model accuracy, drive better results, and catch fraud before it occurs.

Working in Real Time: Predictions and Pipelines That Scale and Flow

Machine-learning models can use the real-time data flowing into Neo4j Graph Database to make immediate fraud predictions or decisions. As the amount of data grows, the machine-learning graph design pattern can scale to accommodate increasing data volume and complexity, ensuring that machine learning models keep operating efficiently and effectively without drifting.

The design pattern also enables Graph Database to integrate with various machine-learning frameworks and platforms. Data scientists can build, train, and deploy models that use graph data to create end-to-end machine-learning pipelines for fraud detection.

Machine-Learning Use Case: Fintech Fraud Finding x3

A fintech company wanted to accelerate its fraud detection, so it built a solution on a Neo4j graph database. This solution uses a machine-learning design pattern to identify potential fraud patterns from hundreds of transactions in a second. It replaced a traditional fraud-detection package, and the difference is stark. The fintech's customers report a [200% increase in fraud detection]() — three times higher than the traditional package — with no change in false positives.

Pathfinding and Anomaly Detection

Pathfinding identifies patterns connecting two entities, while anomaly detection helps you discover the unexpected. Both graph design patterns use well-defined data science algorithms to dive deeper into data.

Getting From Point to Point (and the Hidden Middle)

Pathfinding is the process of finding one or more routes from a starting point to a destination point. (See Figure 3.) It uses various algorithms, each with its own strengths and use cases. For example, Dijkstra's algorithm is well known for finding the shortest path in graphs without negative edge weights. Meanwhile, the computer science and gaming industries use the A* algorithm for its efficiency and ability to incorporate heuristic estimates that guide searches.
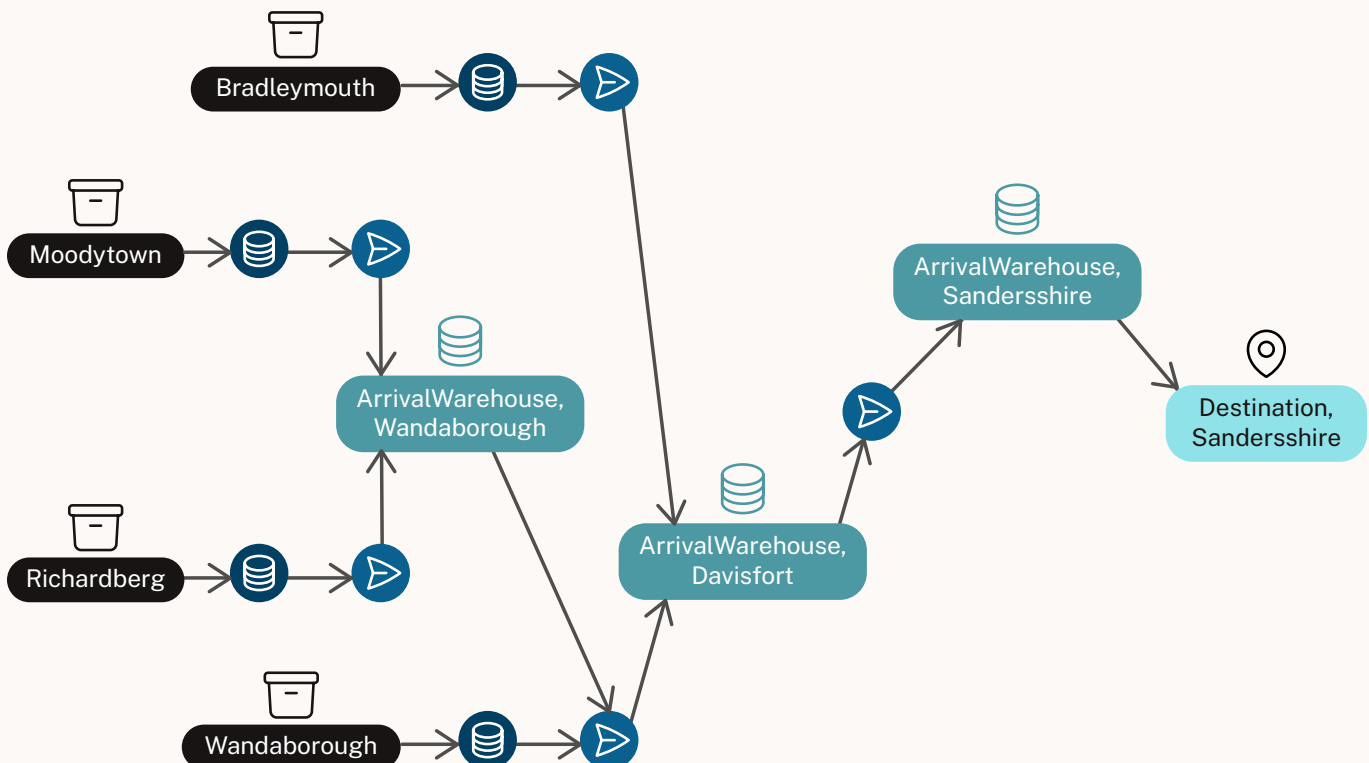

Figure 3: Shortest path

The goal of pathfinding is to traverse paths with Cypher by moving from start to end. When you're working with huge datasets for fraud detection, the pathfinding design pattern can show you hidden bad actors in the middle. It reveals indirect relationships between entities, such as two individuals who seemingly have no direct link but are connected through a series of intermediaries. This can be crucial in detecting fraud rings, where individuals or entities collaborate to conduct fraudulent activities.

The pathfinding design pattern can also predict the likelihood of a future connection between two nodes based on the existing network structure. This helps you identify potential fraudulent connections before they fully materialize.

Detecting the Unexpected

Anomaly detection identifies unusual patterns, behaviors, or outliers in a dataset that do not conform to expected behavior. Sometimes anomaly detection starts when you find a pattern that doesn't make sense and you want to investigate it. Alternatively, it starts because you're seeking anomaly types so you can write code or queries to detect them.

The anomaly detection graph design pattern, together with Neo4j Graph Database and Cypher queries, can uncover the unexpected. This could involve a sudden increase in connected accounts, a surge in transactions, or a huge spike in traffic from particular IP addresses. These all potentially indicate fraudulent activities. Using Cypher, you can define what typical behavior looks like, and then query the graph to find deviations. The anomaly detection graph design pattern can quickly identify these deviations by analyzing graph structure and node relationships. It then displays the results in a visualization tool like Neo4j Bloom.

Pathfinding and Anomaly Detection Use Case: Video-Gaming the System

An online video-gaming company uses a system with a graph database to track typical player behavior. The system uses the anomaly detection graph design pattern to monitor player logins and in-game activities. It identifies unusual activities, like logging in at odd hours from different geographical locations or using new devices, which could suggest unauthorized account access.

If a player suddenly engages in atypical in-game spending, transfers valuable in-game items to unknown players, or alters sensitive account settings, the system flags those activities. Pathfinding also identifies complex patterns that can indicate an account takeover, such as a sudden change in the player's network of interactions, rapid depletion of in-game assets, or abrupt changes in player performance.

Graph Design Patterns Working Together

Graph design patterns for detecting fraud don't operate in a vacuum. For example, for insurance fraud, pattern matching, knowledge graph, entity resolution, and anomaly detection can incorporate risk scoring into multiple scenarios. They automate detection and find complex, multi-hop fraud patterns faster. Together and in tandem, these graph design patterns provide powerful solutions that identify the sophisticated tricks that fraudsters use. How does this work? A look at connected fraud will give you an idea.
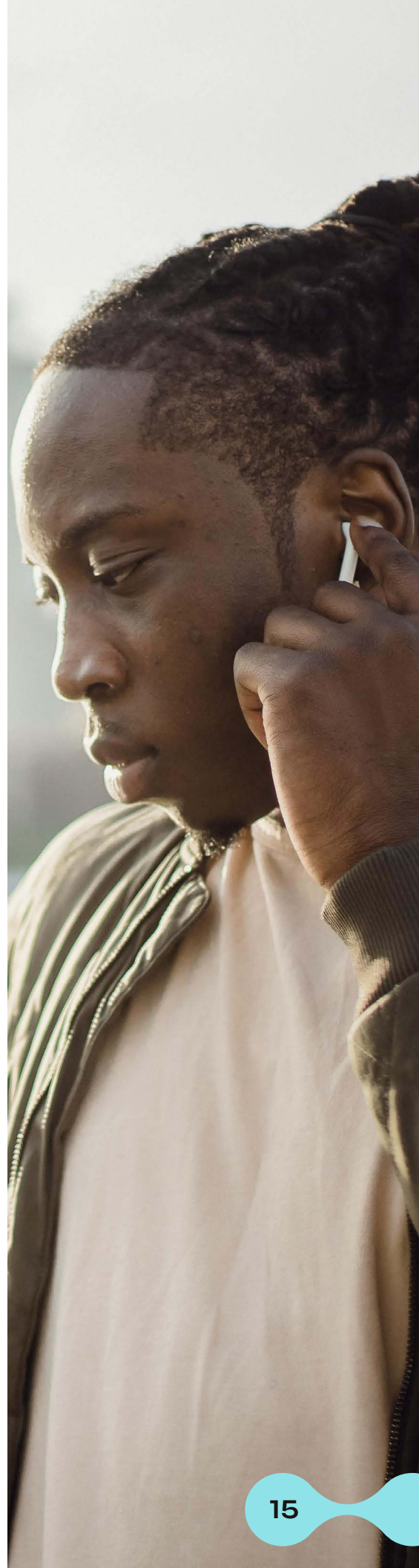
Stop Connected Fraud: Crawl, Walk, and Run

When groups or entities from different industries collude to defraud organizations, they're committing connected fraud. For example, a person files a false insurance claim, and a physician and auto body shop substantiate it. The pattern for connected fraud can be extensive, with complex interlinks, which contributes to a low detection rate. However, if you add graph to your applications and use design patterns in "crawl, walk, and run" stages, you can increase your detection rate.

Crawl

When you first build your graph, start by using the entity-resolution graph design pattern to familiarize yourself with the entities and their relationships. As these come into focus, you'll naturally have questions that simple pattern-matching queries can help you answer. Log your discoveries in your knowledge graph to start capturing the unique semantics and patterns that make up your business.
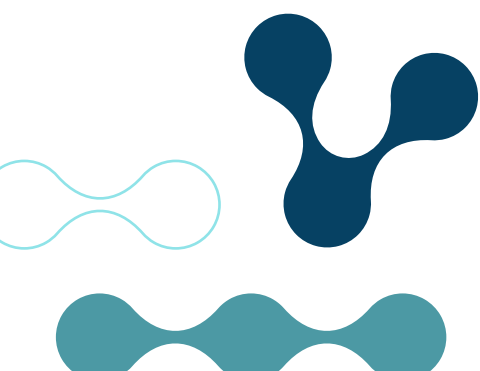
Walk

Once you're oriented in your graph data, use Cypher to expand your search for connected fraud patterns. Assign scores that signal risk levels from related entities. Combined with the rules-based or machine-learning models you already run in other systems, those scores will validate your primary model's findings. Pathfinding can also help you better understand connections between fraudulent actors and different fraud rings.

Run

As you develop your skills with graph, Cypher, and graph data science, you can evolve and expand rules, graph design patterns, and models to identify connected fraud. For example, use the machine-learning design pattern to uncover new features that feed into your graph. Apply anomaly detection to your knowledge graph to surface new patterns. Layering these techniques together can raise your connected fraud-detection baseline for immediate results.

Whether you crawl, walk, or run to stop fraud, Neo4j Graph Database is best suited for using the six graph design patterns to improve your results.
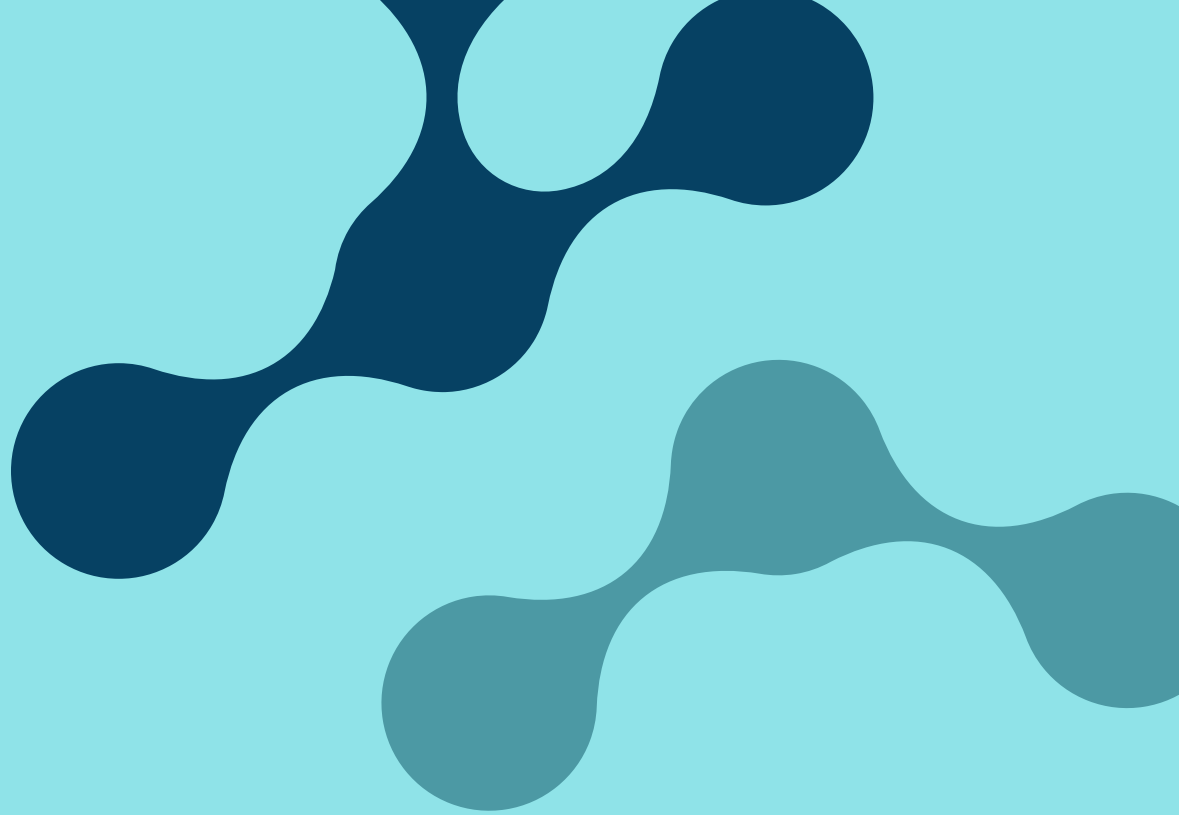
Neo4j: Your Enterprise-Strength Graph Database for Uncovering Complex Fraud Patterns

Use the structure of your connected data to reveal new ways of detecting and stopping fraud, even as your data grows. Neo4j Graph Database excels at the six graph design patterns, significantly enhancing your fraud-detection solutions and reducing false positives. By integrating Neo4j Graph Database into your existing IT architecture, you can handle higher data volumes, more users, and concurrent transactions without creating intersection tables or joins. The Neo4j full graph stack delivers powerful native graph storage, data science, advanced analytics, and visualization with enterprise-grade security controls, scalable architecture, and ACID compliance.

Plus, Neo4j's data leaders comprise a vibrant, open-source community across all industries. Stop fighting fraud alone. Join more than 250,000 developers, data scientists, and architects across hundreds of Fortune 500 companies, government agencies, and NGOs to share best practices with graph leaders across the globe.

AWS and Neo4j: Tackling Fraud Detection Together

Amazon Web Services (AWS) and Neo4j combine the best of both worlds—cloud and data infrastructure plus all the benefits of graph database—to help developers create new, innovative solutions for fraud detection. Graph design patterns and techniques improve fraud detection, and Neo4j Graph Database integrates with AWS data ingestion and import services so it can efficiently receive data from various AWS sources.

Graph databases also complement the AWS tools and services developers use to build applications, enhancing the application development process on AWS and reducing development efforts. Machine-learning tools add intelligent features such as predictive analytics or recommendations. The Neo4j/AWS architecture (See Figure 4.) is an example of how you can configure your AWS ecosystem to supports fraud detection graph use cases:
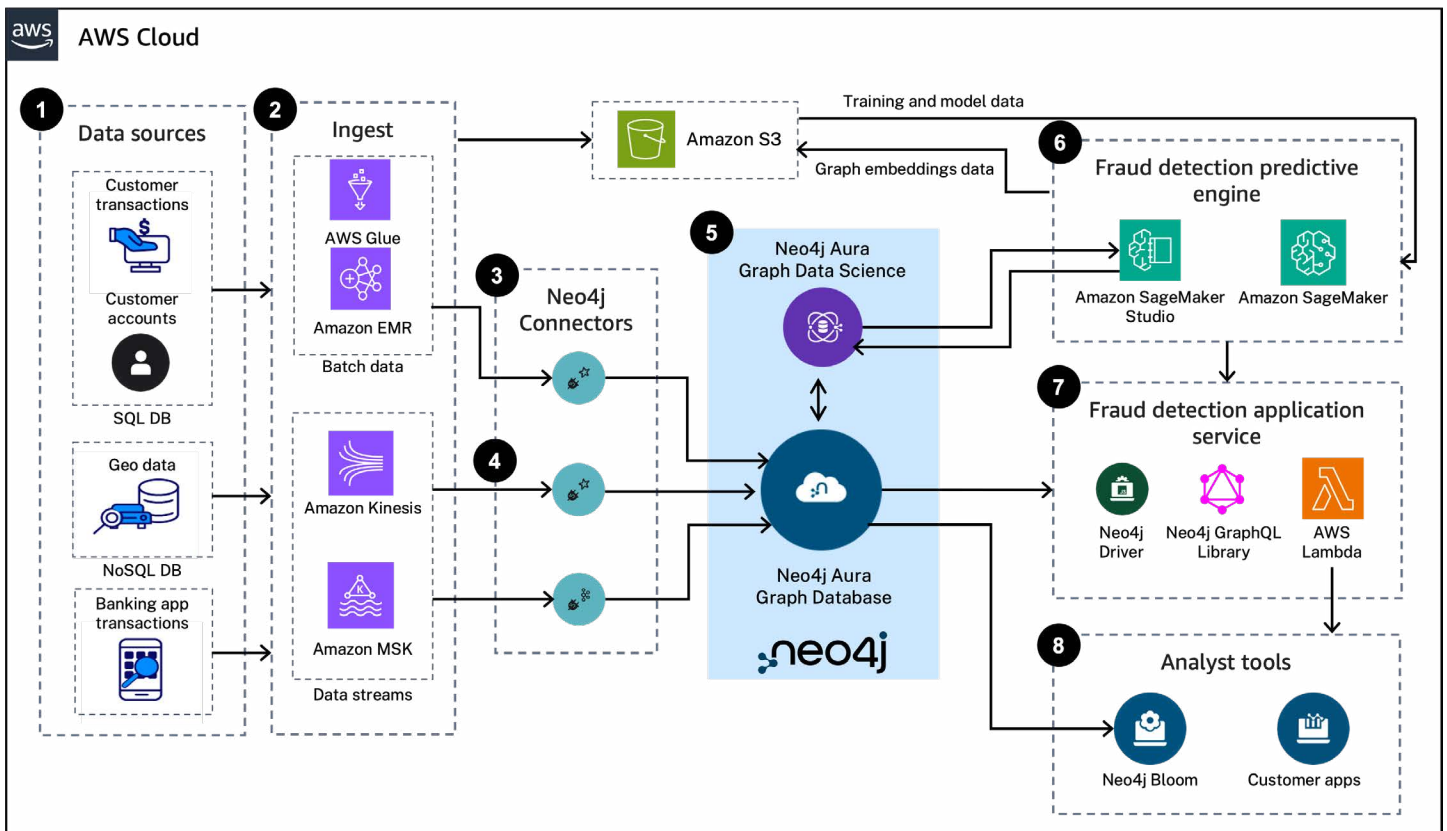

Figure 4: Neo4j and AWS fraud detection sample architecture

Developers running ML models on Amazon SageMaker can improve the accuracy of their models quickly by using graph algorithms to extract hidden insights and create new features. Graph pattern matching can further enhance predictive results by adding weights and explainability.

Similarly, by integrating Neo4j AuraDB with Amazon Bedrock organizations can connect structured and unstructured data sources, enhance Retrieval-Augmented Generation (RAG) workflows, and build knowledge graphs. These knowledge graphs can be used in GraphRAG architectures to create contextual and explainable GenAI tools that provide better grounded, more accurate results than alternative approaches.
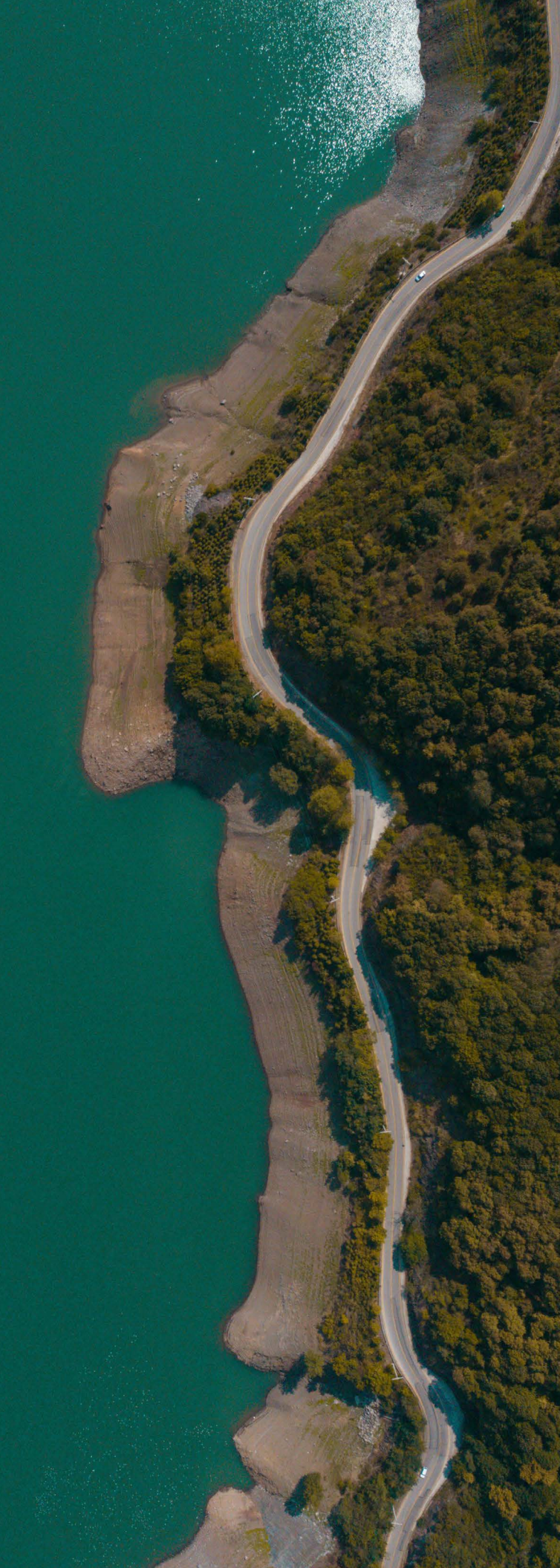
A Better Approach to Fraud Detection With Your Current Solutions

Fraud continues to evolve in sophistication and scale, costing organizations billions annually. Traditional fraud detection methods, built on relational databases and historical machine learning models, struggle to keep pace with increasingly complex schemes. Graph databases are a powerful tool for augmenting fraud detection solutions and accelerating the fraud detection pipeline.

The six graph design patterns covered in this paper provide a framework for using graph databases to uncover sophisticated fraud schemes. When used together, these enable organizations to identify suspicious patterns, merge duplicate entities, understand semantic relationships, improve model accuracy, trace hidden connections, and detect anomalous behaviors.

The result? The applications your organization builds will be better equipped to identify and protect against fraud.
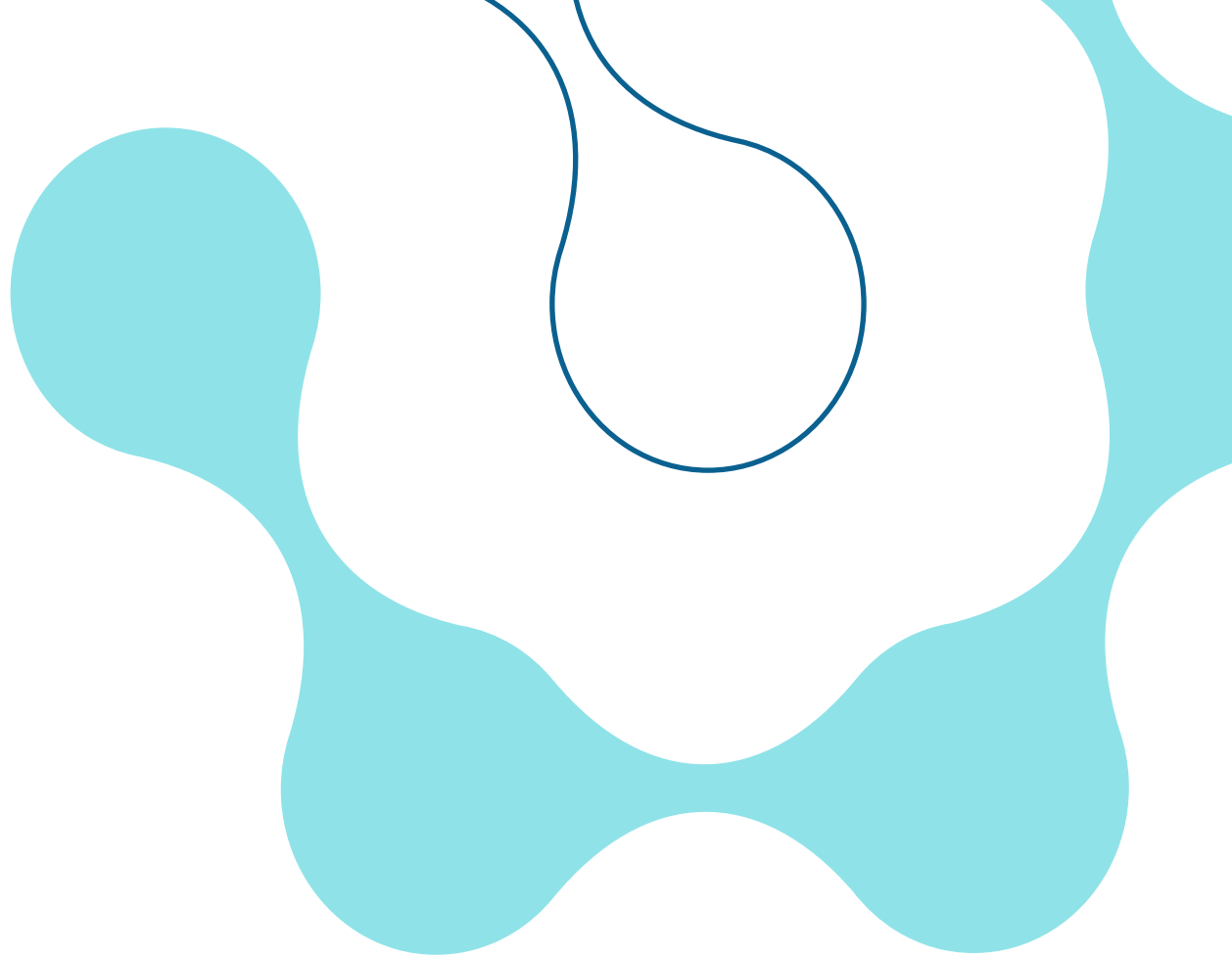
How to Get Started

Neo4j uncovers hidden relationships and patterns across billions of data connections deeply, easily, and quickly, making graph databases an ideal choice for developing robust, scalable, and effective fraud detection or for augmenting existing ones.

Learn More

available in
aws marketplace