graphsummit

ASIA PACIFIC

neo4j

Building a Knowledge Centric Fraud Detection Graph using Neo4j and Python

Introduction

Neo4j Graph Data Platform

Neo4j Graph Data Platform

Native Graph Database
The core component of Neo4j platform

Graph Query Language
Cypher and GQL as the lingua franca for graphs

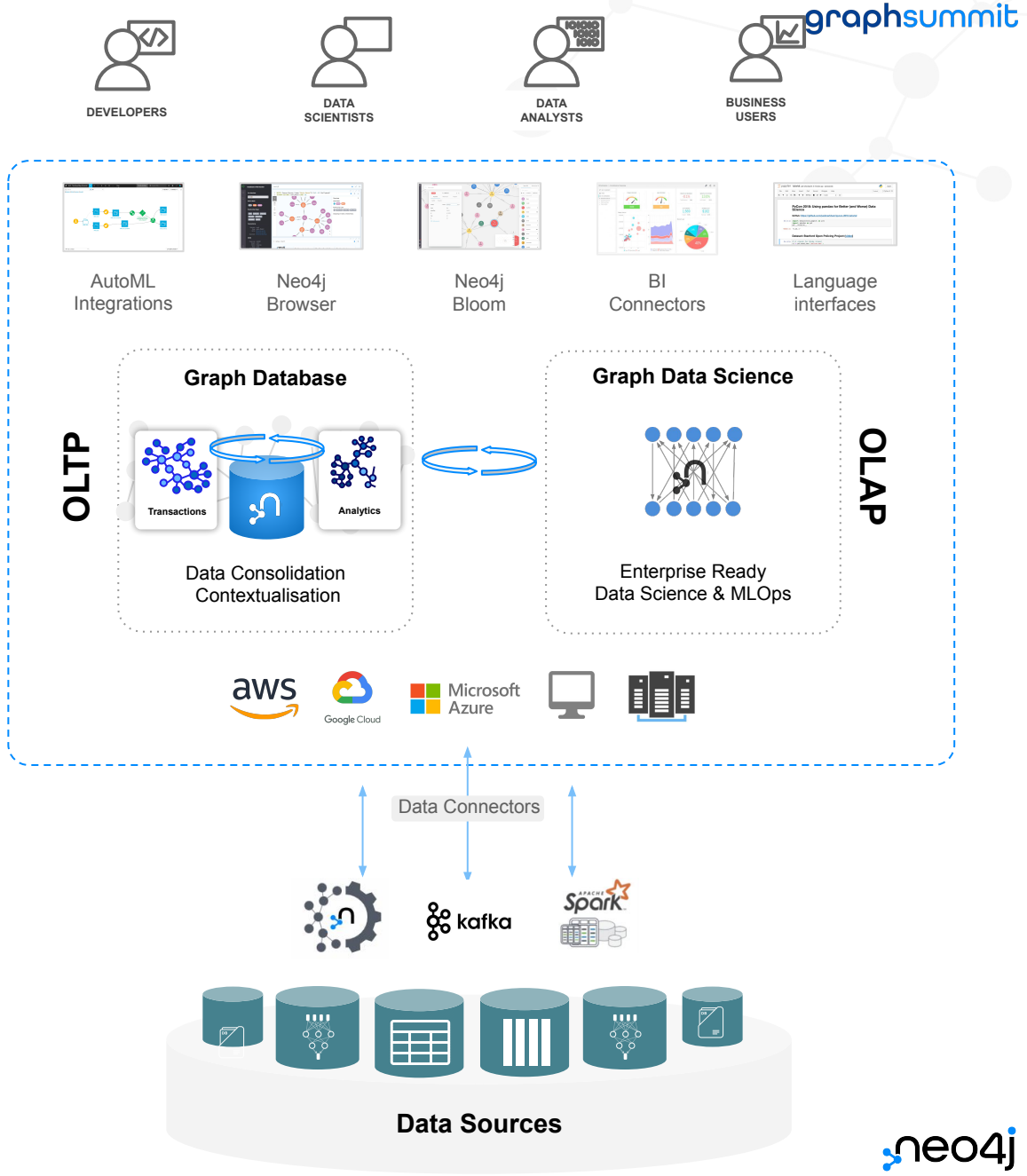
Data Science and Analytics
Tools, algorithms, and Integrated ML framework

Application Development Tools & Frameworks
Tools and APIs for rapid prototyping and development

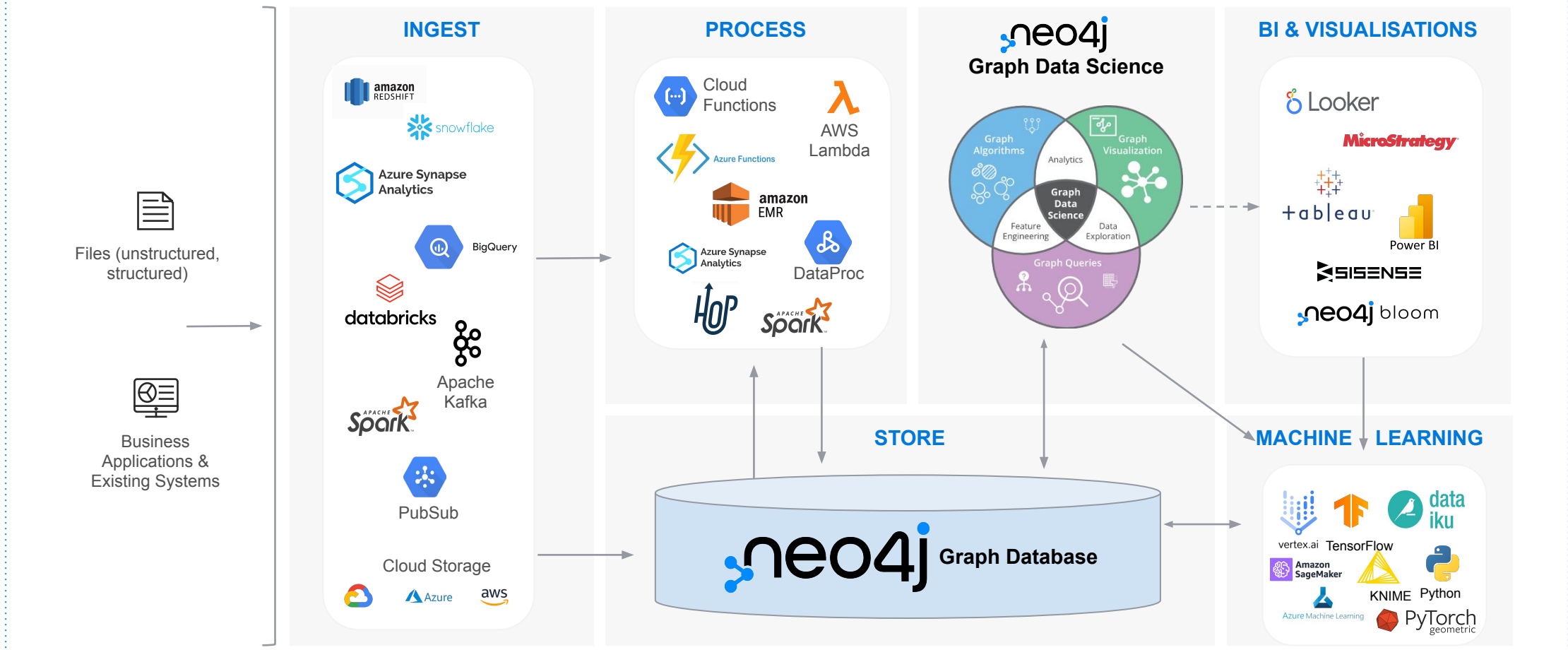
Discovery & Visualisation
Low-code querying, data modeling and exploration tools

Ecosystem & Integrations
Rich set of connectors to plug into existing data ecosystems

Runs Anywhere
Run by yourself or as DBaaS by Neo4j, in the cloud or on premises

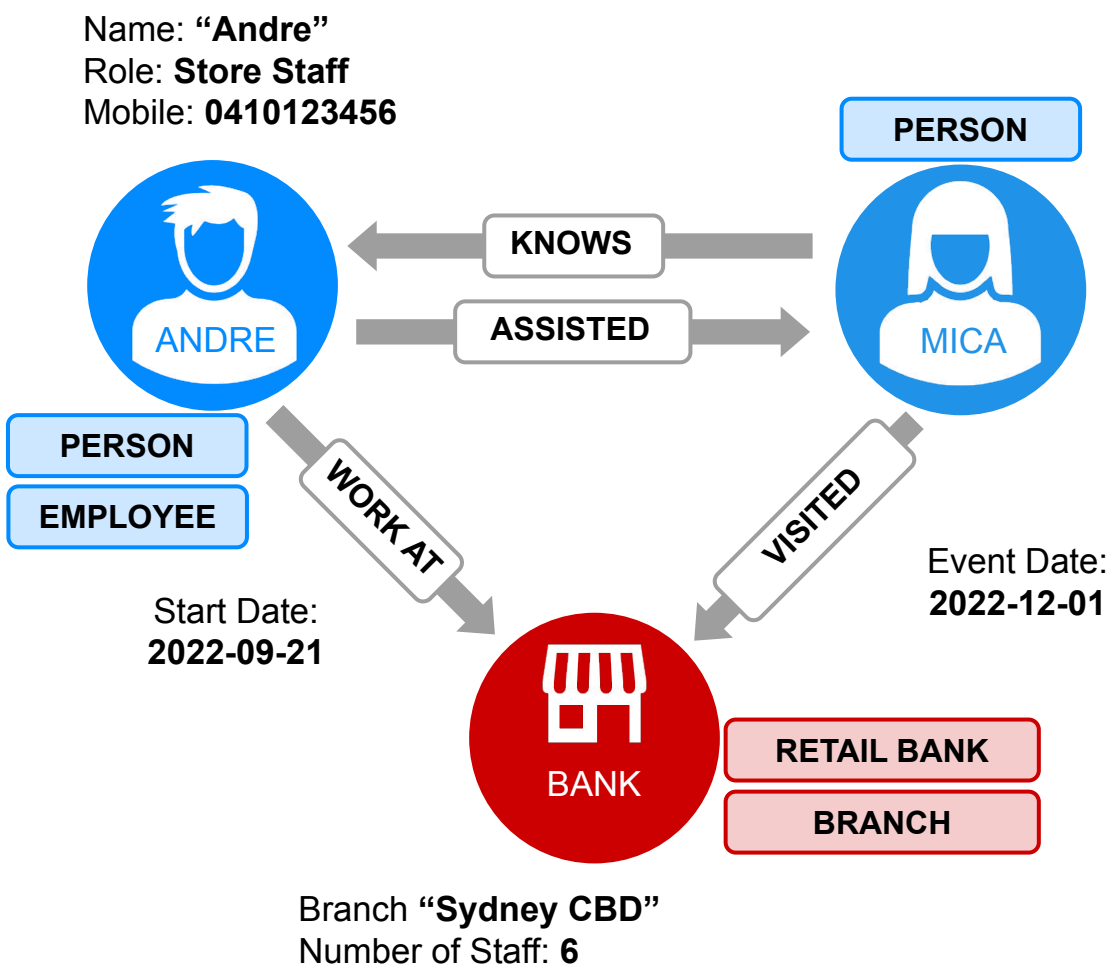


Neo4j Architecture and Deployment



Graph Fundamentals

Graph Building Blocks



Node

Represents an entity in the graph

Relationship

Connect nodes to each other

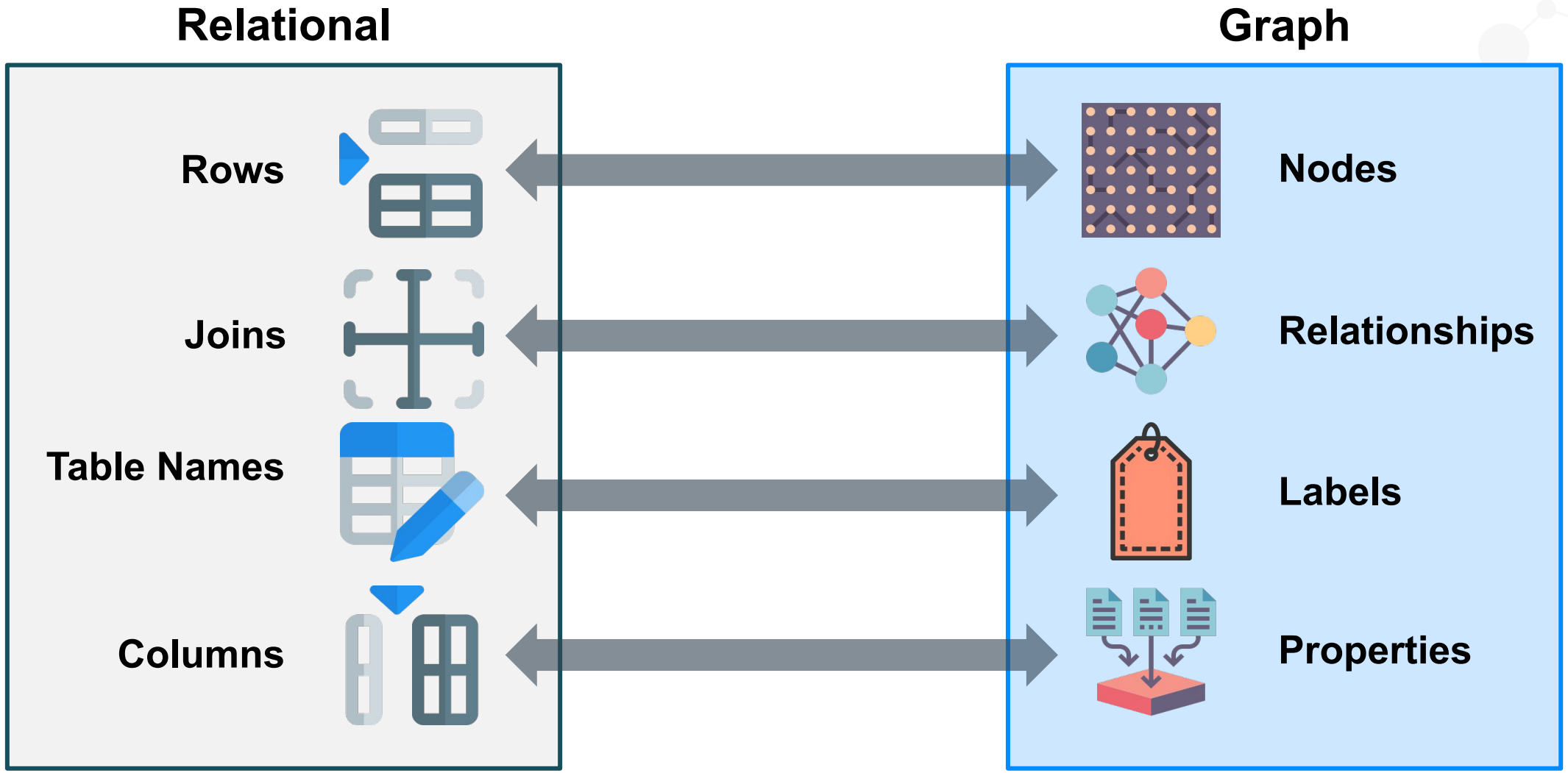
Label

Grouping of similar nodes

Property

Describes a node or relationship: e.g. name, age, address

Conceptual Mapping Relational → Graph

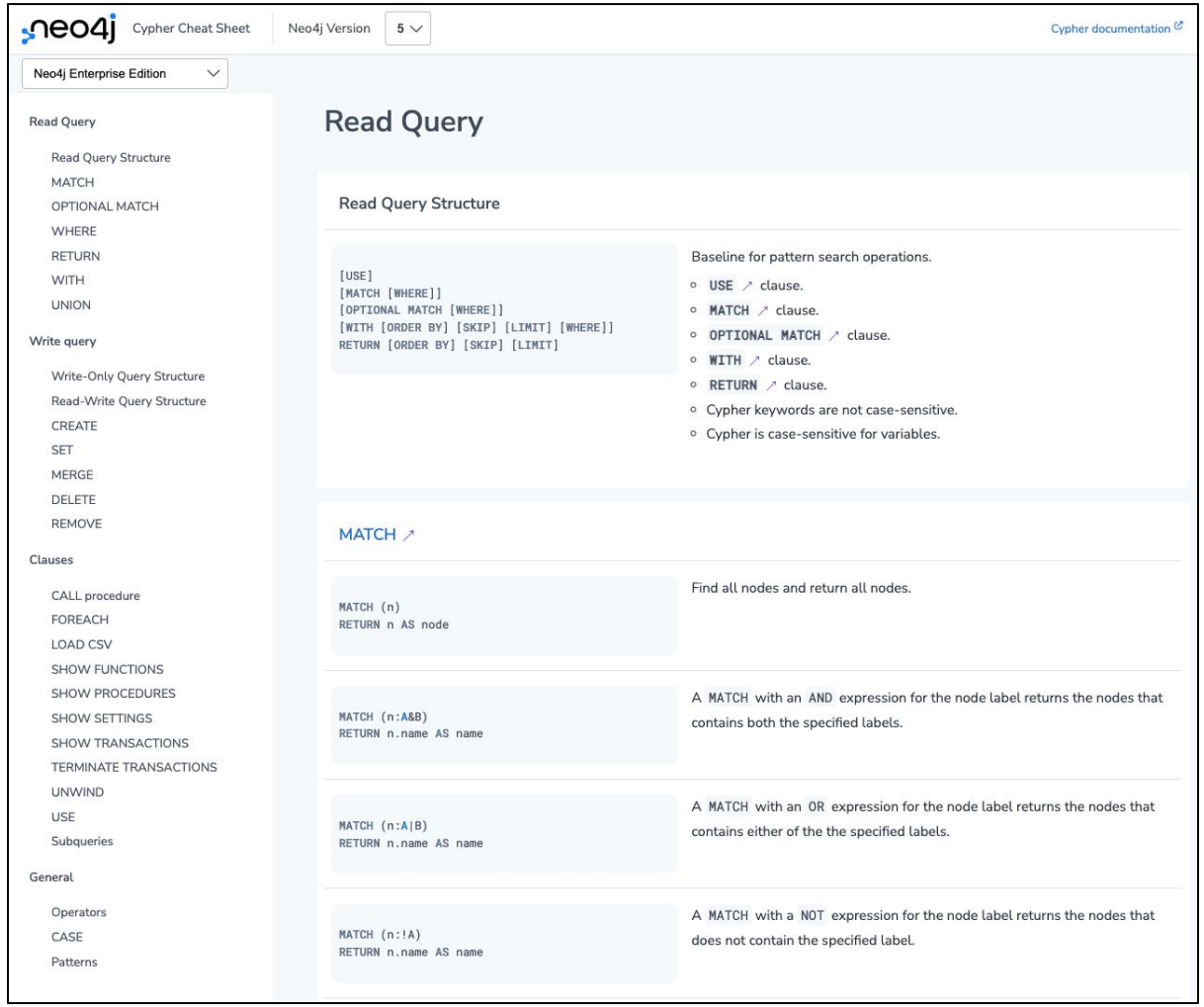


Graph (Cypher) Query Language
Fundamentals

Cypher Cheat Sheet

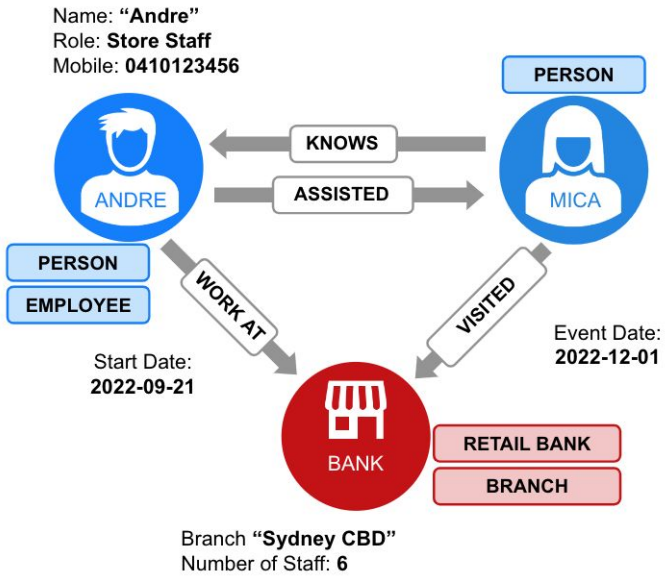
graphsummit

Cypher Cheat Sheet Link



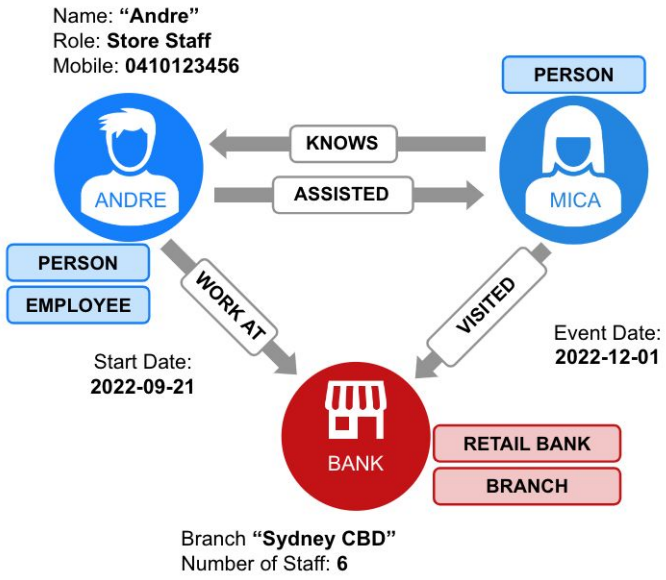
neo4j

Basic Cypher (GQL) Syntax (1/3)




```
   MATCH - nodes and relationship graph pattern matching (GPM)
   WHERE - filter node or relationship properties, labels or GPM
  RETURN - return graph path, nodes, relationship properties or all ("*")
ORDER BY - order set of properties in ascending or descending order
   LIMIT - limit the number of nodes and relationships
```

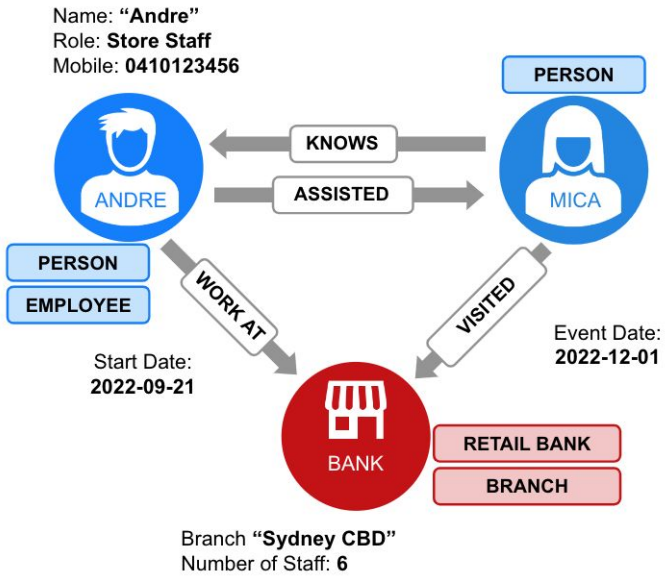
Basic Cypher (GQL) Syntax (2/3)



Node Pattern | Relationship Pattern | Node Pattern

```
MATCH (var:label expression)-[var:type expression]->(var:label expression)
WHERE where clause //This is a comment
  AND logical operator - NOT, OR and XOR can be used as well
RETURN properties or "*"
ORDER BY properties or "*"
LIMIT number
```

Basic Cypher (GQL) Syntax (3/3)

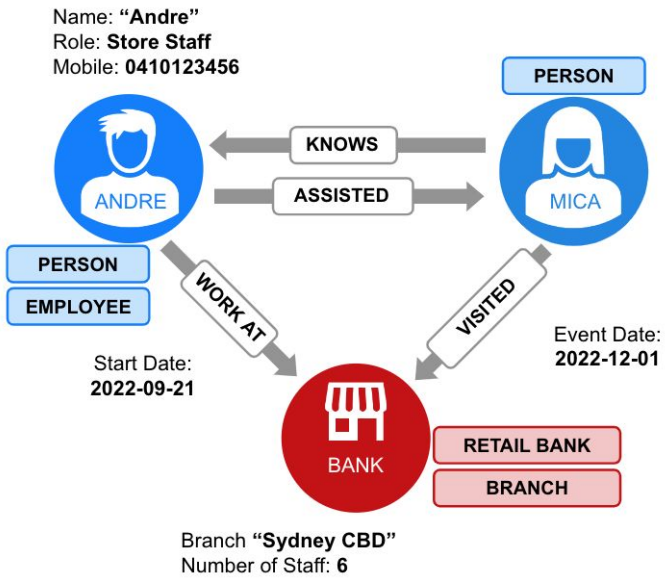


Node Pattern | Relationship Pattern | Node Pattern

```
MATCH (p:Person)-[r:WORK_AT]->(b:Branch)
WHERE p.name = "Andre" // This is a comment
  AND r.startDate = '2022-09-21'
RETURN *
ORDER BY b.branch
LIMIT 1
```

WITH Clause Syntax

The WITH clause is similar to RETURN. It allows query parts to be chained together, piping the results from one to be used as starting points or criteria in the next.




```
    MATCH (p:Person)-[r:WORK_AT]->(b:Branch)

// Only variables declared in the WITH can be referenced later in the query
     WITH b, COUNT(p) AS numEmployees

    WHERE 1 < numEmployees < 4
       OR b.branch STARTS WITH "Sydney"

  RETURN *
```

UNWIND Clause Syntax

graphsummit

The UNWIND clause **expands a list into a sequence of rows**. Multiple UNWIND clauses can be chained to unwind nested list elements.

Inputs

id	roles
1	[“READ”, “WRITE”, “DELETE”]
2	[“READ”]
3	[“READ”, “WRITE”]
4	[“READ”]
5	[“READ”, “WRITE”, “DELETE”]
6	[“READ”, “WRITE”]

UNWIND Syntax

```
  WITH $Inputs AS input

// Expand list of roles into sequence of role.
UNWIND input.roles AS role

RETURN input.id AS id
      ,input.role AS role
```

Output

id	role
1	READ
1	WRITE
1	DELETE
2	READ
3	READ
...	...

neo4j

CONSTRAINT Syntax

graphsummit

Ensures that node and relationship property value are **unique for specific node label or relationship type**. It **automatically adds an index** on that property.

CONSTRAINT Syntax

```
// Unique node property constraints.
CREATE CONSTRAINT constraint_name FOR (p:Client) REQUIRE p.id IS UNIQUE;

// Unique relationship property constraints.
CREATE CONSTRAINT constraint_name FOR ()-[r:LIKED]-() REQUIRE r.date IS UNIQUE;

// Node key constraints.
CREATE CONSTRAINT constraint_name FOR (p:Client) REQUIRE (p.id, p.name) IS NODE KEY;
```


neo4j

LOAD CSV Syntax

graphsummit

To import data from a CSV file into Neo4j, you can use **LOAD CSV to get the data into your query**. Then, you write it to your database using the normal updating clauses of Cypher.

LOAD CSV Syntax

```
// Load CSV with headers based on the file URL.
LOAD CSV WITH HEADERS

// Each line the csv file will be assigned in the variable “row”.
FROM "https://url/clients.csv" AS row
WITH row

// CSV fields can be access using the variable.<field name> e.g. row.ID
MERGE (c:Client { id: row.ID })
  SET c.name = row.NAME
     ,c.isFraud = toBoolean(row.ISFRAUD)
```


neo4j

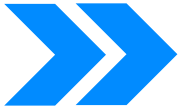
MERGE Clause Syntax (1/3)

graphsummit

The MERGE clause **create (if it doesn't exist) or update nodes, relationships or properties** based on node key (primary key). It is similar to SQL UPSERT clause.

MERGE Syntax

```
MERGE (a:Actor {name: 'Michael Caine'})

// Create 'Michael Caine' node and set born
// property if it doesn't exists.
// Else, just set the born property.
    SET a.born = 1933

RETURN a
```



neo4j

MERGE Clause Syntax (2/3)

graphsummit

The MERGE clause allows you to **specify CREATE or MATCH behavior when merging.**

MERGE Syntax

```
MERGE (a:Person {name: 'Sir Michael Caine'})

// Properties to be set when a new node is
// created.
ON CREATE SET a.born = 1934
              ,a.birthPlace = ‘UK’

// Properties to be set when a existing node
// matches.
ON MATCH SET a.birthPlace = ‘UK2’

RETURN a
```

Output when the node already exists

Table / Text

	a.name	a.born	a.birthPlace
1	"Sir Michael Caine"	1934	"UK2"

neo4j

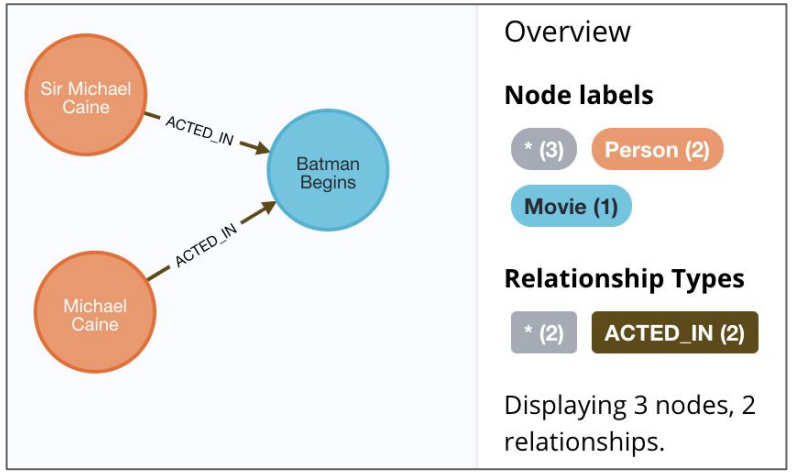
MERGE Clause Syntax (3/3)

The MERGE clause **applies to relationships** as well.

MERGE Syntax

```
        MATCH (p:Person), (m:Movie)
        WHERE m.title = 'Batman Begins'
          AND p.name ENDS WITH 'Caine'
        MERGE (p)-[r:ACTED_IN]->(m)
ON CREATE SET r.role = 'New'
 ON MATCH SET r.role = 'Existing'
       RETURN p, m
```

Output when the relationship doesn't exists



Neo4j APOC Library

graphsummit

APOC stands for Awesome Procedures on Cypher. It is standard utility library for common procedures and functions.

APOC Syntax

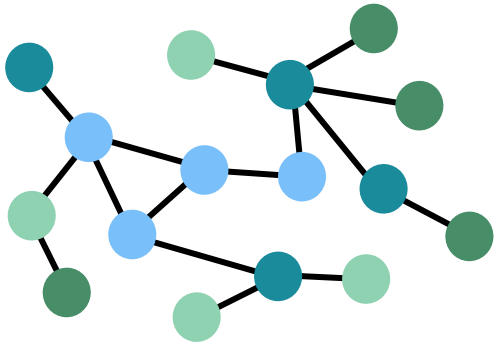
```
// General APOC Syntax
CALL <package>.<subpackage>.<procedure>(<argument1>,<argument2>,...);

// Returns the information stored in the transactional database statistics
 CALL apoc.meta.stats()
YIELD nodeCount, labels;

// Creates a list of lists of adjacent elements in a list, skipping the last item.
// Returns [[1,2],[2,3]].
CALL apoc.coll.pairsMin([1,2,3]) YIELD value;
```


neo4j

Neo4j Graph Data Science (GDS)

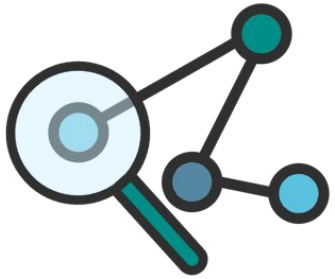
Graph and Data Science

Knowledge Graphs



Find the patterns you're looking for in connected data

Graph Algorithms



Use unsupervised machine learning techniques to identify associations, anomalies, and trends.

Graph Native Machine Learning

Use embeddings to learn the features in your graph that you don't even know are important yet.

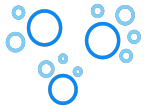
Train in-graph supervised ML models to predict links, labels, and missing data.

Graph Data Science (GDS) Family of Algorithms

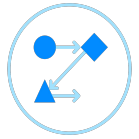
Graph algorithms are a set of instructions that visit the nodes of a graph to analyse the relationships in connected data.

Pathfinding & Search

Centrality & Importance

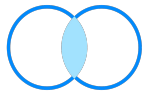


Community Detection

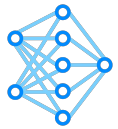


Supervised Machine Learning

Heuristic Link Prediction



Similarity



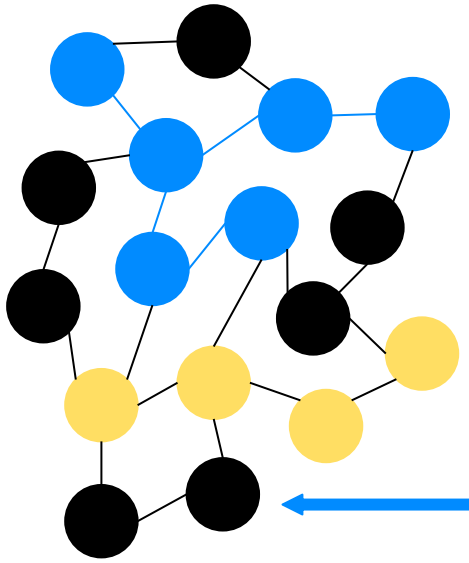
Graph Embeddings

…and more

GDS Process in Neo4j

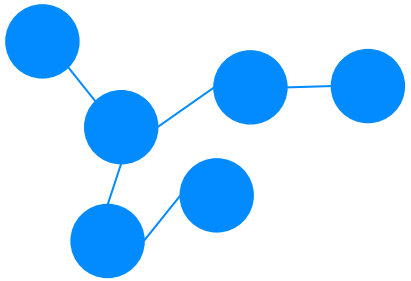
graphsummit

Build Knowledge Graph



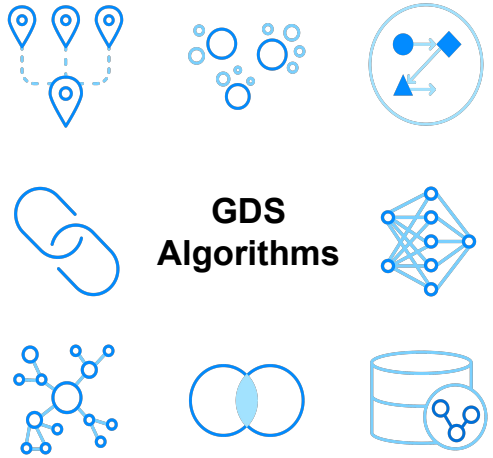
Ingest and model data into the Knowledge Graph. Then, it will be stored into the **disk**.

Graph Projection



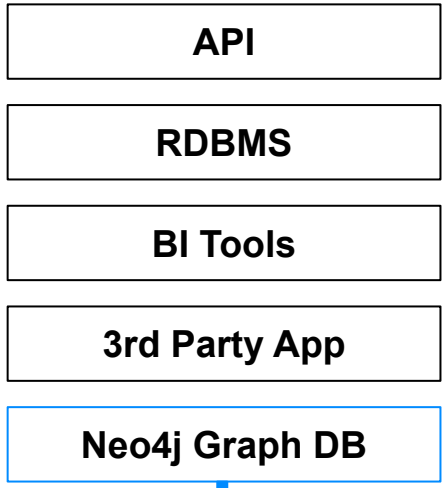
Specific portion of the **graph will be projected in-memory** depending on what the algorithm requires e.g. monopartite, bipartite or multipartite graph.

Execute Algorithm



Graph algorithm and machine learning will be executed within Neo4j.

Use the Graph Insights



Outputs of the GDS algorithms and ML is **feed back** into **Neo4j** or any of the **downstream systems**.

neo4j

Graph Data Science (GDS)
Using Python

Neo4j GDS Python Client Library

graphsummit

Graph Data Science (GDS) Python Client package called **`graphdatascience`** to interact with Neo4j Graph Data Science Library.

#	Syntax	What does it do?
1	`pip install graphdatascience`	Install Graph Data Science (GDS) Python client
2	`gds = GraphDataScience(neo4j_url, auth=(user, password), aura_ds= False)`	Instantiate GDS client session
3	`gds.set_database("neo4j")`	Set specific Neo4j database
4	`gds.run_cypher(<Cypher query>, params={"data": data})`	Run Cypher query with input data
5	`gds.graph.project()`	Create graph projection
6	`gds.graph.drop()`	Drop existing graph projection
7	`gds.<algo>.<mode>()`	Execute a graph algorithm
8	`gds.close()`	Closing a GDS client session

Documentation: Neo4j Graph Data Science Python Client Manual

neo4j

Workshop Scenario

Workshop Environment(s)

colab + neo4j sandbox

jupyter + neo4j desktop

The dataset

README.md

Java CI passing

A Fork of PaySim - Simulating Mobile Money Networks

This is a fork of PaySim 2.0 designed for use as a library while maintaining some ability to run standalone (if desired).

It's first usage is in conjunction with Neo4j to create a network graph, facilitating application of graph algorithms for detecting fraud characteristics.

Why Fork?

Numerous high-level enhancements from the original include:

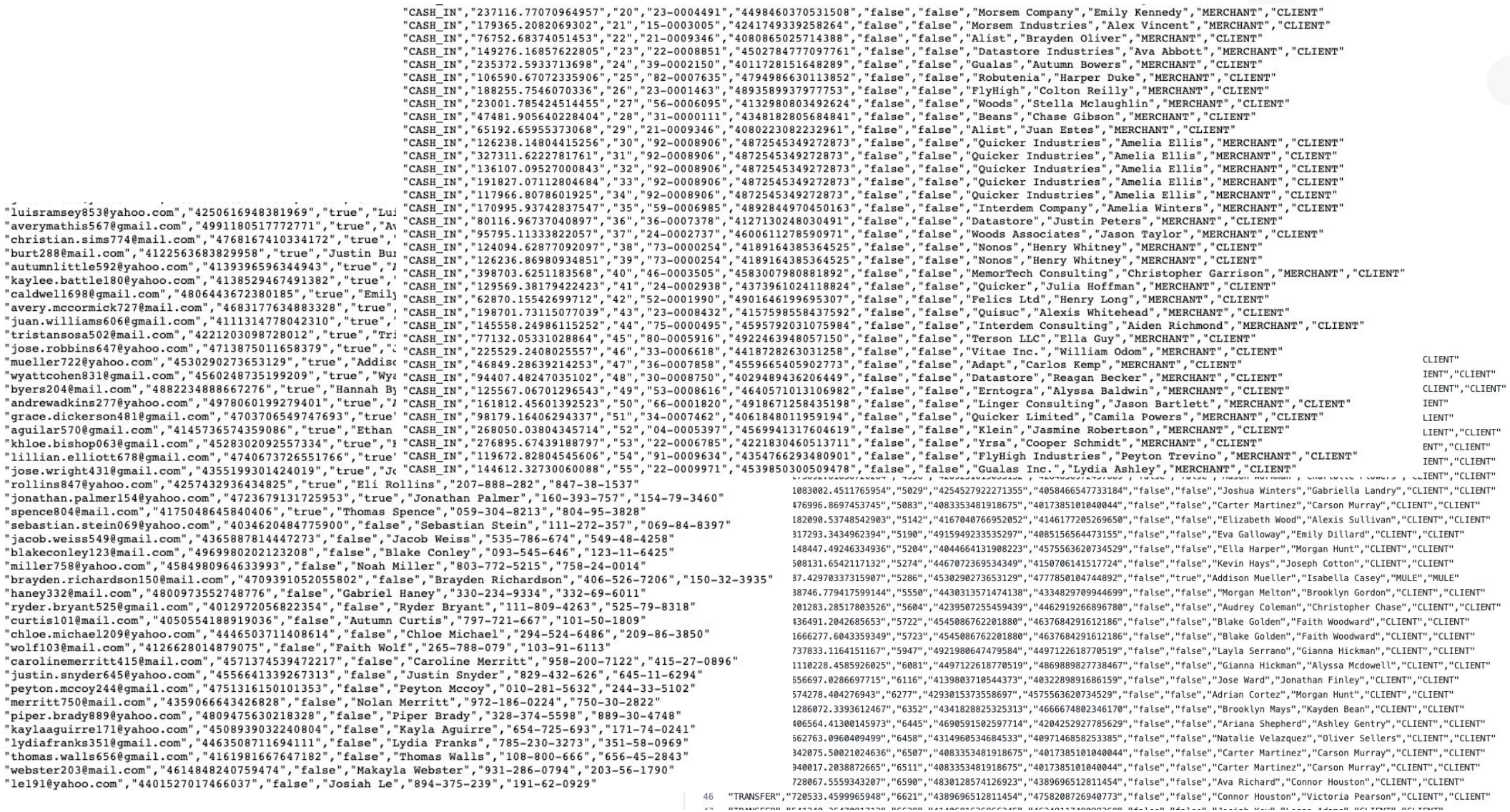
- Implementation of First and Third Party fraudsters
  - 1st Party Fraudsters steal or use fake identities to open new accounts and drain their value
  - 3rd Party Fraudsters prey upon normal clients via "compromised" merchants (like via card skimming) and drain accounts
- Incorporation of "identities" tied to clients to help facilitate First Party Fraud simulation including SSN, Email, and Phone Numbers

https://github.com/voutilad/PaySim

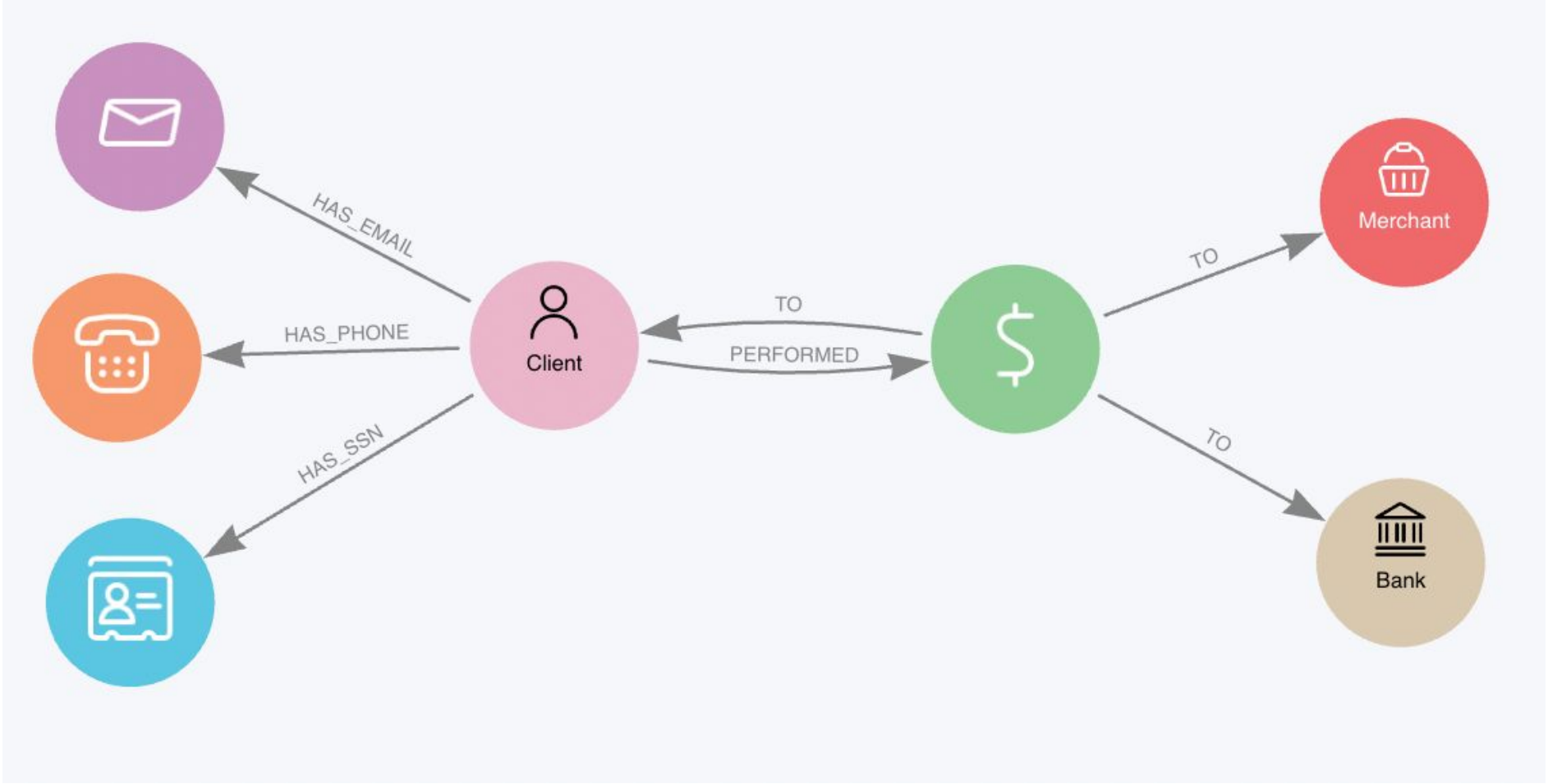
The dataset

Transaction	Description
CashIn	A Client moves money into the network via a Merchant
CashOut	A Client moves money out of the network via a Merchant
Debit	A Client moves money into a Bank
Transfer	A Client sends money to another Client
Payment	A Client exchanges money for something from a Merchant

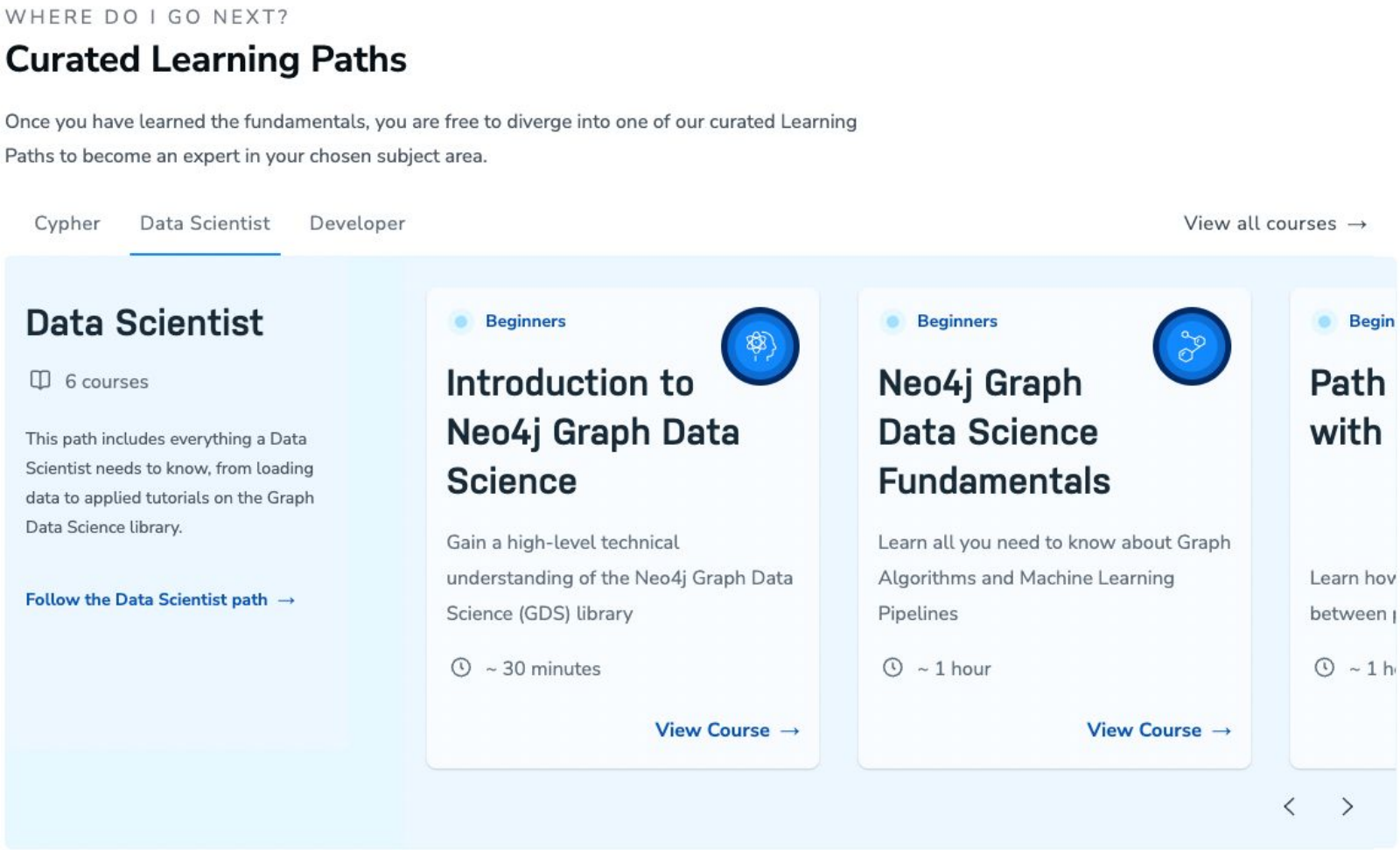
We are starting with lots of this….



…and making sense of it like this



We are just scratching the surface….



https://graphacademy.neo4j.com/

Lets get started !

graphsummit

https://bit.ly/graph-summit-apac-2023

neo4j

graphsummit

Thank You!
Coming up…
Graph Summit Meetup