BI Connector Technical Guide

Powering Business Intelligence with Integrated Graph Database Technology

David Allen, Partner Solution Architect

TABLE OF CONTENTS

The Neo4j BI Connector

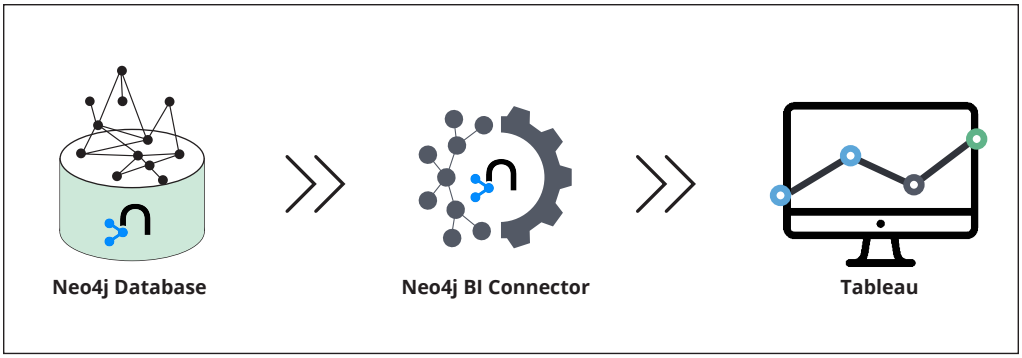
Introduction to Neo4j

Neo4j is the leading graph database for applications including fraud detection and prevention, recommendation engines, master data management, knowledge graphs and so much more. Neo4j brings a connections-first approach to applications and analytics across the enterprise.

Our graph platform is built around the Neo4j native graph database, which supports:

- Graph analytics that help analysts, investigators and data scientists to glean new insights from data
- Data integration to expedite distilling tabular data and big data into graphs and graph visualizations
- •Discovery to help communicate data analyses throughout your organization

With the Neo4j BI Connector, you can deliver direct access to Neo4j graph data from business intelligence (BI) tools such as Tableau, Looker, TIBCO Spotfire Server and Microstrategy. The BI Connector is enterprise-ready and supported to deliver seamless, real-time results.



This paper is focused on just one component of the graph platform's analytics story: Neo4j's connection to BI. There are other facets of graph analysis as well, including the graph data science library, all of which can be added together and used with the BI Connector.

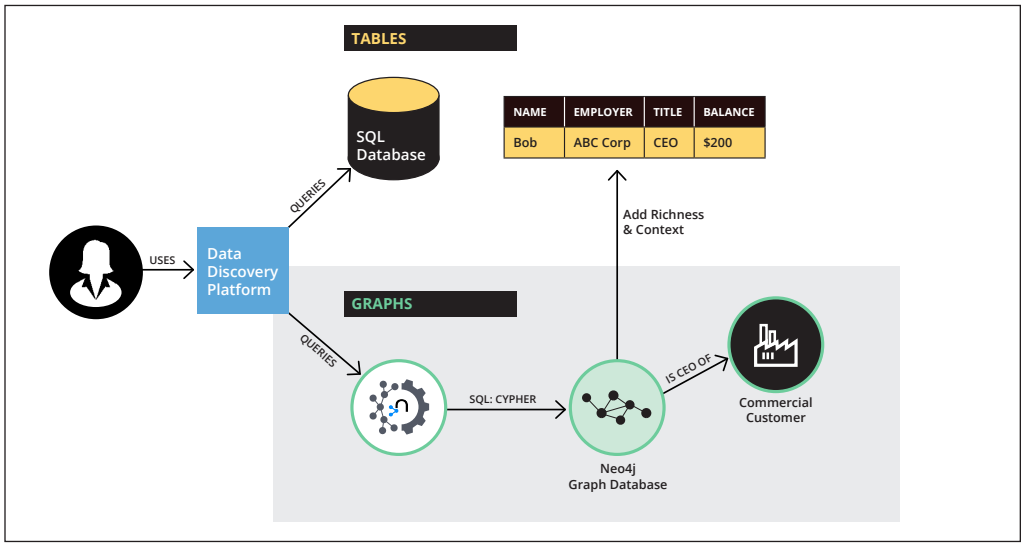
Introduction to the BI Connector

The Neo4j BI Connector delivers direct access to Neo4j graph data from business intelligence (BI) tools such as Tableau, Looker, TIBCO Spotfire Server and Microstrategy. It's the first enterprise-ready, supported product to deliver connected data results to BI users avoiding coding, custom scripting and ungoverned access.

Capabilities

The BI Connector enables data analysts, investigators and data scientists to:

- Connect to a Neo4j database in real time
- Select graph data in the same manner as relational and NoSQL data
- Query using SQL to retrieve data from Neo4j in tabular form
- Analyze and visualize graph data results to gain unique insights from highly connected data



The Neo4j BI Connector enriches traditional analytic workflows with connected data insights.

Availability

The BI Connector is available at no extra charge for Neo4j Enterprise Edition customers. Functionally, the product:

- Builds on the Java Database Connectivity (JDBC) standard
- Translates SQL into Neo4j's native, graph-optimized Cypher language
- Makes connected data insights accessible in real time
- Is fully supported and ready for production and ready for enterprise deployment

Why BI and Graphs Are in Demand

Business intelligence (BI) tooling has been used for years to inform key decisions. The idea is simply to gather information from around an enterprise, fuse it, and build decision-oriented dashboards around that information. This gives executives a view of KPIs and what's happening throughout a complex enterprise.

"Neo4j has always provided great options for enterprise data connectivity with its many language drivers, APIs and data source connectors. The new Neo4j BI Connector is a huge step forward, helping to establish Neo4j as a fully mainstream data management platform that can deliver graph-based insights broadly throughout the enterprise using commonly available BI tools and workflows."

– Dr. Michael Moore, National Practice Lead for Enterprise Knowledge Graphs + AI EY's Data and Analytics Group

"Our team has been testing the Neo4j BI Connector with several popular business intelligence and data visualization tools with great results.Our team has been testing the Neo4j BI Connector with several popular business intelligence and data visualization tools with great results."

– Dr. Michael Moore, National Practice Lead for Enterprise Knowledge Graphs + AI EY's Data and Analytics Group

Drive Decision-Making with Graph Insights

Neo4j and graph approaches, on the other hand, represent a powerful set of capabilities for getting insight into data. But what use is that insight if it cannot be readily combined with existing decision processes?

The Neo4j BI Connector acts as a bridge that makes it easy to get your graph data into other tooling, but the raw technology isn't the story. The real benefit lies in improved decision-making based on better information and key graph insights.

Customers get substantial value out of these graph insights just on their own, but they become even more powerful when they are mixed, matched and integrated with other data sources elsewhere in the enterprise.

Connect the Dots within Organizations

In very large organizations, it's also valuable to share data between organizational units in a simple way. Many Neo4j customers have fraud analytics applications, or other uses of graphs, which are specific to a particular department or project within their enterprise. Business intelligence, however, is often organized as a cross-functional project elsewhere in the enterprise.

The BI Connector acts as a bridge between these units in the enterprise, making it easy for a project to expose their fraud insights to a different BI group, inform other processes and drive decisions. In this scenario, Neo4j is put on equal footing with other databases or data warehouses – such as Oracle. Any of these data sources can be used with equal ease to drive dashboards and insights.

Low Code / No Code Integration

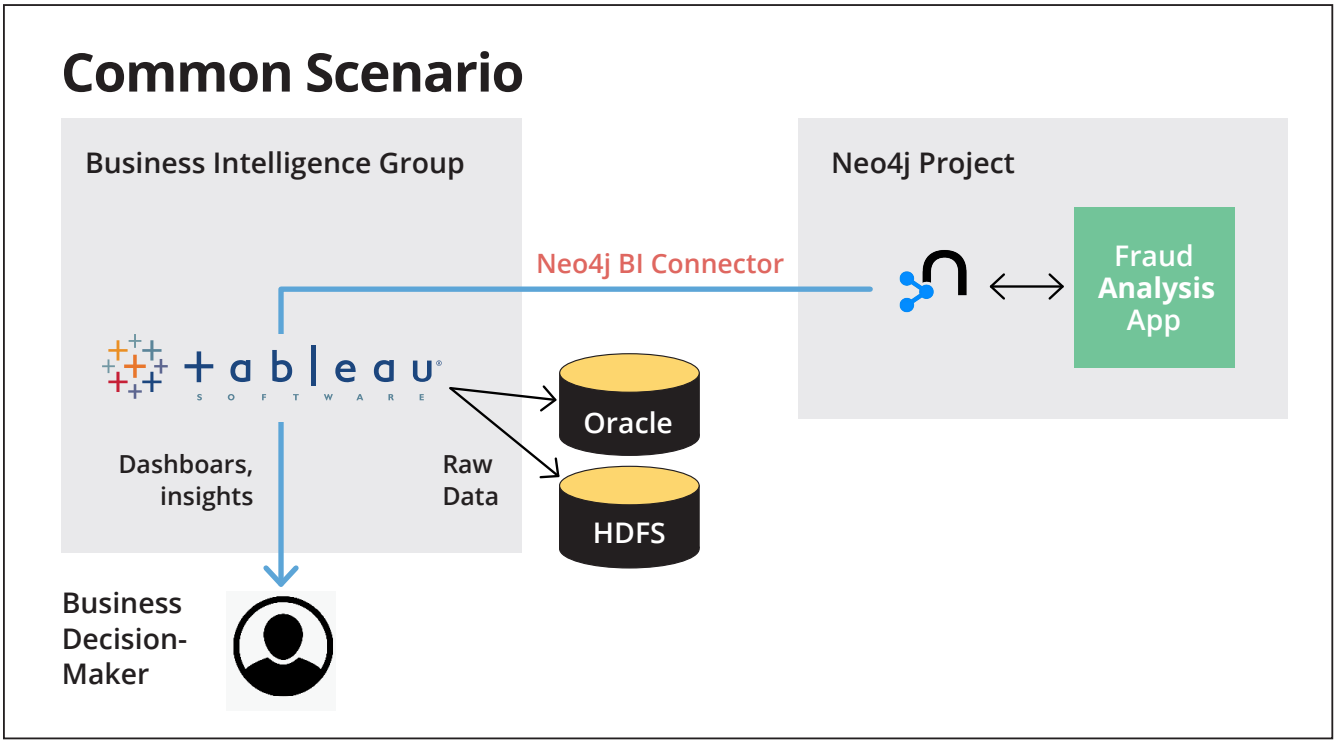
Low-code software development trends favor increasing use of BI platforms over time. There has been an explosion of available information, and tooling allows business analysts to contribute without a deep software engineering background. The tooling, the wide base of analyst talent, and just a little bit of SQL are enablers for getting value out of data. Exposing graph data in a way that is friendly to the relational / BI world simply adds fuel to the fire.

Prior to the BI Connector, graph data was accessible to those who knew Cypher and had graph tooling, but harder to use for non-graph users. Custom code was required, and the state of the art was to develop scripts to export data to CSV periodically.

Benefits and Business Use Cases

Money Laundering and Financial Fraud

Consider a customer that uses Neo4j for money laundering detection. Within such a project, Neo4j performs well. In a wider enterprise scope, however, we need to know whether the overall fraudulent activity is up relative to last year or whether the detected activity is related to what other businesses or governments have reported.



Business Impact Planning

Another customer uses graphs to inform KPIs and business impact planning. In the case of a large supply chain or economic shock, businesses need to know their exposure. By using graph techniques, they can determine the total value of a complex web of relationships between companies. They can then use that total value to inform business decisions on how to adapt to the shock, whether it's a supply chain shock caused by a natural disaster, a pandemic like COVID-19 or a financial crisis.

Network and IT Operations

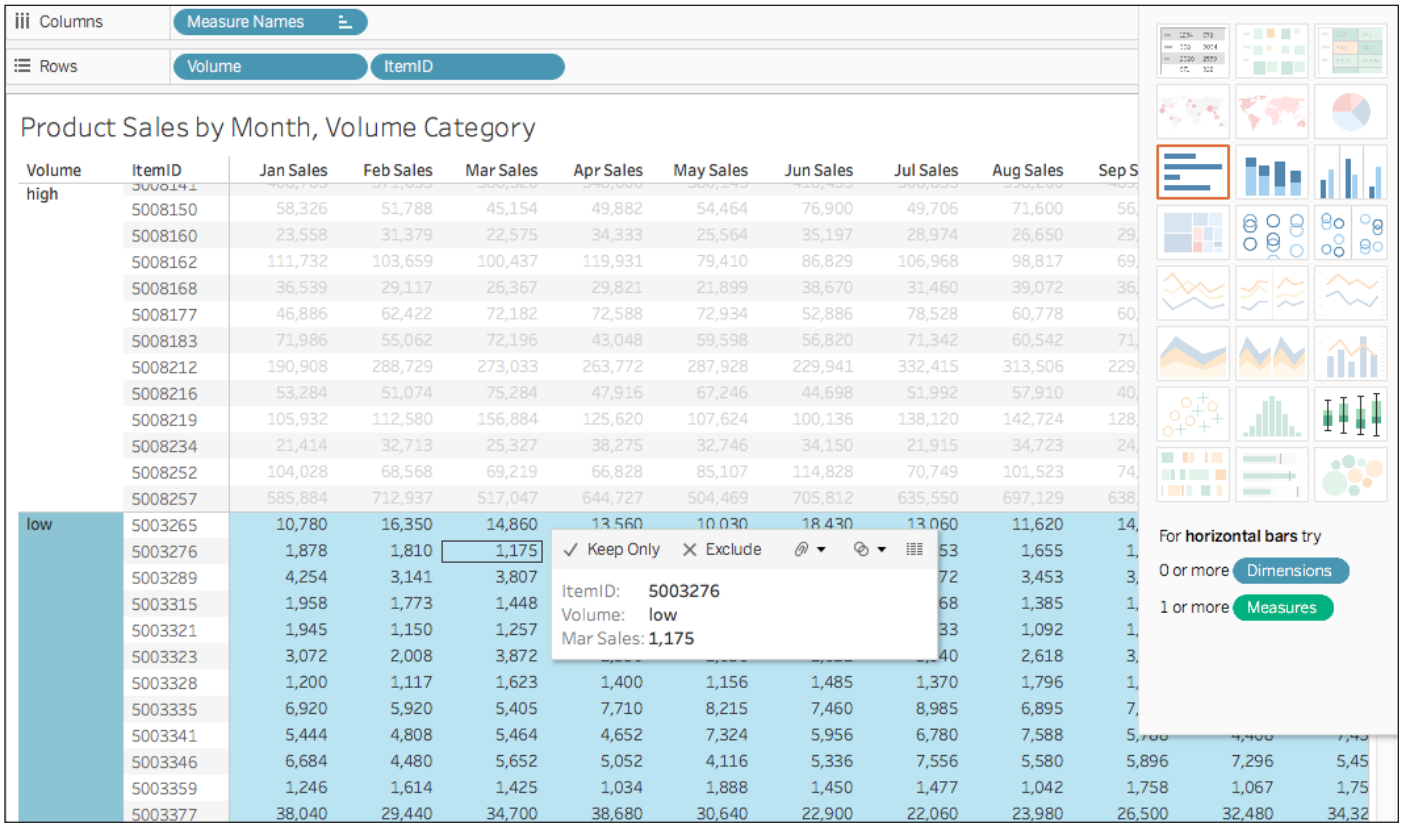
In the Network and IT Management use case, customers use graphs to model the structure of their critical computing infrastructure. This is a different form of impact planning, where we can tie hardware outages within a particular data center up to the business consequences caused by the lack of availability of a certain system.

In the next section, we'll take a deeper look at one business use case – product recommendations – and building a retail promotional discounting program to see how all of the dots connect.

Retail Sales Program: BI Visualizations on Graph Value

Let's imagine we're working with a big online retailer with a lot of data about household purchases on their site. In the above image, we're presenting basic sales figures for available items, along with a classifier of whether they are high or low volume items. This is just a basic starting point though; it would be good to see which households have similar purchase behaviors, across our entire dataset, to better understand purchasing patterns and create a promotional sales program.

Retail Sales Program: BI Visualizations on Graph Value



Volume	ItemID	Jan Sales	Feb Sales	Mar Sales	Apr Sales	May Sales	Jun Sales	Jul Sales	Aug Sales	Sep S
high	5008150	58,326	51,788	45,154	49,882	54,464	76,900	49,706	71,600	56,
	5008160	23,558	31,379	22,575	34,333	25,564	35,197	28,974	26,650	29,
	5008162	111,732	103,659	100,437	119,931	79,410	86,829	106,968	98,817	69,
	5008168	36,539	29,117	26,367	29,821	21,899	38,670	31,460	39,072	36,
	5008177	46,886	62,422	72,182	72,588	72,934	52,886	78,528	60,778	60,
	5008183	71,986	55,062	72,196	43,048	59,598	56,820	71,342	60,542	71,
	5008212	190,908	288,729	273,033	263,772	287,928	229,941	332,415	313,506	229,
	5008216	53,284	51,074	75,284	47,916	67,246	44,698	51,992	57,910	40,
	5008219	105,932	112,580	156,884	125,620	107,624	100,136	138,120	142,724	128,
	5008234	21,414	32,713	25,327	38,275	32,746	34,150	21,915	34,723	24,
	5008252	104,028	68,568	69,219	66,828	85,107	114,828	70,749	101,523	74,
	5008257	585,884	712,937	517,047	644,727	504,469	705,812	635,550	697,129	638,
low	5003265	10,780	16,350	14,860	13,560	10,030	18,430	13,060	11,620	14,
	5003276	1,878	1,810	1,175				53	1,655	1,
	5003289	4,254	3,141	3,807				72	3,453	3,
	5003315	1,958	1,773	1,448				68	1,385	1,
	5003321	1,945	1,150	1,257				33	1,092	1,
	5003323	3,072	2,008	3,872				40	2,618	3,
	5003328	1,200	1,117	1,623	1,400	1,156	1,485	1,370	1,796	1,
	5003335	6,920	5,920	5,405	7,710	8,215	7,460	8,985	6,895	7,
	5003341	5,444	4,808	5,464	4,652	7,324	5,956	6,780	7,588	5,
	5003346	6,684	4,480	5,652	5,052	4,116	5,336	7,556	5,580	5,896
	5003359	1,246	1,614	1,425	1,034	1,888	1,450	1,477	1,042	1,758
	5003377	38,040	29,440	34,700	38,680	30,640	22,900	22,060	23,980	26,500

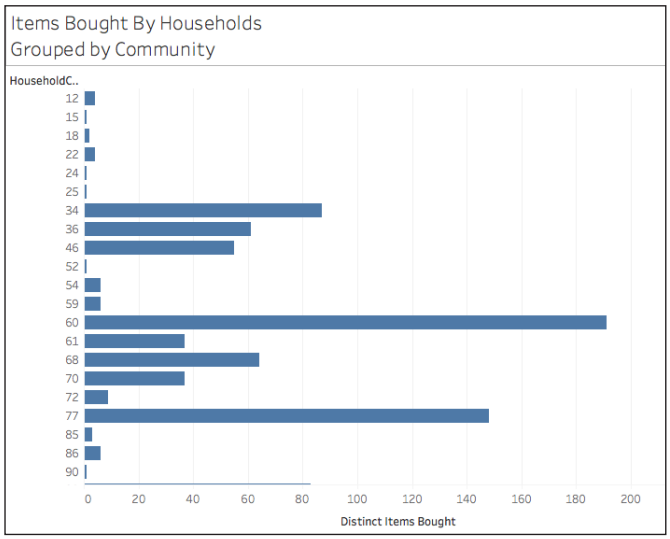
Household Purchasing Patterns

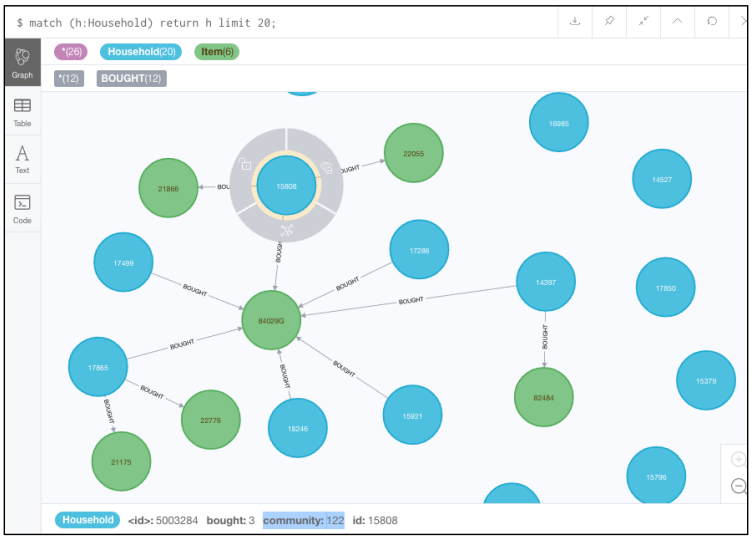
Graph value comes in when you need to understand those similar purchasing behaviors. In the following traditional BI visualization example, you see we've done exactly that. We've grouped all the households into communities based on purchase behavior, and then show how many different items each community bought.

This community grouping data is made possible by Neo4j, which is currently in use by this retailer for product recommendations.

Product recommendations are a very "graphy" use case, because doing it right requires information about the connections between all households, items and purchasing patterns over time.

Below is a behind-the-scenes graph driving that household communities visualization. There are two kinds of things here: households buying things and the items they buy. The relationships between them represent who bought what.





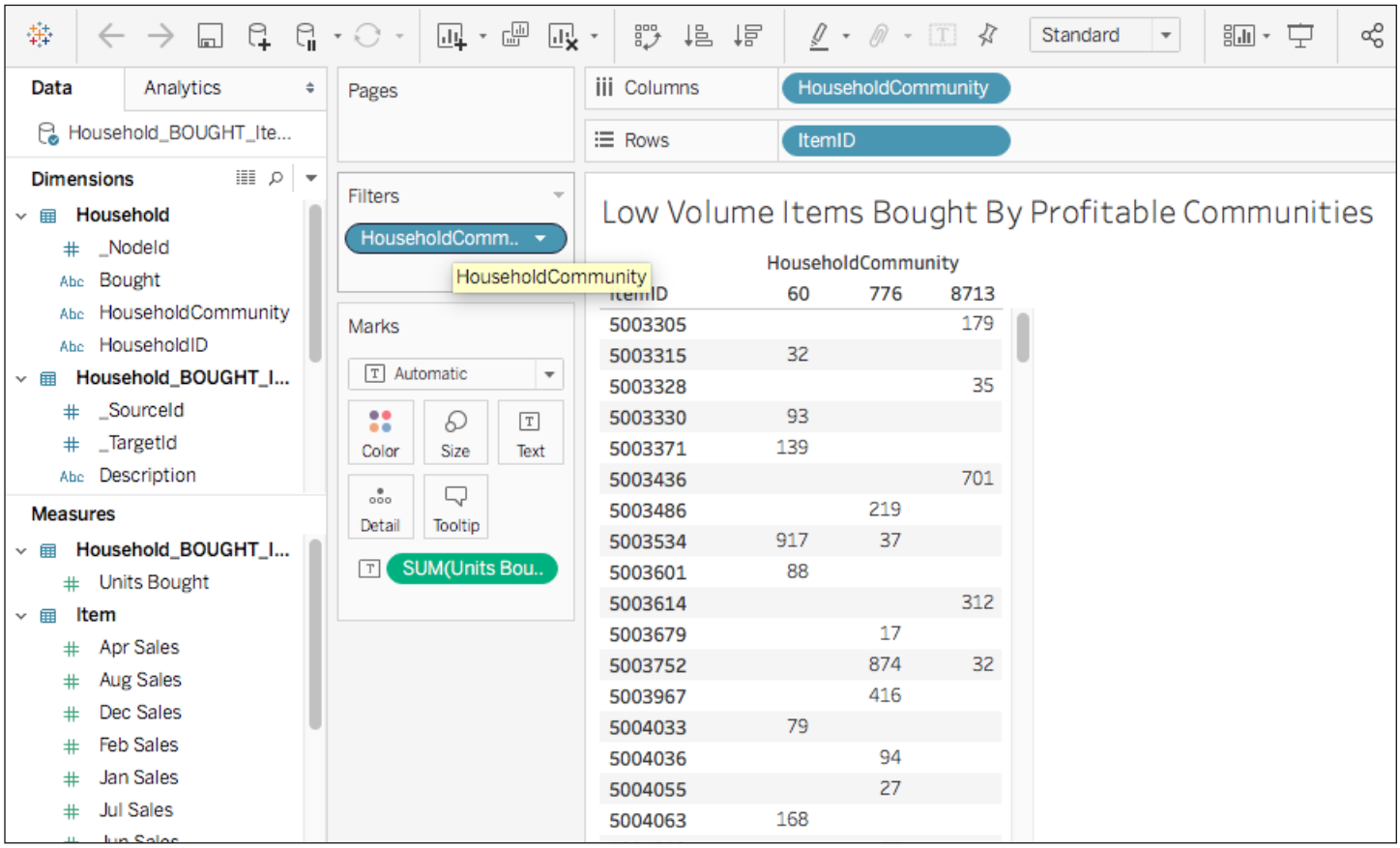
By inspecting each of the items, we can see sales figures for those items. Looking at each of the households, we can see they each have an ID and a community identifier as well. The community identifier was calculated by using the Neo4j Graph Data Science Library to do Louvain Community Detection on the households.

By having this graph and seeing the pattern of which households are buying which items, the retailer can make great recommendations for their online experience. (Customers who enjoyed this item also enjoyed these other items, and so forth.)

Building a Promotional Discounting Program

Now let's imagine that a retail analyst is looking at this data and trying to build a program to increase sales for the site. One of the ways we might do this is to look at low-volume items that are bought by profitable communities. We can look at those communities and pull out the communities that are particularly profitable.

If we focus on profitable communities that buy a lot, we can then look for low-volume items that appeal to them and run a promotional discounting program on those items.



On the top we have communities, and on the left we have IDs. We see that a particular household bought a particular item. We know that certain households have a tendency to buy those items already, and we know they're generally profitable. When they get on a retail site, they're likely to buy other things as well, which is part of our strategy.

One of the ways you can tell Neo4j did a good job with identifying distinct communities in the data is that there is little overlap between their buying patterns. Community 8713 buys a certain item quite a bit, but the other communities didn't buy that item at all. This is to be expected. Because we grouped households into communities, their behavior is different.

Technical Use Case: JDBC Data Integration

The Neo4j BI Connector is a JDBC driver. Fundamentally, what it does is expose graph data via a "virtual relational schema" so that other tools that know how to work with JDBC data can access the graph.

In this sample code repository, a worked example is provided of how to write Java applications that query Neo4j with SQL. While many graph workloads benefit from the use of an official Bolt driver and the Cypher query language through Java, certain applications have the requirement to use JDBC, either as part of an application integration scenario, legacy code or other approach.This may include the use of ETL tooling such as Informatica, Ab Initio and others, but will require some testing on your part to see how this approach fits into your data integration architecture.

The primary limitation to bear in mind is that the Neo4j BI Connector is a read-only connection; that is, you can query tables, but INSERT, UPDATE and DELETE are not supported operations.

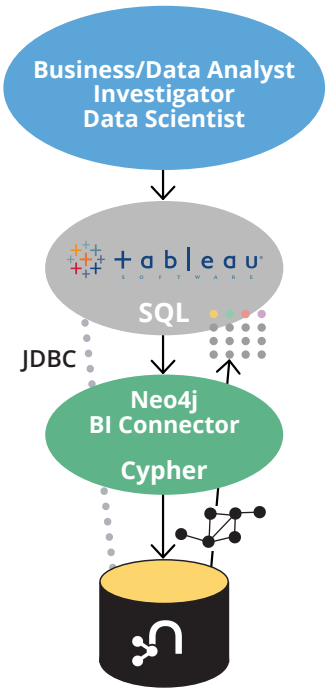
Architecture and How to Approach Implementation

How the Neo4j BI Connector Works

Inside of the BI Connector is a regular instance of the Neo4j official Java Driver / Bolt client for Neo4j.

When users specify a JDBC URL to connect to Neo4j, they embeda Neo4j URL that is passed to the Driver and functions exactly as the standard supported Java Driver works. This means that the BI Connector will transparently support the different connection schemes that Neo4j supports.

It also means that identity separation, security, query throttling and other features, as they pertain to the BI Connector, can be handled as any other Bolt client would be, which we discuss more in the security section.



Graph Visualization Options

With the addition of the BI Connector to the Neo4j portfolio, customer options have expanded. There is now a rich set of different options with complementary strengths.

In this section, we cover where to use each tool, where to avoid them and how a combination of the available tools provides the best insight.

BI Tooling

Traditional tools such as Tableau excel at providing quick drag-and-drop style experiences for building visualizations over tables.

Advantages

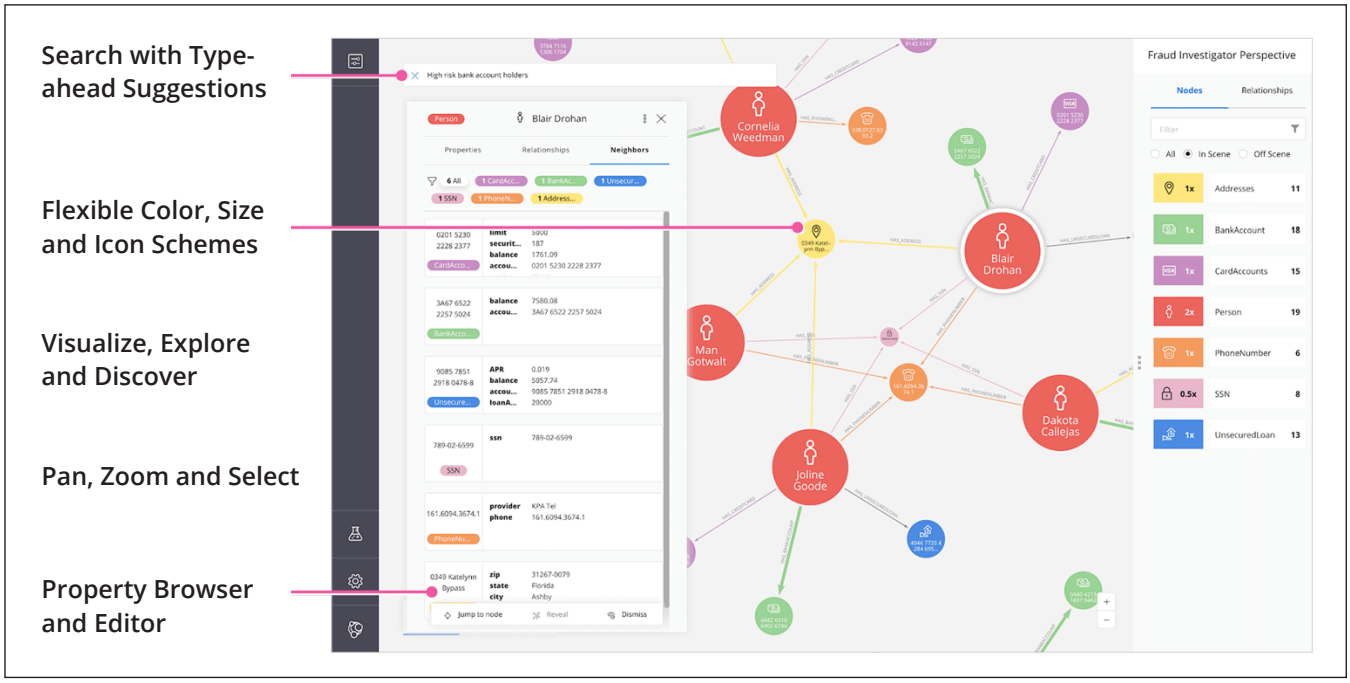
- Very accessible to business analysts all over the enterprise
- Based on SQL technology, with widespread knowledge and ability among the workforce
- Excellent at pie charts, bar charts, line graphs and standard data visualization techniques that easily resonate with decision-makers

Disadvantages

- BI tooling cannot take advantage of "graph native" features, such as Path Finding, as there is no way to "ask graphy questions" of data
- Users will usually write extra data to their graph, representing the results of their calculations such as Community Detection, Path Weighting and so on, so that the result can be visualized

Neo4j Bloom

Bloom gives graph novices and experts the ability to visually investigate and explore their graph data from different business perspectives. Its illustrative, codeless search-to-visualization design makes it the ideal interface for fostering communication between peers, managers and executives, and share the innovative work of their graph development and analytics teams.



Advantages

- Very flexible and open-ended, perfect for exploratory data visualization and hypothesis formation
- Easily allows going from a topological, shaped view of a large network down to the specifics of an individual node

Disadvantages

- Supports primarily a force-directed graph layout; support does not include more rudimentary visualizations such as pie charts, bar charts, etc.

Batch vs. Live Pull

Most BI tooling has the option to "live pull" the data as needed, or to do so in batch, storing the results in a temporary database local to the BI tool. Careful consideration should be given to these options, depending on the size of the database and the frequency of query.

The two options represent a straightforward tradeoff. Live pulling data means the results are always fresh and up to date, but it may incur expensive repeated queries to the database, putting extra load on the system and performing more slowly if the result set is large.

On the other hand, batch operations allow you to pull the data once and store it locally. This usually results in faster local analysis, at the cost that the data is not always fresh and up to date; the batch must be periodically updated for the most recent results. Please consult the Performance Considerations section of this document for more information about memory management in the BI Connector and how the query execution model affects performance.

As a general rule, the bigger the dataset from Neo4j, the more appropriate that a batch pull operation will be over live data query. In the absolute largest scenarios, to avoid load on operational systems, enterprises typically opt for a data warehouse approach that completely offloads all real-time query from transactional systems (like Neo4j) to a centralized data warehouse.

	Live Data Query	Batch Pull
Advantages	Always fresh data	Fast local analysis: little to no impact on the underlying database
Disadvantages	Slow for large volumes, places more load on the underlying production database	Data can become out of date, and batces have to be regularly refreshed

ODBC

As of this writing, the Neo4j BI Connector was a JDBC-only connector. This means the tooling it supports out of the box is limited to those tools that can make JDBC connections to other data sources, which unfortunately excludes some popular BI suites such as PowerBI.

Third party ODBC-to-JDBC bridges are available, such as the one from Progress Software. In this scenario, a JDBC data source such as the BI Connector can be wrapped in an ODBC interface, to allow ODBC tooling to connect. While this introduces another software layer, it's presently the main option if you need to connect to software requiring ODBC.

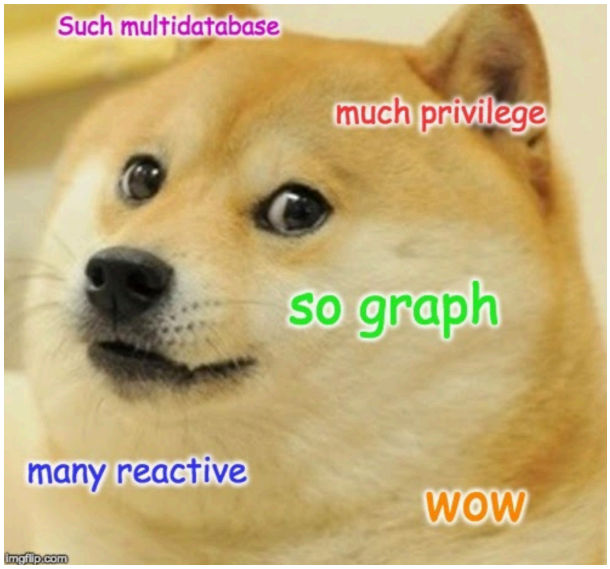
An ODBC version of the Neo4j BI Connector is planned at a later date, and is part of the development roadmap.

Security

In January 2020, Neo4j released the biggest set of improvements to the product ever with version 4.0. This release included a powerful set of new security features, including multi-tenancy and fine-grained access control within a graph.

Neo4j 4.0 Security Controls

Users can now segregate data into multiple, separated graphs to control access and further control access within a specific graph by granting or denying read access down to the individual label, property and relationship.



Restricting Access for BI Users

Remember that the BI Connector functions as a regular client of the database by using regular user credentials and the Bolt protocol. And so, all of this security configuration immediately applies to the BI Connector. Users are able to see the schema the Neo4j graph contains, but queries for data will be subject to the security rules enforced by the database.

Access Control Example

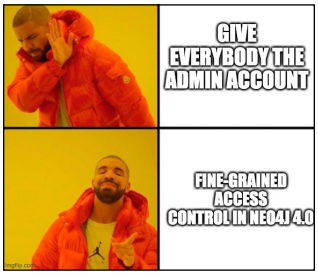
To give a concrete example of these recommendations, an appendix provides a simple scenario of users with address information, phone and social security number.

We create a sample user marketing_analyst who has the security role bi_user. That user is permitted to read and traverse the data in the Neo4j database, but is specifically disallowed from accessing the following information:

- Traversing relationships that connect a user to their social security number
- Reading the SSN attribute, which provides the actual user social security number

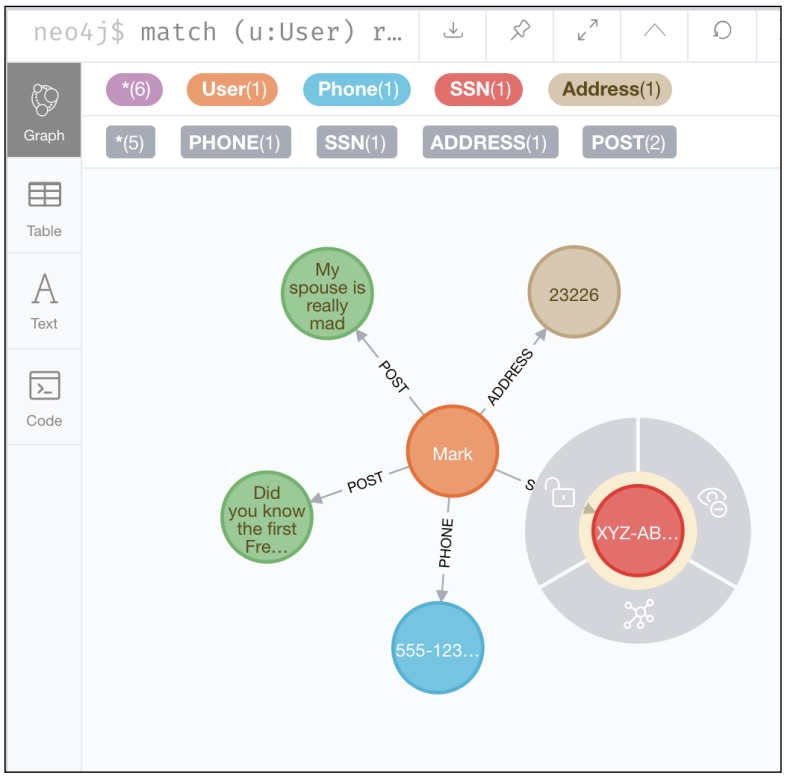
All of this is done in pure system commands to Neo4j, which handles enforcement for us.

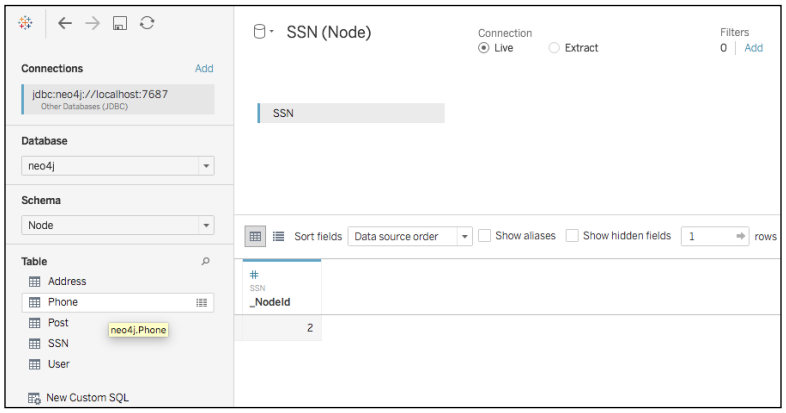
(continues next page)



Access Control Example (continued from previous page)

At the BI Connector layer, all we need to do is connect to the database as the marketing_analyst user. The result is that the SSN table appears empty except for a node ID. It isn't actually empty, this user simply isn't permitted to see the single attribute that is there – the actual social security number.





Key Recommendations:

- External users of the BI Connector should be given their own user accounts and passwords
- Graph administrators should grant appropriate privileges to only the databases necessary
- Centralize access management using existing Neo4j techniques to secure access to your graph data

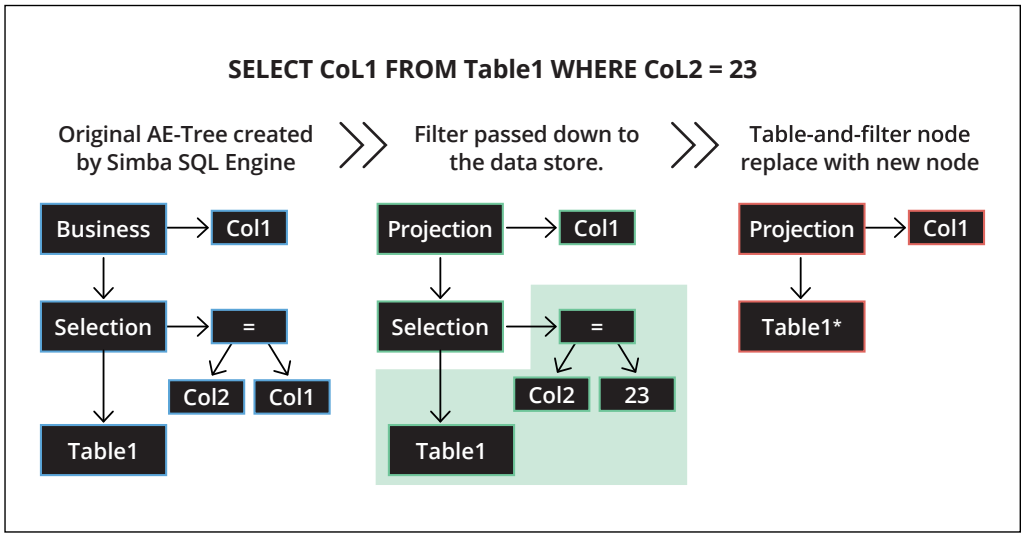
Performance Considerations

As we described above, the BI Connector works by taking SQL queries from a client, transforming them into Cypher and fetching the results. In some cases, a tool might express a SQL construct that does not have an equivalent in Cypher, the BI Connector has an in-memory SQL Execution Engine that can be used to satisfy any standard SQL query.

So while you'll find that you can throw just about any SQL query at the BI Driver and it will work, in some cases the query may be answered by pulling back more information from Neo4j than is strictly necessary to process the results on the client side.

As the BI Driver evolves over time, one of the key priorities is to increase the number of push-down operations to get the best possible performance and fetch the minimum data. Subsequent releases will have more efficient push-down operations, thus improving performance.

If you ever want or need to see what the BI Connector is doing in great detail, enable query logging on your Neo4j server and use a Neo4j monitoring tool to even track those queries while they are in flight.

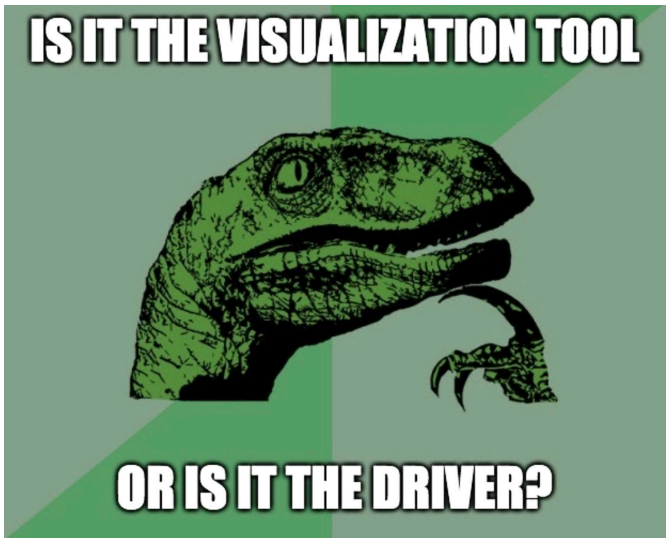


Inside of the SQL Execution Engine, there are a number of memory management techniques that can affect performance for large datasets.

The SQL engine attempts to keep as much data in-memory as possible for performance, but of course there is a tradeoff between execution speed and not consuming too much client side memory. As a result, the SQL engine may spill to disk if the data set gets too big, which can slow down query response times as a tradeoff for not consuming too many resources.

If you would like to read more about the technical details of how query execution takes place, we recommend you consult the Overview of Collaborative Query Execution.

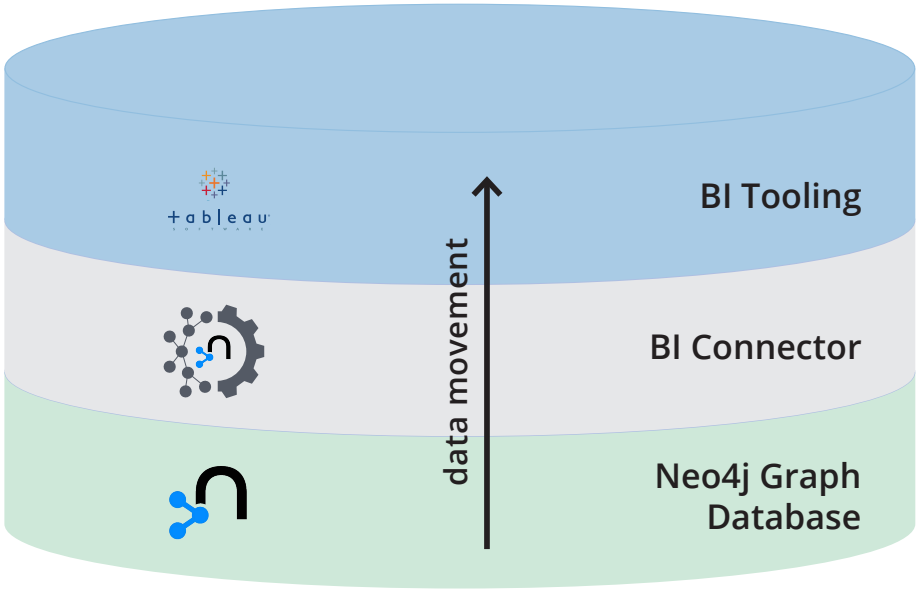
During query execution, the driver will cache intermediate data in memory up to 1 GB and starts spilling to disk if it goes beyond that limit. The driver looks at the java.io.tmpdir system property to determine where in the file system to create the swap files. Users can control the swap file location via the -Djava.io.tmpdir JVM property. Currently, the driver does not have a configuration for the users to control the memory limit.



Troubleshooting

When issues arise, it's a good thing to start with a visualization of what's happening in our technology stack, so we quickly isolate what the issue is and move to resolution.

The following image shows a flow of data from a Neo4j database at the bottom to BI Tooling (such as Tableau) at the top.



Let's address each layer individually, starting at the bottom, to provide some trouble-shooting tips and tricks for when things aren't working correctly.

Neo4j Database

- At the database layer, you may wish to enable query logging to see the direct Cypher queries the BI Connector is sending to the database.
- The use of a Neo4j monitoring tool such as Halin provides valuable advice on how much free memory your graph cluster has, and what overall throughput is like. This is important context; if the cluster is overloaded, the BI Connector as another client cannot perform correctly.

Neo4j BI Connector

The use of connector logging is highly recommended to help troubleshoot; in the event that you need to submit a ticket to Neo4j, it's likely you would be asked to provide these logs. Connector logging is enabled by adding something to the JDBC connection string, like so:

```
jdbc:neo4j://archimedes:5480?LogLevel=3&LogPath=C:\\
temp&EnableJavaDriverLogging=true
```

This instructs the driver to keep logs of its interactions with SQL clients and store those text logs in the temp directory. The EnableJavaDriverLogging flag tells the BI Connector to log its interactions with Neo4j (as a Bolt client) as well.

When inspecting these logs, there are several things to look for:

- Actual SQL sent to the driver, and whether or not the driver could respond / encountered an error
- Evidence of any driver communication errors to the Neo4j database
- Evidence of the use of connections; some bad-behaved SQL clients may create an excessive number of parallel open connections, which can slow things down

BI Tooling

Ultimately, there are as many ways of troubleshooting these tools as there are tools. For the most popular Tableau installations, we recommend looking at these instructions to change the log level associated with the product, which gives some insight into what Tableau is doing.

Appendix 1: Permissions Scenario

This appendix contains sample data necessary to demonstrate the points made in the security section. This simple example can be used to demonstrate some of the key concepts behind fine-grained access control in Neo4j 4.0.

Sample Data

```
CREATE (mark:User { name: 'Mark' })
WITH mark
CREATE (mark)-[:PHONE]->(:Phone { number: '555-123-456' })
CREATE (mark)-[:SSN]->(:SSN { ssn: 'XYZ-ABC-DEFG' })
CREATE (mark)-[:ADDRESS]->(:Address {
    street: '123 Elm St',
    state: 'VA',
    zip: '23226'
})
CREATE (mark)-[:POST]->(:Post {
    content: 'My spouse is really mad at the fact that I have
no sense of direction. So I packed up my stuff and right.'
})
CREATE (mark)-[:POST]->(:Post {
    content: "Did you know the first French fries weren't
actually cooked in France? They were cooked in Greece."
});
```

Role Setup

```
:use system
create role bi_user;
GRANT ACCESS ON DATABASE neo4j TO bi_user;
GRANT MATCH {*} ON GRAPH neo4j TO bi_user;
GRANT READ { * } ON GRAPH neo4j NODES User, Phone, Address TO
bi_user;
GRANT TRAVERSE ON GRAPH neo4j RELATIONSHIPS PHONE, ADDRESS TO
bi_user;
DENY TRAVERSE ON GRAPH neo4j RELATIONSHIPS SSN TO bi_user;
DENY READ { * } ON GRAPH neo4j NODES SSN to bi_user;
```

User Creation and Role Assignment

```
CREATE USER marketing_analyst SET PASSWORD 'secret' SET
PASSWORD CHANGE NOT REQUIRED;
GRANT ROLE bi_user TO marketing_analyst;
```

Appendix 2: Java JDBC Access to Graph Data

The following sample listing is taken from the example repository that shows the use of this technique. The code listing below shows the essence of how the BI Connector can be used just like any other JDBC driver.

```
public static void main(String[] args) throws Exception {
    Connection conn = null;
    Statement stmt = null;

    try {
        //STEP 2: Register JDBC driver
        Class.forName(JDBC_DRIVER);

        //STEP 3: Open a connection
        System.out.println("Connecting to database...");
        conn = DriverManager.getConnection(DB_URL, USER, PASS);

        DatabaseMetaData md = conn.getMetaData();
        ResultSet rs = md.getTables(null, null, null, null);

        while(rs.next()) {
            String schema = rs.getString("TABLE_SCHEM");
            String name = rs.getString("TABLE_NAME");

            System.out.println("\n\nThis connection has a table named " + schema
  + "." + name);

            firstRecordFrom(conn, name);
        }
        rs.close();
        stmt.close();
        conn.close();
    } catch (SQLException se) {
        //Handle errors for JDBC
        se.printStackTrace();
    } catch (Exception e) {
        //Handle errors for Class.forName
        e.printStackTrace();
    } finally {
        //finally block used to close resources
        try {
            if (stmt != null)
                stmt.close();
        } catch (SQLException se2) {

        }// nothing we can do
        try {
            if (conn != null)
                conn.close();
        } catch (SQLException se) {
            se.printStackTrace();
        } //end finally try
    } //end try
    System.out.println("Goodbye!");
}
```