Magnitude Simba Neo4j Data Connector for Business Intelligence Tools

JDBC Installation and Configuration Guide

Version 1.0.10

January 2022

Copyright

This document was released in January 2022.

Copyright ©2014–2022 Magnitude Software, Inc., an insightsoftware company. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from Magnitude, Inc.

The information in this document is subject to change without notice. Magnitude, Inc. strives to keep this information accurate but does not warrant that this document is error-free.

Any Magnitude product described herein is licensed exclusively subject to the conditions set forth in your Magnitude license agreement.

Simba, the Simba logo, SimbaEngine, and Simba Technologies are registered trademarks of Simba Technologies Inc. in Canada, the United States and/or other countries. All other trademarks and/or servicemarks are the property of their respective owners.

All other company and product names mentioned herein are used for identification purposes only and may be trademarks or registered trademarks of their respective owners.

Information about the third-party products is contained in a third-party-licenses.txt file that is packaged with the software.

Contact Us

Magnitude Software, Inc.

www.magnitude.com

About This Guide

Purpose

The *Magnitude Simba Neo4j Data Connector for Business Intelligence Tools JDBC Installation and Configuration Guide* explains how to install and configure the Magnitude Simba Neo4j Data Connector for Business Intelligence Tools on all supported platforms. The guide also provides details related to features of the connector.

Audience

The guide is intended for end users of the Simba Neo4j BI Connector.

Knowledge Prerequisites

To use the Simba Neo4j BI Connector, the following knowledge is helpful:

- Familiarity with the platform on which you are using the Simba Neo4j BI Connector
- Ability to use the data store to which the Simba Neo4j BI Connector is connecting
- An understanding of the role of JDBC technologies in connecting to a data store
- Experience creating and configuring JDBC connections
- Exposure to SQL

Document Conventions

Italics are used when referring to book and document titles.

Bold is used in procedures for graphical user interface elements that a user clicks and text that a user types.

`Monospace font` indicates commands, source code or contents of text files.

> **Note:**
>
> A text box with a pencil icon indicates a short note appended to a paragraph.

> **Important:**
>
> A text box with an exclamation mark indicates an important comment related to the preceding paragraph.

Contents

About the Simba Neo4j BI Connector 6
System Requirements 7
Simba Neo4j BI Connector Files 8
Installing and Using the Simba Neo4j BI Connector 9
Referencing the JDBC Connector Libraries 9
Registering the Connector Class 10
Building the Connection URL 11
Configuring Authentication 13
Configuring SSL Connections 14
Configuring Logging 15
Features 17
Security and Authentication 17
SQL Connector 18
Catalog and Schema Support 18
Schema Definition 18
Nodes and Relationships 20
Aggregate Function Passdown 21
Join Passdown 23
Filter Passdown 25
Cypher-backed Views 25
Data Types 37
Connector Configuration Options 39
AssumeUTC 39
Auth_Type 39
ConnectionTimeoutMS 40
DefaultBinaryColumnLength 40
DefaultStringColumnLength 40
EnableJavaDriverLogging 41
ExcludeLabels 41
ExcludeRels 41

FetchSize 42
IncludeLabels 42
IncludeRels 42
LabelSeparator 43
LogLevel 43
LogPath 44
LabelsSampleSize 45
MaxIdentifierLen 45
PWD 45
RelNodeSeparator 45
RelsSampleSize 46
ServerPolicy 46
SSL 46
sslCustomCertPath 47
sslTrustStrategy 47
sslVerifyHostname 48
StrictlyUseBoltScheme 48
UID 49
ViewDefinitionFile 50

Third-Party Trademarks 51

About the Simba Neo4j BI Connector

The Magnitude Simba Neo4j Data Connector for Business Intelligence Tools enables SQL queries to be performed on a Neo4j graph database. The connector enables organizations to use their BI tools to analyze and construct reports on such databases.

The Simba Neo4j BI Connector complies with the JDBC 4.2 data standard. JDBC is one of the most established and widely supported APIs for connecting to and working with databases. At the heart of the technology is the JDBC connector, which connects an application to the database. For more information about JDBC, see *Data Access Standards* on the Simba Technologies website: https://www.simba.com/resources/data-access-standards-glossary.

This guide is suitable for users who want to access data residing within Neo4j from their desktop environment. Application developers might also find the information helpful. Refer to your application for details on connecting via JDBC.

System Requirements

Each machine where you use the Simba Neo4j BI Connector must have Java Runtime Environment (JRE) 8.0 or 11.0 installed.

Note:

The reactor-core third-party library inside the Neo4j JDBC connector uses JDK internal APIs that have been removed from Java 11. However, these APIs are only used to enhance its stack traces and will not be invoked if Java 11 is detected.

The Simba Neo4j BI Connector supports Neo4j server version 4.x and Neo4j Aura. In addition, the Neo4j APOC library must be installed on the server machine.

The required APOC version depends on the Neo4j version that you are connecting to:

- Neo4j server version 4.x requires the matching APOC version or later (minimum version 4.0.0.4).

We recommend that you make sure APOC procedures are made accessible by adding or appending `apoc.*` to the list of comma separated procedures provided through `dbms.security.procedures.unrestricted` in the Neo4j server configuration file.

For example:

```
dbms.security.procedures.unrestricted=algo.*,apoc.*
```

Simba Neo4j BI Connector Files

The Simba Neo4j BI Connector is delivered in a ZIP archive named `SimbaNeo4jJDBC-[Version].zip`, where *[Version]* is the version number of the connector.

The archive contains the connector supporting the JDBC API version indicated in the archive name, as well as release notes and third-party license information. In addition, the required third-party libraries and dependencies are packaged and shared in the connector JAR file in the archive.

Installing and Using the Simba Neo4j BI Connector

To install the Simba Neo4j BI Connector on your machine, extract the files from the ZIP archive to the directory of your choice.

Important:

If you received a license file through email, then you must copy the file into the same directory as the connector JAR file before you can use the Simba Neo4j BI Connector.

To access a Neo4j data store using the Simba Neo4j BI Connector, you need to configure the following:

- The list of connector library files (see Referencing the JDBC Connector Libraries on page 9)
- The `Driver` or `DataSource` class (see Registering the Connector Class on page 10)
- The connection URL for the connector (see Building the Connection URL on page 11)

Important:

The Simba Neo4j BI Connector provides read-only access to Neo4j data stores.

Referencing the JDBC Connector Libraries

Before you use the Simba Neo4j BI Connector, the JDBC application or Java code that you are using to connect to your data must be able to access the connector JAR file. In the application or code, specify the appropriate fat JAR file for the JDBC version that you are using.

Using the Connector in a JDBC Application

Most JDBC applications provide a set of configuration options for adding a list of connector library files. Use the provided options to include the appropriate fat JAR file from the ZIP archive as part of the connector configuration in the application. For more information, see the documentation for your JDBC application.

Using the Connector in Java Code

You must include all the connector library files in the class path. This is the path that the Java Runtime Environment searches for classes and other resource files. For more information, see "Setting the Class Path" in the appropriate Java SE Documentation.

- For Windows: http://docs.oracle.com/javase/8/docs/technotes/tools/windows/classpath.html
- For Linux and Solaris: http://docs.oracle.com/javase/8/docs/technotes/tools/unix/classpath.html

Registering the Connector Class

Before connecting to your data, you must register the appropriate class for your application.

The following classes are used to connect the Simba Neo4j BI Connector to Neo4j data stores:

- The `Driver` classes extend `java.sql.Driver`.
- The `DataSource` classes extend `javax.sql.DataSource` and `javax.sql.ConnectionPoolDataSource`.

The connector supports the following fully-qualified class names (FQCNs) that are independent of the JDBC version:

- `com.simba.neo4j.jdbc.Driver`
- `com.simba.neo4j.jdbc.DataSource`

The following sample code shows how to use the `DriverManager` class to establish a connection for JDBC 4.2:

```
private static Connection connectViaDM() throws Exception
{

    Connection connection = null;
    connection = DriverManager.getConnection(CONNECTION_
    URL);
    return connection;

}
```

The following sample code shows how to use the `DataSource` class to establish a connection:

```
private static Connection connectViaDS() throws Exception
{

    Connection connection = null;
    DataSource ds = new com.simba.neo4j.jdbc.DataSource
    ();
    ds.setURL(CONNECTION_URL);
    connection = ds.getConnection();
    return connection;

}
```

Building the Connection URL

Use the connection URL to supply connection information to the data store that you are accessing. Depending on the version of the Neo4j server that you are connecting to, the connector may require different connection information:

- For connection information for a Neo4j 4.x instance, see Building the Connection URL for Neo4j 4.x on page 11.

Building the Connection URL for Neo4j 4.x

The following is the format of a basic connection URL for connecting to a Neo4j 4.x instance:

```
jdbc:neo4j://[Host]:[Port]/[Database]
```

In this example:

- *[Host]* is the host name or IP address of the Neo4j server.
- *[Port]* is the number of the TCP port that the Neo4j server uses to listen for client connections. The default Neo4j port is 7687.
- *[Database]* is the optional name of the database to connect to on the Neo4j server. This parameter is optional. If it is not specified, the connector connects to the default database specified by the server.

You can also specify optional settings such as authentication, logging, or any of the other connection properties supported by the connector. Additional properties must be separated by either an ampersand (`&`) or a semicolon (`;`). For a list of the properties available in the connector, see Connector Configuration Options on page 39.

The following is the format of a connection URL that specifies some optional settings:

```
jdbc:neo4j://[Host]:[Port]/[OptionalDatabase]?[Property1]=
[Value]&[Property2]=[Value]&...
```

For example, to connect to a Neo4j 4.x instance on port 7687 with the database `mydata`, on a Neo4j server installed on a machine named `archimedes`, using basic authentication, with the username `skroob` and password `12345`, you would use the following connection URL:

```
jdbc:neo4j://archimedes:7687/mydata?&UID=skroob&PWD=12345
```

Important:

- Property values are case-sensitive.
- Do not duplicate properties in the connection URL.

Configuring Authentication

By default, the Simba Neo4j BI Connector requires a user name and password when connecting to a Neo4j data store. You can also configure the connector to connect to the data store without requiring authentication.

You provide this information to the connector in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 11.

To configure authentication:

1. Set the `UID` property to an appropriate user name for accessing the Neo4j server.
2. Set the `PWD` property to the password corresponding to the user name you provided.

For example:

```
jdbc:neo4j://archimedes:5480?UID=skroob&PWD=12345
```

To disable authentication:

- Set the `Auth_Type` property to `None`.

For example:

```
jdbc:neo4j://archimedes:5480?Auth_Type=None
```

Configuring SSL Connections

Note:

In this documentation, "SSL" indicates both TLS (Transport Layer Security) and SSL (Secure Sockets Layer). The connector supports industry-standard versions of TLS/SSL.

If you are connecting to a Neo4j server that has SSL enabled, you can configure the connector to connect to an SSL-enabled socket. When connecting to a server over SSL, the connector uses one-way authentication to verify the identity of the server.

You provide the configuration information to the connector in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 11.

To configure an SSL connection:

1. Set the `SSL` property to `true`.
2. To perform host name verification, set the `sslVerifyHostname` property to `true`.
3. Depending on the trust store strategy you want to use, set the `sslTrustStrategy` property to one of the following:
   - To trust all certificates, set the property to `TRUST_ALL_CERTIFICATES`.
   - To trust only certificates that have been signed by a trusted authority, set the property to `TRUST_CUSTOM_CA_SIGNED_CERTIFICATES`.
   - To use the system trust store certificates, set the property to `TRUST_SYSTEM_CA_SIGNED_CERTIFICATES`.
4. If you set the `sslTrustStrategy` property to `TRUST_CUSTOM_CA_SIGNED_CERTIFICATES`, set the `sslCustomCertPath` property to the full path of the TrustStore that you want to use.

For example:

```
jdbc:neo4j
://archimedes:7687/mydata?SSL=true&sslVerifyHostname=true&ssl
TrustStrategy=TRUST_CUSTOM_CA_SIGNED_
CERTIFICATES&sslCustomCertPath=C:\\TrustStore
```

Configuring Logging

To help troubleshoot issues, you can enable logging in the connector.

Important:

Only enable logging long enough to capture an issue. Logging decreases performance and can consume a large quantity of disk space.

The settings for logging apply to every connection that uses the Simba Neo4j BI Connector, so make sure to disable the feature after you are done using it.

In the connection URL, set the `LogLevel` key to enable logging at the desired level of detail. The following table lists the logging levels provided by the Simba Neo4j BI Connector, in order from least verbose to most verbose.

LogLevel Value	Description
0	Disable all logging.
1	Log severe error events that lead the connector to abort.
2	Log error events that might allow the connector to continue running.
3	Log events that might result in an error if action is not taken.
4	Log general information that describes the progress of the connector.
5	Log detailed information that is useful for debugging the connector.
6	Log all connector activity.

To enable logging:

1. Set the `LogLevel` property to the desired level of information to include in log files.
2. Set the `LogPath` property to the full path to the folder where you want to save log files. To make sure that the connection URL is compatible with all

JDBC applications, escape the backslashes (\) in your file path by typing another backslash.

For example, the following connection URL enables logging level 3 and saves the log files in the C:\temp folder:

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

3. Optionally, to include information from the Neo4j Java connector in the log, set EnableJavaDriverLogging to true.
4. To make sure that the new settings take effect, restart your JDBC application and reconnect to the server.

The Simba Neo4j BI Connector produces the following log files in the location specified in the LogPath property:

- A simbaneo4jjdbcdriver.log file that logs connector activity that is not specific to a connection.
- A simbaneo4jjdbcdriver_connection_*[Number]*.log file for each connection made to the database, where *[Number]* is a number that identifies each log file. This file logs connector activity that is specific to the connection.

If the LogPath value is invalid, then the connector sends the logged information to the standard output stream (System.out).

To disable logging:

1. Set the LogLevel property to 0.
2. To make sure that the new setting takes effect, restart your JDBC application and reconnect to the server.

Features

More information is provided on the following features of the Simba Neo4j BI Connector:

- Security and Authentication on page 17
- SQL Connector on page 18
- Catalog and Schema Support on page 18
- Schema Definition on page 18
- Nodes and Relationships on page 20
- Aggregate Function Passdown on page 21
- Join Passdown on page 23
- Filter Passdown on page 25
- Cypher-backed Views on page 25
- Data Types on page 37

Security and Authentication

Note:

In this documentation, "SSL" indicates both TLS (Transport Layer Security) and SSL (Secure Sockets Layer). The connector supports industry-standard versions of TLS/SSL.

To protect data from unauthorized access, some Neo4j data stores require connections to be authenticated with user credentials. For information about configuring the connector to authenticate your connections, see Configuring Authentication on page 13. For information about configuring authentication on your Neo4j server, see the Neo4j documentation.

Additionally, the connector supports one-way SSL encryption. SSL encryption protects data and credentials when they are transferred over the network, and provides stronger security than authentication alone.

The SSL version that the connector supports depends on the JVM version that you are using. For information about the SSL versions that are supported by each version of Java, see "Diagnosing TLS, SSL, and HTTPS" on the Java Platform Group Product Management Blog: https://blogs.oracle.com/java-platform-group/entry/diagnosing_tls_ssl_and_https.

Note:

The SSL version used for the connection is the highest version that is supported by both the connector and the server, which is determined at connection time.

SQL Connector

The SQL Connector feature of the connector enables applications to execute standard SQL queries against Neo4j. It converts SQL-92 queries to Neo4j API calls and processes the results.

Catalog and Schema Support

The Simba Neo4j BI Connector supports both catalogs and schemas to make it easy for the connector to work with various JDBC applications.

Databases are mapped to catalogs. Depending on the version of Neo4j Server, the connector supports the following databases:

- For Neo4j Server 4.x, you can specify which database to connect to. If you do not specify a database, the connector connects to the default database that is specified by the server.

The connector supports two schemas:

- Node: a schema containing all the nodes in the graph that the connector connects to.
- Relationship: a schema containing all the relationships in the graph that the connector connects to.

For information about how the connector maps node and relationship data to a standard relational table format, see Nodes and Relationships on page 20.

Schema Definition

When the connector connects to a Neo4j database, it generates a temporary schema definition. This definition is based on the metadata retrieved through the following APOC procedures:

- **apoc.meta.nodeTypeProperties()**: the API call is used to obtain metadata for nodes with labels.
- **apoc.meta.relTypeProperties()**: the API call is used to obtain metadata for relationship types.

The input parameters for these API calls and their mapping to the connector's connection string properties are as defined in the table below:

Input Parameter	Configuration Option
includeLabels	IncludeLabels
includeRels	IncludeRels
excludeLabels	ExcludeLabels
excludeRels	ExcludeRels
sample	LabelsSampleSize
maxRels	RelsSampleSize

Temporary schema definitions generated by the connector do not persist after the connection is closed. Also, the connector might generate different schema definitions during subsequent connections to the same database.

Note:

Make sure to configure the connector to sample all the necessary data. Nodes and relationship types that are not sampled by the APOC procedures are not included in the temporary schema definition, and therefore are not available in JDBC applications.

Mapping Retrieved Metadata

Neo4j is a graph database and therefore does not contain data in the traditional, relational form. Since traditional JDBC toolsets might not support such datasets, the Simba Neo4j BI Connector generates a schema definition that maps the Neo4j data to a JDBC-compatible format.

When the Simba Neo4j BI Connector generates a schema definition, it performs the following tasks:

1. Retrieves metadata from Simba Neo4j BI Connector through the above-mentioned APOC procedure calls and generates tables for node labels and relationship types based on the retrieved metadata. For more information, see Nodes and Relationships on page 20.
2. Assigns a Neo4j data type to each column.

3. Maps each Neo4j data type to the SQL data type that is best able to represent the greatest number of values.

During this schema generation process, the connector defines data types for each column, but does not change the data types of the individual node or relationship properties in the database. As a result, columns might contain mixed data types. During read operations, values are converted to match the SQL data type of the column so that the connector can work with all the data in the column consistently.

Nodes and Relationships

The Simba Neo4j BI Connector creates tables to allow Neo4j nodes and relationships to be queried through SQL.

Nodes

The connector creates one table for each distinct combination of node labels.

For example, if the data store contains the following nodes:

- Node1, with the label [Alphabet]
- Node2, with the label [Google]
- Node3, with the labels [Alphabet, Google]

Then the connector creates the following tables:

- Alphabet: a table containing information for nodes that only have the label name "Alphabet".
- Alphabet__Google: a table containing information for nodes that only have the label names [Alphabet, Google] or [Google, Alphabet].
- Google: a table containing information for nodes that only have the label name "Google".

Note:

- The connector only creates tables for nodes that have labels.
- The default separator between the node label names is two underscores (__). To use a different separator, set the `LabelSeparator` configuration property. For more information, see LabelSeparator on page 43.

Each node table contains a virtual column named _NodeId_ that contains the ID for the representative node.

Relationships

The connector creates one table for each distinct combination of source label, relationship type, and target label.

For example, if the data store contains a relationship between the source labels Google and Alphabet, the relationship type Companies, and the target label Search, the connector creates the table Google__Alphabet_COMPANIES_Search.

Note:

- The default separator between the node label names is two underscores (__). To use a different separator, set the `LabelSeparator` configuration property.
- The default separator between node label names and relationship names is an underscore (_). To use a different separator, set the `RelNodeSeparator` configuration property.

For more information, see LabelSeparator on page 43 or RelNodeSeparator on page 45.

Each relationship table contains the following virtual columns:

Column name	Content
SourceId	The ID for the source node being connected.
TargetId	The ID for the target node being connected.

Aggregate Function Passdown

The Simba Neo4j BI Connector can pass certain queries down to the Neo4j server for execution. This improves the performance of the connector. Queries that are not passed down are executed by the connector.

Important:

The connector handles SUM(NULL) differently from the Neo4j server.

- The connector returns NULL for SUM(NULL)
- The Neo4j server returns 0 for SUM(NULL)

This might result in a discrepancy between queries that are passed down and queries that are resolved by the connector.

The connector supports a limited aggregate function passdown for simple queries. It enforces the following limitations:

- The supported expressions for the argument of an aggregate function are column references and literals.
- The only supported expressions in the GROUP BY clause are column references.
- Aggregate functions must have 0 or 1 arguments.
- Aggregate functions are only supported in one level of any nested queries.

For example, the following queries can be passed down to the server:

- SELECT C1, C2, AVG(C3) + SUM(C4) FROM T1 GROUP BY C1, C2
- SELECT C1, SUM(1.1) FROM T1 GROUP BY C1
- SELECT SUM(C1) FROM T1
- SELECT SUM(1.1) FROM T1 GROUP BY C1

The following queries are not supported for aggregate function passdown:

- SELECT C1, SUM(C2 + C3) FROM T1 GROUP BY C1
- SELECT C1 + 1.0, SUM(C2) FROM T1 GROUP BY C1 + 1.0
- SELECT COUNT(*) FROM (SELECT C1, COUNT(*) AS test FROM T1 GROUP BY C1) Ta GROUP BY test

Important:

Aggregate values calculated by the server can be different than values calculated by the connector.

For example, the SQL query SELECT AVG(COL1) FROM t1, with COL1 containing the following values: 0.0, null, -0.0001, 4.9999, 99999.9999, 1.0, 1.0, 2.0, 3.0, 4.0, 5.0.

If the connector passes down the aggregation operation performed on COL1 to the Neo4j server, the aggregation result is determined to be 10002.099969999997. However, if the connector resolves the query and does not pass down the aggregation operation, the aggregation result is determined to be 10002.09997.

Join Passdown

The Simba Neo4j BI Connector can pass down join operations to the Neo4j server for execution. This improves the performance of the connector. Queries that are not passed down are executed by the connector.

The connector can pass down queries that involve single columns or literals as the operands.

For example, the following join conditions can be passed down to the server:

- table1.col1 = table2.col1
- table1.col1 > table2.col1 AND table1.col2 LIKE table2.col2
- table1.col1 <> 'value'

The following conditions are not supported for join passdown:

- table1.col1 + table1.col2 < table2.col3
- CONCAT('this', ' value') = table1.col1

Join Operations on Subqueries

The join operation can only be passed down if both operands of the join are individually able to be passed down. If one of the operands contains a clause that is not able to be passed down, then it is not possible to pass down the join operation.

For example, the following query cannot be passed down:
SELECT * FROM (SELECT * FROM table1 WHERE col1 + col2 > 1000) t1 INNER JOIN table2 ON t1.col1=t2.col2

In this example, the "col1 + col2 > 1000" condition cannot be passed down, so the join operation cannot be passed down.

Note:

Subqueries involving TOP, LIMIT, or aggregation operations are also not supported as one of the JOIN operands.

Multiple Joins with AND

When conditions are separated by AND and one condition cannot be passed down, the join operation on remaining conditions are passed down to the Neo4j server while the conditions not passed down are handled by the SQLEngine.

For example, the following query can be fully passed down to the Neo4j server:
SELECT * FROM table1 INNER JOIN table2 ON table1.col1=table2.col1 INNER JOIN table3 ON table2.col1=table2.col1 AND table2.col2 LIKE table3.col2

As another example, the following query, using AND, cannot be fully passed down to the Neo4j server:
SELECT * FROM table1 INNER JOIN table2 ON table1.col1=table2.col1 AND table1.col2+table1.col3 > table2.col2

In this example, the "SELECT * FROM table1 INNER JOIN table2 ON table1.col1=table2.col1" condition is passed down to the Neo4j server, while the "table1.col2+table1.col3 > table2.col2" condition is handled by the SQLEngine.

Important:

When comparing strings with different lengths, the connector handles the "=" condition on strings differently from the Neo4j server. The connector pads the value with whitespaces, while the Neo4j server does not.

For example, when resolving "col1=col2", if the value of col1 is 'ValueTest ' and col2 is 'ValueTest', the connector reports a match, but the Neo4j server does not.

Filter Passdown

The Simba Neo4j BI Connector can pass down filters to the Neo4j server for execution. This improves the performance of the connector. Queries that are not passed down are executed by the connector.

The filter pass down supports conditions that operate on single columns.

For example, the following queries can be passed down to the server:

- SELECT * FROM table WHERE col1 = 'value'
- SELECT * FROM table WHERE col1 > 2000 AND col2 LIKE '%sale%'
- SELECT * FROM table WHERE col1 <> 'value' OR col2 IS NULL

The following queries are not supported for filter passdown:

- SELECT * FROM table WHERE col1 + col2 < 200
- SELECT * FROM table WHERE col1 = CONCAT('this', ' value')

Important:

When comparing strings with different lengths, the connector handles the "=" condition on strings differently from the Neo4j server. The connector pads the value with whitespaces, while the Neo4j server does not.

For example, when resolving "col1=col2", if the value of col1 is 'ValueTest ' and col2 is 'ValueTest', the connector reports a match, but the Neo4j server does not.

Cypher-backed Views

Cypher-backed Views enable users to define views based on a Cypher query. The definitions of the Cypher-backed Views are stored in JSON format in a text file. The file is then passed to the connector via the connection URL. Cypher-backed Views include the metadata and schema for the view as well as the corresponding Cypher query.

To use Cypher-backed Views, in the connection URL, specify the full path to the View Definition file using the `ViewDefinitionFile` property. For details, see ViewDefinitionFile on page 50.

For more information about Cypher-backed Views, see the following sections:

- View Definition File Specification on page 26
- View Definition File Overview on page 26

- Comprehensive View Definition File Example on page 28
- Succinct View Definition File Example on page 36

View Definition File Specification

The View Definition file can define multiple schemas. Using these schemas, users can define the Cypher-backed View tables. By default, the Cypher-backed View tables are categorized under `schema: View`, unless a different name is provided in the View Definition file.

Note:

- The Node and Relationship schemas cannot be used to view Cyper-backed View tables. For details about Node and Relationship schemas, see Catalog and Schema Support on page 18.
- Before connecting to a database using a View Definition file, make sure that the database contains the nodes or relationships present in the Cypher query provided for views.

View Definition File Overview

The View Definition file is a JSON formatted file. The View Definition begins with a JSON object structure containing a single property:

Property Name	JSON Structure	Required	Default Value
Schemas	Array containing schema definitions as array elements.	Yes	N/A

Each schema definition is encapsulated in a JSON object structure with the following properties:

Property Name	JSON Structure	Required	Default Value
Name	String	No	View
Hidden	Boolean	No	False

Property Name	JSON Structure	Required	Default Value
Views	Array containing Cypher-backed View table definitions as array elements for the parent schema.	Yes	N/A

Each Cypher-backed View table definition is encapsulated in a JSON object structure with the following properties:

Property Name	JSON Structure	Required	Default Value
Name	String	Yes	N/A
Hidden	Boolean	No	False
CypherQuery	String	Yes	N/A
Columns	Array containing Cypher-backed View column definitions as array elements for the parent table.	Yes	N/A

Lastly, each Cypher-backed View column definition is encapsulated in a JSON object structure with the following properties:

Property Name	JSON Structure	Required	Default Value
Name	String	Yes	N/A
SourceName	String	No	N/A
Mandatory	Boolean	No	False
Hidden	Boolean	No	False

Property Name	JSON Structure	Required	Default Value
Neo4jType	String or Array containing Neo4j types as JSON string values.	No	String

Comprehensive View Definition File Example

In this example, we asked the connector to expose, via applications, a new Schema named `PlayersBio` and a table named `PlayerProfile`. The `PlayerProfile` table represents the Cypher-backed View provided via the Cypher query present in the `CypherQuery` field below. The columns that exist within `PlayerProfile` are: `_NodeId_`, `Last Name`, `Country`, `Ranking`, and `DOB`.

Note:

The top-level structure needs to have a JSON object structure containing a single key-value pair. The key present in the example below is `Schemas`.

The following example displays the file format and the various properties needed for defining the Cypher-backed View tables:

```
{
  "Schemas": [
    {
        "Name": "PlayersBio",
        "Hidden": false,
        "Views": [
          {
            "Name": "PlayerProfile",
            "CypherQuery": "MATCH (node:Player) RETURN
            ID(node), node.LastName as lastName,
            node.Country, node.Ranking, node.DOB",
            "Columns": [
              {
                "Name": "_NodeId_",
                "SourceName": "ID(node)",
                "Neo4jType": ["Long"],
                "Mandatory": true,
                "Hidden": false
              },
```

```
                {
                  "Name": "Last Name",
                  "SourceName": "lastName",
                  "Neo4jType": ["String"],
                  "Mandatory": false,
                  "Hidden": false
                },
                {
                  "Name": "Country",
                  "SourceName": "node.Country",
                  "Neo4jType": "String",
                  "Mandatory": false,
                  "Hidden": false
                },
                {
                  "Name": "Ranking",
                  "SourceName": "node.Ranking",
                  "Neo4jType": ["String","Long"],
                  "Mandatory": false,
                  "Hidden": false
                },
                {
                  "Name": "DOB",
                  "SourceName": "node.DOB",
                  "Neo4jType": ["Date"],
                  "Mandatory": false,
                  "Hidden": false
                }
              ]
            }
          ]
        }
      ]
    }
```

For information about the key-value pairs in the example above, see the following:

Schemas (ref. - "Schemas": [...])

A JSON array structure within which schema definitions are provided.

Default Value	Data Type	Required
None	Array of JSON objects, where each object contains a schema definition.	Yes

The schema definition present as an array value within `Schemas` contains the following keys:

- Name (ref. - "Name": "PlayerProfile") on page 31
- Hidden (ref. - "Hidden": false) on page 30
- Views (ref. - "Views": [...]) on page 30

Name (ref. - "Name": "PlayersBio")

The name of the view schema which contains Cypher-backed View tables.

Default Value	Data Type	Required
View	String	No

Note:

Duplicate schema names are not allowed by the connector. The matching performed on schema names is case-sensitive, therefore schema names such as `TestView` and `testview` are not considered duplicates.

Hidden (ref. - "Hidden": false)

Indicates whether the defined view schema is displayed in applications.

Default Value	Data Type	Required
False	Boolean	No

Views (ref. - "Views": [...])

A JSON array structure that contains the view table definitions.

Default Value	Data Type	Required
None	Array of JSON objects, where each object contains a view table definition.	Yes

The view table definition contains the following keys:

- Name (ref. - "Name": "PlayerProfile") on page 31
- Hidden (ref. - "Hidden": false) on page 31
- CypherQuery (ref. - "CypherQuery": "MATCH (node:Player)...") on page 31
- Columns (ref. - "Columns": [...]) on page 32

Name (ref. - "Name": "PlayerProfile")

The Name of the Cypher-backed View table that is displayed in applications.

Default Value	Data Type	Required
None	String	Yes

Note:

Duplicate view table names are not allowed by the connector. However, the matching performed on table names is case-sensitive, therefore names such as `TestViewTable` and `testviewtable` are not considered duplicates.

Hidden (ref. - "Hidden": false)

Key to indicate whether the defined view table is displayed in applications.

Default Value	Data Type	Required
False	Boolean	No

CypherQuery (ref. - "CypherQuery": "MATCH (node:Player)...")

The Cypher query used for obtaining results for the Cypher-backed View table.

Default Value	Data Type	Required
None	String	Yes

Note:

- The Cypher query must return only specific properties of the nodes or relationships and not the full nodes or relationships.
- The connector does not validate the Cypher query string, therefore the Cypher query provided via the `CypherQuery` field is executed if the view gets queried. Make sure that the read-only query is provided in the `CypherQuery` field. If not, the data is manipulated when the view is queried.

Columns (ref. - "Columns": [...])

A JSON array structure that contains the view column definitions.

Default Value	Data Type	Required
None	Array of JSON objects, where each object contains a view column definition.	Yes

The view column definition contains the following keys:

- Name (ref. - "Name": "_NodeId_") on page 32
- SourceName (ref. - "SourceName": "ID(node)") on page 33
- Neo4jType (ref. - "Neo4jType": ["Long"]) on page 33
- Mandatory (ref. - "Mandatory": true) on page 34
- Hidden (ref. - "Hidden": false) on page 35

Name (ref. - "Name": "_NodeId_")

The view column name that is displayed in applications. For details on the value specification of `Name` for column when `SourceName` is not provided, see the definition of `SourceName` below.

Default Value	Data Type	Required
None	String	Yes

Note:

Duplicate view column names are not allowed by the connector. However, the matching performed on column names is case-insensitive and therefore names such as `TestCol` and `testcol` are considered duplicates.

SourceName (ref. - "SourceName": "ID(node)")

The Neo4j property name provided verbatim in the Cypher query. The `SourceName` field is used by the connector when retrieving data for the column being queried.

If an alias name is used for the Neo4j property name in the provided Cypher query, the source name must match the alias name. In the example above, this is shown in column definition (`"SourceName": "lastName"`).

If the source name is not provided, the column's name attribute (`"Name"`) has to match the referenced Neo4j property name from the Cypher query so that the connector can locate data for the column when retrieving result set.

Default Value	Data Type	Required
None	String	No

To change the column name displayed by applications of the referenced Neo4j property returned in the view's Cypher query:

- In the `SourceName` field, define the original Neo4j property name returned.
- In the column's `Name` field, define the name you want the application to display.

For example, to change the column name from `ID(node)` to `PlayerID`, define the column's name as `"Name": "PlayerID"` and the column's source name as `"SourceName": "ID(node)"`.

If you do not want to change the column name, do not define `SourceName` and define the column name as `"Name": "ID(node)"`.

Neo4jType (ref. - "Neo4jType": ["Long"])

The Neo4j type for the view column. The Neo4j type can be provided as an individual string value, like `"Neo4jType": "Long"`, or as an array of string values.

If the Neo4j property contains values of different types, the connector handles the data type coercion in case multiple types are provided for a column. For example, if a property contains values of `Date`, `String` and `Long` types, the key value pair for the Neo4j type is `"Neo4jType": ["Date", "String", "Long"]`.

The following Neo4j types can be provided as values to the connector for Cypher-backed Views:

`Boolean`	`BooleanArray`	`ByteArray`
`Long`	`LongArray`	`Integer`
`IntegerArray`	`Double`	`DoubleArray`
`Float`	`String`	`StringArray`
`Point`	`PointArray`	`Date`
`DateArray`	`LocalTime`	`LocalTimeArray`
`Time`	`TimeArray`	`LocalDateTime`
`LocalDateTimeArray`	`DateTime`	`DateTimeArray`
`Duration`	`DurationArray`	

Default Value	Data Type	Required
String	String or Array of string values	No

Note:

Any Neo4j data types that do not have a specific SQL mapping are mapped to SQL_VARCHAR. For details on mapping Neo4j types to SQL and Java types, see Data Types on page 37.

Mandatory (ref. - "Mandatory": true)

Indicates whether a column is nullable or not. `Mandatory:false` indicates `nullability:true`, while `Mandatory:true` indicates `nullability:false`.

Default Value	Data Type	Required
False	Boolean	No

Note:

When specifying whether a column is mandatory or not, check the existing constraint set on the respective property in the Neo4j graph before defining the `Mandatory` field.

For example, suppose a column or property contain nulls and `Mandatory:true` is specified. This is a contradiction because even though the graph does contain nulls for the column or property, by providing `Mandatory:true` instead of `Mandatory:false`, the connector incorrectly imposes `nullability:false`.

Hidden (ref. - "Hidden": false)

Indicates whether the defined view column is displayed by applications.

Default Value	Data Type	Required
False	Boolean	No

For example, you are connecting to a database called "neo4j" whose graph contains a node Player with the following data:

ID(node) : 12345
LastName : 'Federer'
Country : 'Switzerland'
Ranking : 2
DOB : 1981-08-08

Based on the above View Definition file example, the connector creates a table with the following information:

Catalog: neo4j
Schema : PlayersBio
Table : PlayerProfile
Columns: _NodeId_, Last Name, Country, Ranking and DOB

NodeId	Last Name	Country	Ranking	DOB
12345	Federer	Switzerland	2	1981-08-08

Succinct View Definition File Example

Based on the above View Definition file definitions, we can see that certain fields have default values and therefore can be omitted. A minimized version of the above View Definition file looks like the following:

```
{
   "Schemas": [
      {
         "Views": [
            {
             "Name": "PlayerProfile",
             "CypherQuery": "MATCH (node:Player RETURN
             ID(node), node.LastName as lastName,
             node.Country, node.Ranking, node.DOB",
             "Columns": [
                  {
                     "Name": "ID(node)"
                  },
                  {
                     "Name": "lastName"
                  },
                  {
                     "Name": "node.Country"
                  },
                  {
                     "Name": "node.Ranking"
                  },
                  {
                     "Name": "node.DOB"
                  }
               ]
            }
         ]
      }
   ]
}
```

Based on the View Definition file above, the table created by the connector has the following information:

Catalog: neo4j
Schema : View
Table : PlayerProfile
Columns: ID(node), lastName, node.Country, node.Ranking and node.DOB

ID(node)	lastName	node.Country	node.Ranking	node.DOB
12345	Federer	Switzerland	2	1981-08-08

Data Types

The Simba Neo4j BI Connector supports many common data formats, converting between Neo4j data types and SQL data types.

The table below lists the supported data type mappings.

Note:

Any Neo4j data types that do not have a specific SQL mapping are mapped to SQL_VARCHAR.

Neo4j Type	SQL Type	Java Type
Boolean	SQL_BOOLEAN	Boolean
ByteArray	SQL_LONGVARBINARY	Byte[]
Date	SQL_DATE	Date
DateTime	SQL_TIMESTAMP	Timestamp
Duration	SQL_VARCHAR	Varchar
Float	SQL_DOUBLE	Double
Integer	SQL_BIGINT	Long
LocalDateTime	SQL_TIMESTAMP	Timestamp

Neo4j Type	SQL Type	Java Type
LocalTime	SQL_TIME	Time
Point	SQL_VARCHAR	Varchar
String	SQL_VARCHAR	Varchar
Time	SQL_TIME	Time

Connector Configuration Options

Connector Configuration Options lists and describes the properties that you can use to configure the behavior of the Simba Neo4j BI Connector.

You can set configuration properties using the connection URL. For more information, see Building the Connection URL on page 11.

Note:

Property names and values are case-sensitive.

AssumeUTC

Default Value	Data Type	Required
`false`	Boolean	No

Description

This property specifies whether the connector assumes that columns whose Neo4j type is DateTime or Time columns only contain values in the UTC time zone.

- `true`: The connector assumes that DateTime and Time columns only contain values in the UTC time zone.
- `false`: The connector assumes that DateTime and Time columns might contain values in time zones other than UTC.

Auth_Type

Default Value	Data Type	Required
`Basic`	String	No

Description

This property specifies the authentication method to use.

By default, the connection is authenticated by using basic authentication. To disable authentication, set this property to `None`.

ConnectionTimeoutMS

Default Value	Data Type	Required
5000	Integer	No

Description

The amount of time, in milliseconds, that the connector waits when establishing a connection before timing out the connection. If connecting to the server takes longer than the specified amount of time, then the connector stops the connection attempt.

A value of 0 indicates that the connector never times out the connection.

Important:

Setting this property to 0 is not recommended.

DefaultBinaryColumnLength

Default Value	Data Type	Required
32767	Integer	No

Description

The maximum data length for binary columns.

DefaultStringColumnLength

Default Value	Data Type	Required
255	Integer	No

Description

The maximum number of characters that can be contained in string columns.

EnableJavaDriverLogging

Default Value	Data Type	Required
`false`	Boolean	No

Description

This property specifies whether the connector records activity from the Neo4j Java connector. For information about logging, see Configuring Logging on page 15.

- `true`: If logging is enabled, the connector records activity from the Neo4j Java connector in the connector log file.
- `false`: The connector does not record activity from the Java connector.

ExcludeLabels

Default Value	Data Type	Required
None	String	No

Description

A comma-separated list of node labels to exclude during the metadata retrieval of nodes or relationships. Each item in the list must be surrounded by single quotation marks (').

For example:

```
ExcludeLabels='Person','Friend'
```

ExcludeRels

Default Value	Data Type	Required
None	String	No

Description

A comma-separated list of relationship types to exclude during the metadata retrieval of nodes or relationships. Each item in the list must be surrounded by single quotation marks (').

For example:

```
ExcludeRels='HAS','INCLUDES'
```

FetchSize

Default Value	Data Type	Required
1000	Integer	No

Description

The number of results that the connector retrieves in each batch.

A value of -1 indicates that the connector retrieves the entire result set in a single batch.

IncludeLabels

Default Value	Data Type	Required
None	String	No

Description

A comma-separated list of node labels to include during the metadata retrieval of nodes or relationships. Each item in the list must be surrounded by single quotation marks (').

For example:

```
IncludeLabels='Person','Friend'
```

IncludeRels

Default Value	Data Type	Required
None	String	No

Description

A comma-separated list of relationship types to include during the metadata retrieval of nodes or relationships. Each item in the list must be surrounded by single quotation

marks (').

For example:

```
IncludeRels='HAS','INCLUDES'
```

LabelSeparator

Default Value	Data Type	Required
Two underscores (__)	String	No

Description

The separator between labels that the connector uses when creating a table name for a node with multiple labels.

LogLevel

Default Value	Data Type	Required
0	Integer	No

Description

Use this property to enable or disable logging in the connector and to specify the amount of detail included in log files.

Important:

Only enable logging long enough to capture an issue. Logging decreases performance and can consume a large quantity of disk space.

The settings for logging apply to every connection that uses the Simba Neo4j BI Connector, so make sure to disable the feature after you are done using it.

Set the property to one of the following numbers:

- 0: Disable all logging.
- 1: Enable logging on the FATAL level, which logs very severe error events that will lead the connector to abort.

- `2`: Enable logging on the ERROR level, which logs error events that might still allow the connector to continue running.
- `3`: Enable logging on the WARNING level, which logs events that might result in an error if action is not taken.
- `4`: Enable logging on the INFO level, which logs general information that describes the progress of the connector.
- `5`: Enable logging on the DEBUG level, which logs detailed information that is useful for debugging the connector.
- `6`: Enable logging on the TRACE level, which logs all connector activity.

When logging is enabled, the connector produces the following log files in the location specified in the `LogPath` property:

- A `simbaneo4jjdbcdriver.log` file that logs connector activity that is not specific to a connection.
- A `simbaneo4jjdbcdriver_connection_[Number].log` file for each connection made to the database, where *[Number]* is a number that identifies each log file. This file logs connector activity that is specific to the connection.

If the `LogPath` value is invalid, then the connector sends the logged information to the standard output stream (`System.out`).

LogPath

Default Value	Data Type	Required
The current working directory	String	No

Description

The full path to the folder where the connector saves log files when logging is enabled.

Note:

To make sure that the connection URL is compatible with all JDBC applications, it is recommended that you escape the backslashes (`\`) in your file path by typing another backslash.

LabelsSampleSize

Default Value	Data Type	Required
`1000`	Long	No

Description

The interval at which the connector samples nodes when scanning through the data store. For example, if you set this property to `2000`, then the connector samples one node for every 2000 nodes in the data store.

MaxIdentifierLen

Default Value	Data Type	Required
`4096`	Integer	No

Description

This property resets the maximum identifier length for the connection. Maximum possible value is 2147483647.

PWD

Default Value	Data Type	Required
None	String	Yes, unless `Auth_Type` is set to `None`.

Description

The password corresponding to the user name that you provided using the property UID on page 49.

RelNodeSeparator

Default Value	Data Type	Required
Underscore (_)	String	No

Description

The separator between labels and relationship types that the connector uses when creating a table name for a relationship type.

RelsSampleSize

Default Value	Data Type	Required
`100`	Long	No

Description

The maximum number of relations that the connector samples for a given relationship type.

ServerPolicy

Default Value	Data Type	Required
None	String	No

Description

The server policy to use when connecting to the Neo4j server.

Server policies can be defined on the Neo4j server to customize various properties, including routing context. For information on creating and configuring a server policy, see "Multi-data center load balancing" in the Neo4j documentation: https://neo4j.com/docs/operations-manual/4.0/clustering-advanced/multi-data-center/load-balancing/.

SSL

Default Value	Data Type	Required
`false`	String	No

Description

This property specifies whether the connector communicates with the Neo4j server through an SSL-enabled socket.

- `true`: The connector connects to SSL-enabled sockets.
- `2`: The connector connects to SSL-enabled sockets using two-way authentication.
- `false`: The connector does not connect to SSL-enabled sockets.

Note:

SSL is configured independently of authentication. When authentication and SSL are both enabled, the connector performs the specified authentication method over an SSL connection.

sslCustomCertPath

Default Value	Data Type	Required
None	String	Yes, if SSL is enabled and `sslTrustStrategy` is set to `TRUST_CUSTOM_CA_SIGNED_CERTIFICATES`.

Description

The certificate file path containing the trusted certificates to be used when connecting to an SSL-enabled Neo4j instance.

sslTrustStrategy

Default Value	Data Type	Required
`TRUST_SYSTEM_CA_SIGNED_CERTIFICATES`	String	No

Description

This property specifies what certificates the connector trusts when connecting to a Neo4j data store with SSL enabled.

- `TRUST_ALL_CERTIFICATES`: The connector trusts all certificates.

- `TRUST_CUSTOM_CA_SIGNED_CERTIFICATES`: The connector only trusts certificates that have been signed by a trusted authority and are stored in the path specified by `sslCustomCertPath`.
- `TRUST_SYSTEM_CA_SIGNED_CERTIFICATES`: The connector uses the system trust store certificates.

sslVerifyHostname

Default Value	Data Type	Required
`true`	String	No

Description

This property specifies whether the connector performs host name verification when connecting using SSL.

- `true`: The connector performs host name verification when connecting to an SSL-enabled Neo4j instance.
- `false`: The connector does not perform host name verification.

StrictlyUseBoltScheme

Default Value	Data Type	Required
`false`	Boolean	No

Description

This property specifies whether the connector uses the `bolt://` scheme for connection.

- `true`: The connector only attempts to connect using the `bolt://` scheme.
- `false`: The connector attempts to connect using the `neo4j://`scheme. If the connection fails, the connector then attempts to connect using the `bolt://` scheme.

We recommend that you use the `bolt://` scheme to connect to a standalone endpoint, and the `neo4j://` scheme to connect to a clustered endpoint.

The following table describes how the connector behaves in each scenario.

StrictlyUseBoltScheme Value	Server Version	Behavior
`false` (default)	4.x	1. The connector attempts to connect using the `neo4j://` scheme. a. Whether the endpoint is a clustered or a standalone endpoint, the connection is established, as the `neo4j://` scheme is designed to work with any Neo4j 4.x server. 2. The database name is the name provided in the JDBC connection string. However: a. If an invalid database is provided, an error is displayed. b. If no database name is provided, the connector retrieves and uses the database name from the Neo4j server.

UID

Default Value	Data Type	Required
None	String	Yes, unless `Auth_Type` is set to `None`.

Description

The user name that you use to access the Neo4j server.

ViewDefinitionFile

Default Value	Data Type	Required
None	String	No

Description

The full path to the JSON formatted view definition file containing the Cypher-backed View definitions.

Third-Party Trademarks

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Neo4j, Neo Technology, Cypher, Neo4j Bloom, and Neo4j Aura are registered trademarks of Neo4j, Inc.

All other trademarks are trademarks of their respective owners.