Magnitude Simba Neo4j Data Connector for Business Intelligence Tools

ODBC Installation and Configuration Guide

Version 1.0.2

August 2022

Copyright

Contact Us

Magnitude Software, Inc.

www.magnitude.com

About This Guide

Purpose

The *Magnitude Simba Neo4j Data Connector for Business Intelligence Tools ODBC Installation and Configuration Guide* explains how to install and configure the Magnitude Simba Neo4j Data Connector for Business Intelligence Tools. 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, as well as administrators and developers integrating the 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 source to which the Simba Neo4j BI Connector is connecting
- An understanding of the role of ODBC technologies and driver managers in connecting to a data source
- Experience creating and configuring ODBC 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

The Simba Neo4j BI Connector enables Business Intelligence (BI), analytics, and reporting on data. The connector complies with the ODBC 3.80 data standard and adds important functionality such as Unicode, as well as 32- and 64-bit support for high-performance computing environments on all platforms.

ODBC 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 ODBC connector, which connects an application to the database. For more information about ODBC, see *Data Access Standards* on the Simba Technologies website: https://www.simba.com/resources/data-access-standards-glossary. For complete information about the ODBC specification, see the *ODBC API Reference* from the Microsoft documentation: https://docs.microsoft.com/en-us/sql/odbc/reference/syntax/odbc-api-reference.

The *ODBC Installation and Configuration Guide* is suitable for users who are looking to access Neo4j data from their desktop environment. Application developers might also find the information helpful. Refer to your application for details on connecting via ODBC.

Windows Connector

Windows System Requirements

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.*
```

Install the connector on client machines where the application is installed. Before installing the connector, make sure that you have the following:

- Administrator rights on your machine.
- A machine that meets the following system requirements:
  - One of the following operating systems:
    - Windows 10 or 8.1
    - Windows Server 2019, 2016, or 2012
  - 100 MB of available disk space

Before the connector can be used, the Visual C++ Redistributable for Visual Studio 2015 with the same bitness as the connector must also be installed. If you obtained the connector from the Simba website, then your installation of the connector automatically includes this dependency. Otherwise, you must install the redistributable manually. You can download the installation packages for the redistributable at https://www.microsoft.com/en-ca/download/details.aspx?id=48145.

Installing the Connector on Windows

If you did not obtain this connector from the Simba website, you might need to follow a different installation procedure. For more information, see the *Simba OEM ODBC Connectors Installation Guide*.

On 64-bit Windows operating systems, you can execute both 32- and 64-bit applications. However, 64-bit applications must use 64-bit connectors, and 32-bit applications must use 32-bit connectors. Make sure that you use a connector whose bitness matches the bitness of the client application:

- `Simba Neo4j 1.0 32-bit.msi` for 32-bit applications
- `Simba Neo4j 1.0 64-bit.msi` for 64-bit applications

You can install both versions of the connector on the same machine.

To install the Simba Neo4j BI Connector on Windows:

1. Depending on the bitness of your client application, double-click to run **Simba Neo4j 1.0 32-bit.msi** or **Simba Neo4j 1.0 64-bit.msi**.
2. Click **Next**.
3. Select the check box to accept the terms of the License Agreement if you agree, and then click **Next**.
4. To change the installation location, click **Change**, then browse to the desired folder, and then click **OK**. To accept the installation location, click **Next**.
5. Click **Install**.
6. When the installation completes, click **Finish**.
7. If you received a license file through email, then copy the license file into the `\lib` subfolder of the installation folder you selected above. You must have Administrator privileges when changing the contents of this folder.

Creating a Data Source Name on Windows

Typically, after installing the Simba Neo4j BI Connector, you need to create a Data Source Name (DSN).

Alternatively, for information about DSN-less connections, see Using a Connection String on page 35.

To create a Data Source Name on Windows:

1. From the Start menu, go to **ODBC Data Sources**.

   Note:

   Make sure to select the ODBC Data Source Administrator that has the same bitness as the client application that you are using to connect to Neo4j.

2. In the ODBC Data Source Administrator, click the **Drivers** tab, and then scroll down as needed to confirm that the appears in the alphabetical list of ODBC connectors that are installed on your system.
3. Choose one:
   - To create a DSN that only the user currently logged into Windows can use, click the **User DSN** tab.
   - Or, to create a DSN that all users who log into Windows can use, click the **System DSN** tab.

   Note:

   It is recommended that you create a System DSN instead of a User DSN. Some applications load the data using a different user account, and might not be able to detect User DSNs that are created under another user account.

4. Click **Add**.
5. In the Create New Data Source dialog box, select and then click **Finish**. The DSN Setup dialog box opens.
6. In the **Data Source Name** field, type a name for your DSN.
8. Configure the data source name by specifying values for the properties.
9. To configure the sampling process for the connector, click **Advanced Options**. For more information, see Configuring Additional Options on Windows on page 11.
10. To configure logging behavior for the connector, click **Logging Options**. For more information, see Configuring Logging Options on Windows on page 12.
11. To test the connection, click **Test**. Review the results as needed, and then click **OK**.

    Note:

    If the connection fails, then confirm that the settings in the Simba Neo4j ODBC Driver DSN Setup dialog box are correct. Contact your Neo4j server administrator as needed.

12. To save your settings and close the Simba Neo4j ODBC Driver DSN Setup dialog box, click **OK**.
13. To close the ODBC Data Source Administrator, click **OK**.

Configuring Authentication on Windows

Some Neo4j databases are configured to require authentication for access. To connect to a Neo4j server, you must configure the Simba Neo4j BI Connector to use the authentication mechanism that matches the access requirements of the server and provides the necessary credentials.

Basic 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.

To configure basic authentication on Windows:

1. To access authentication options, open the ODBC Data Source Administrator where you created the DSN, select the DSN, and then click **Configure**.
2. In the **Method** drop-down list, select **Basic**.
3. In the **User** field, type your Neo4j user name.
4. In the **Password** field, type the password associated with your Neo4j user name.
5. To encrypt your credentials, click the **Options...** dialog and choose one of the following:
   - If the credentials are used only by the current Windows user, select **Current User Only**.
   - Or, if the credentials are used by all users on the current Windows machine, select **All Users Of This Machine**.

   Note:

   The credentials in the DSN are always encrypted.
6. To save your settings and close the DSN Setup dialog box, click **OK**.

Configuring Additional Options on Windows

You can configure the sampling process used to create the node and relationship tables for the connector.

To configure additional options on Windows:

1. To access advanced options, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **Additional Options**.

2. To configure a table name for a node with multiple labels, under **Labels**:
    a. In the **Separator** field, type a separator that is going between labels.
    b. In the **Sample Size** field, type the interval at which the connector samples nodes when scanning through the data store.
3. To configure a table name for a relationship type, under **Relationship Nodes**:
    a. In the **Separator** field, type a separator that is going between labels and relationship types.
    b. In the **Sample Size** field, type maximum number of relations that the connector samples for a given relationship type.
4. In the **IncludeLabels** field, type a comma-separated list of node labels to include during the metadata retrieval of nodes or relationships.
5. In the **IncludeRels** field, type a comma-separated list of relationship types to include during the metadata retrieval of nodes or relationships.
6. In the **ExcludeLabels** field, type a comma-separated list of node labels to exclude during the metadata retrieval of nodes or relationships.
7. In the **ExcludeRels** field, type a comma-separated list of relationship types to exclude during the metadata retrieval of nodes or relationships.
8. To save your settings and close the Additional Configuration dialog box, click **OK**.
9. To save your settings and close the Simba Neo4j ODBC Driver DSN Setup dialog box, click **OK**.

Configuring Logging Options on Windows

To help troubleshoot issues, you can enable logging. In addition to functionality provided in the Simba Neo4j BI Connector, the ODBC Data Source Administrator provides tracing functionality.

⚠ **Important:**

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

Configuring Connector-wide Logging Options

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. To configure logging for the current connection, see Configuring Logging for the Current Connection on page 14.

To enable connector-wide logging on Windows:

1. To access logging options, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **Logging Options**.
2. From the **Log Level** drop-down list, select the logging level corresponding to the amount of information that you want to include in log files:

Logging Level	Description
OFF	Disables all logging.
FATAL	Logs severe error events that lead the connector to abort.
ERROR	Logs error events that might allow the connector to continue running.
WARNING	Logs events that might result in an error if action is not taken.
INFO	Logs general information that describes the progress of the connector.
DEBUG	Logs detailed information that is useful for debugging the connector.
TRACE	Logs all connector activity.

3. In the **Log Path** field, specify the full path to the folder where you want to save log files.
4. In the **Max Number Files** field, type the maximum number of log files to keep.

Note:

After the maximum number of log files is reached, each time an additional file is created, the connector deletes the oldest log file.

5. In the **Max File Size** field, type the maximum size of each log file in megabytes (MB).

> **Note:**
>
> After the maximum file size is reached, the connector creates a new file and continues logging.

6. Click **OK**.
7. Restart your ODBC application to make sure that the new settings take effect.

The Simba Neo4j BI Connector produces the following log files at the location you specify in the Log Path field:

- A `simbaneo4jodbcdriver.log` file that logs connector activity that is not specific to a connection.
- A `simbaneo4jodbcdriver_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.

To disable connector logging on Windows:

1. Open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then click **Logging Options**.
2. From the **Log Level** drop-down list, select **LOG_OFF**.
3. Click **OK**.
4. Restart your ODBC application to make sure that the new settings take effect.

Configuring Logging for the Current Connection

You can configure logging for the current connection by setting the logging configuration properties in the DSN or in a connection string. For information about the logging configuration properties, see Configuring Logging Options on Windows on page 12. Settings in the connection string take precedence over settings in the DSN, and settings in the DSN take precedence over connector-wide settings.

> **Note:**
>
> If the LogLevel configuration property is passed in via the connection string or DSN, the rest of the logging configurations are read from the connection string or DSN and not from the existing connector-wide logging configuration.

To configure logging properties in the DSN, you must modify the Windows registry. For information about the Windows registry, see the Microsoft Windows documentation.

Important:

Editing the Windows Registry incorrectly can potentially cause serious, system-wide problems that may require re-installing Windows to correct.

To add logging configurations to a DSN on Windows:

1. On the Start screen, type **regedit**, and then click the **regedit** search result.
2. Navigate to the appropriate registry key for the bitness of your connector and your machine:
   - 32-bit System DSNs: **HKEY_LOCAL_ MACHINE\SOFTWARE\WOW6432Node\ODBC\ODBC.INI\*[DSN Name]***
   - 64-bit System DSNs: **HKEY_LOCAL_ MACHINE\SOFTWARE\ODBC\ODBC.INI\*[DSN Name]***
   - 32-bit and 64-bit User DSNs: **HKEY_CURRENT_ USER\SOFTWARE\ODBC\ODBC.INI\*[DSN Name]***
3. For each configuration option that you want to configure for the current connection, create a value by doing the following:
   a. If the key name value does not already exist, create it. Right-click the *[DSN Name]* and then select **New > String Value**, type the key name of the configuration option, and then press **Enter**.
   b. Right-click the key name and then click **Modify**.

      To confirm the key names for each configuration option, .
   c. In the Edit String dialog box, in the **Value Data** field, type the value for the configuration option.
4. Close the Registry Editor.
5. Restart your ODBC application to make sure that the new settings take effect.

Configuring SSL Connections on Windows

If you are connecting to a Neo4j server that has Secure Sockets Layer (SSL) enabled, then you can configure the connector to connect to an SSL-enabled socket and encrypt the connection. When connecting to a server over SSL, the connector uses one-way authentication to verify the identity of the server.

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.

Configuring an SSL Connection without Identity Verification

You can configure a connection that is encrypted by SSL but does not verify the identity of the client or the server.

To configure an SSL connection without verification on Windows:

1. To access the SSL options, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**.
2. Select the **Use SSL** check box.
3. Select the **Allow Self-Signed Certificates** check box.
4. To save your settings and close the Simba Neo4j ODBC Driver DSN Setup dialog box, click **OK**.

Configuring One-way SSL Identity Verification

You can configure one-way verification so that the client verifies the identity of the Neo4j server.

To configure one-way SSL identity verification on Windows:

1. To access the SSL options, open the ODBC Data Source Administrator where you created the DSN, then select the DSN, then click **Configure**, and then select the **SSL** tab.
2. Select the **Use SSL** check box.
3. In the **PEM Certificate File** field, specify the full path of the PEM file containing the certificate for verifying the server.
4. To save your settings and close the Simba Neo4j ODBC Driver DSN Setup dialog box, click **OK**.

Exporting a Data Source Name on Windows

After you configure a DSN, you can export it to be used on other machines. When you export a DSN, all of its configuration settings are saved in a `.sdc` file. You can then distribute the `.sdc` file to other users so that they can import your DSN configuration and use it on their machines.

To export a Data Source Name on Windows:

1. Open the ODBC Data Source Administrator where you created the DSN, select the DSN, click **Configure**, and then click **Logging Options**.
2. Click **Export Configuration**, specify a name and location for the exported DSN, and then click **Save**.

Your DSN is saved as a `.sdc` file in the location that you specified.

Importing a Data Source Name on Windows

You can import a DSN configuration from a `.sdc` file and then use those settings to connect to your data source.

To import a Data Source Name on Windows:

1. Open the ODBC Data Source Administrator where you created the DSN, select the DSN, click **Configure**, and then click **Logging Options**.
2. Click **Import Configuration**, browse to select the `.sdc` file that you want to import the DSN configuration from, and then click **Open**.
3. Click **OK** to close the Logging Options dialog box.

The Simba Neo4j BI Connector DSN Setup dialog box loads the configuration settings from the selected `.sdc` file. You can now save this DSN and use it to connect to your data source.

Verifying the Connector Version Number on Windows

If you need to verify the version of the Simba Neo4j BI Connector that is installed on your Windows machine, you can find the version number in the ODBC Data Source Administrator.

To verify the connector version number on Windows:

1. From the Start menu, go to **ODBC Data Sources**.

   Note:

   Make sure to select the ODBC Data Source Administrator that has the same bitness as the client application that you are using to connect to Neo4j.

2. Click the **Drivers** tab and then find the Simba Neo4j BI Connector in the list of ODBC connectors that are installed on your system. The version number is displayed in the **Version** column.

macOS Connector

macOS System Requirements

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.*
```

Install the connector on client machines where the application is installed. Each client machine that you install the connector on must meet the following minimum system requirements:

- One of the following macOS versions:
  - macOS 10.14
  - macOS 10.15
  - macOS 11
- 150MB of available disk space
- One of the following ODBC driver managers installed:
  - iODBC 3.52.9 or later
  - unixODBC 2.2.14 or later

Installing the Connector on macOS

If you did not obtain this connector from the Simba website, you might need to follow a different installation procedure. For more information, see the *Simba OEM ODBC Connectors Installation Guide*.

The Simba Neo4j BI Connector is available for macOS as a `.dmg` file named `Simba Neo4j 1.0.dmg`. The connector supports both 32- and 64-bit client applications.

To install the Simba Neo4j BI Connector on macOS:

1. Double-click **Simba Neo4j 1.0.dmg** to mount the disk image.
2. Double-click **Simba Neo4j 1.0.pkg** to run the installer.
3. In the installer, click **Continue**.
4. On the Software License Agreement screen, click **Continue**, and when the prompt appears, click **Agree** if you agree to the terms of the License Agreement.
5. Optionally, to change the installation location, click **Change Install Location**, then select the desired location, and then click **Continue**.

   Note:

   By default, the connector files are installed in the `/Library/simba/neo4j` directory.

6. To accept the installation location and begin the installation, click **Install**.
7. When the installation completes, click **Close**.
8. If you received a license file through email, then copy the license file into the `/lib` subfolder in the connector installation directory. You must have root privileges when changing the contents of this folder.

   For example, if you installed the connector to the default location, you would copy the license file into the `/Library/simba/neo4j/lib` folder.

Next, configure the environment variables on your machine to make sure that the ODBC driver manager can work with the connector. For more information, see Configuring the ODBC Driver Manager on Non-Windows Machines on page 22.

Verifying the Connector Version Number on macOS

If you need to verify the version of the Simba Neo4j BI Connector that is installed on your macOS machine, you can query the version number through the Terminal.

To verify the connector version number on macOS:

- At the Terminal, run the following command:

```
pkgutil --info simba.neo4jodbc
```

The command returns information about the Simba Neo4j BI Connector that is installed on your machine, including the version number.

Linux Connector

Linux System Requirements

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).

Install the connector on client machines where the application is installed. Each client machine that you install the connector on must meet the following minimum system requirements:

- One of the following distributions:
  - Red Hat® Enterprise Linux® (RHEL) 7 or 8
  - CentOS 7 or 8
  - SUSE Linux Enterprise Server (SLES) 12 or 15
  - Ubuntu 18.04 or 20.04
- One of the following ODBC driver managers installed:
  - iODBC 3.52.9 or later
  - unixODBC 2.2.14 or later

To install the connector, you must have root access on the machine.

Installing the Connector Using the Tarball Package

If you did not obtain this connector from the Simba website, you might need to follow a different installation procedure. For more information, see the *Simba OEM ODBC Connectors Installation Guide*.

The Simba Neo4j BI Connector is available as a tarball package named `SimbaNeo4jODBC-[Version].[Release]-Linux.tar.gz`, where *[Version]* is the version number of the connector and *[Release]* is the release number for this version of the connector. The package contains both the 32-bit and 64-bit versions of the connector.

On 64-bit editions of Linux, you can execute both 32- and 64-bit applications. However, 64-bit applications must use 64-bit connectors, and 32-bit applications must use 32-bit connectors. Make sure that you use a connector whose bitness matches the bitness of the client application. You can install both versions of the connector on the same machine.

To install the connector using the tarball package:

1. Log in as the root user, and then navigate to the folder containing the tarball package.
2. Run the following command to extract the package and install the connector:

   ```
   tar -zxvf [TarballName]
   ```

   Where *`[TarballName]`* is the name of the tarball package containing the connector.

   The Simba Neo4j BI Connector files are installed in the `/opt/simba/neo4j` directory.
3. If you received a license file through email, then copy the license file into the `opt/simba/neo4j/lib/32` or `opt/simba/neo4j/lib/64` folder, depending on the version of the connector that you installed.

Next, configure the environment variables on your machine to make sure that the ODBC driver manager can work with the connector. For more information, see Configuring the ODBC Driver Manager on Non-Windows Machines on page 22.

Verifying the Connector Version Number on Linux

If you need to verify the version of the Simba Neo4j BI Connector that is installed on your Linux machine, you can search the connector's binary file for version number information.

To verify the driver version number on Linux using the binary file:

1. Navigate to the `/lib` subfolder in your connectorinstallation directory. By default, the path to this directory is: `/opt/simba/neo4j/lib`.
2. Open the driver's `.so` binary file in a text editor, and search for the text `$driver_version_sb$:`. The connector's version number is listed after this text.

Configuring the ODBC Driver Manager on Non-Windows Machines

To make sure that the ODBC driver manager on your machine is configured to work with the Simba Neo4j BI Connector, do the following:

- Set the library path environment variable to make sure that your machine uses the correct ODBC driver manager. For more information, see Specifying ODBC Driver Managers on Non-Windows Machines on page 22.
- If the connector configuration files are not stored in the default locations expected by the ODBC driver manager, then set environment variables to make sure that the driver manager locates and uses those files. For more information, see .

After configuring the ODBC driver manager, you can configure a connection and access your data store through the connector.

Specifying ODBC Driver Managers on Non-Windows Machines

You need to make sure that your machine uses the correct ODBC driver manager to load the connector. To do this, set the library path environment variable.

macOS

If you are using a macOS machine, then set the DYLD_LIBRARY_PATH environment variable to include the paths to the ODBC driver manager libraries. For example, if the libraries are installed in `/usr/local/lib`, then run the following command to set DYLD_LIBRARY_PATH for the current user session:

```
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:/usr/local/lib
```

For information about setting an environment variable permanently, refer to the macOS shell documentation.

Linux

If you are using a Linux machine, then set the LD_LIBRARY_PATH environment variable to include the paths to the ODBC driver manager libraries. For example, if the libraries are installed in `/usr/local/lib`, then run the following command to set LD_LIBRARY_PATH for the current user session:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

For information about setting an environment variable permanently, refer to the Linux shell documentation.

Specifying the Locations of the Connector Configuration Files

By default, ODBC driver managers are configured to use hidden versions of the `odbc.ini` and `odbcinst.ini` configuration files (named `.odbc.ini` and `.odbcinst.ini`) located in the home directory, as well as the `simba.neo4jodbc.ini` file in the `lib` subfolder of the connector installation directory. If you store these configuration files elsewhere, then you must set the environment variables described below so that the driver manager can locate the files.

If you are using iODBC, do the following:

- Set ODBCINI to the full path and file name of the `odbc.ini` file.
- Set ODBCINSTINI to the full path and file name of the `odbcinst.ini` file.
- Set SIMBANEO4JODBCINI to the full path and file name of the `simba.neo4jodbc.ini` file.

If you are using unixODBC, do the following:

- Set ODBCINI to the full path and file name of the `odbc.ini` file.
- Set ODBCSYSINI to the full path of the directory that contains the `odbcinst.ini` file.
- Set SIMBANEO4JODBCINI to the full path and file name of the `simba.neo4jodbc.ini` file.

For example, if your `odbc.ini` and `odbcinst.ini` files are located in `/usr/local/odbc` and your `simba.neo4jodbc.ini` file is located in `/etc`, then set the environment variables as follows:

For iODBC:

```
export ODBCINI=/usr/local/odbc/odbc.ini
export ODBCINSTINI=/usr/local/odbc/odbcinst.ini
export SIMBANEO4JODBCINI=/etc/simba.neo4jodbc.ini
```

For unixODBC:

```
export ODBCINI=/usr/local/odbc/odbc.ini
export ODBCSYSINI=/usr/local/odbc
export SIMBANEO4JODBCINI=/etc/simba.neo4jodbc.ini
```

To locate the `simba.neo4jodbc.ini` file, the connector uses the following search order:

1. If the SIMBANEO4JODBCINI environment variable is defined, then the connector searches for the file specified by the environment variable.
2. The connector searches the directory that contains the connector library files for a file named `simba.neo4jodbc.ini`.
3. The connector searches the current working directory of the application for a file named `simba.neo4jodbc.ini`.
4. The connector searches the home directory for a hidden file named `.simba.neo4jodbc.ini` (prefixed with a period).
5. The connector searches the `/etc` directory for a file named `simba.neo4jodbc.ini`.

Configuring ODBC Connections on a Non-Windows Machine

The following sections describe how to configure ODBC connections when using the Simba Neo4j BI Connector on non-Windows platforms:

Creating a Data Source Name on a Non-Windows Machine

When connecting to your data store using a DSN, you only need to configure the `odbc.ini` file. Set the properties in the `odbc.ini` file to create a DSN that specifies the connection information for your data store. For information about configuring a DSN-less connection instead, see Configuring a DSN-less Connection on a Non-Windows Machine on page 27.

If your machine is already configured to use an existing `odbc.ini` file, then update that file by adding the settings described below. Otherwise, copy the `odbc.ini` file from the `Setup` subfolder in the driver installation directory to the home directory, and then update the file as described below.

To create a Data Source Name on a non-Windows machine:

1. In a text editor, open the `odbc.ini` configuration file.

   Note:

   If you are using a hidden copy of the `odbc.ini` file, you can remove the period (`.`) from the start of the file name to make the file visible while you are editing it.

2. In the `[ODBC Data Sources]` section, add a new entry by typing a name for the DSN, an equal sign (=), and then the name of the driver.

   For example, on a macOS machine:

   ```
   [ODBC Data Sources]
   Sample DSN=Simba Neo4j BI Connector
   ```

   As another example, for a 32-bit driver on a Linux machine:

   ```
   [ODBC Data Sources]
   Sample DSN=Simba Neo4j BI Connector 32-bit
   ```

3. Create a section that has the same name as your DSN, and then specify configuration options as key-value pairs in the section:

   a. Set the `Driver` property to the full path of the driver library file that matches the bitness of the application.

      For example, on a macOS machine:

      ```
      Driver=/Library/simba/neo4j/lib/libneo4jodbc_
      sbu.dylib
      ```

      As another example, for a 32-bit driver on a Linux machine:

      ```
      Driver=/opt/simba/neo4j/lib/32/libneo4jodbc_sb32.so
      ```

   b. Set the `Host` property to the IP address or host name of the server, and then set the `Port` property to the number of the TCP port that the server uses to listen for client connections.

      For example:

      ```
      Host=192.168.222.160
      Port=31010
      ```

   c. Optionally, set additional key-value pairs as needed to specify other optional connection settings. For detailed information about all the configuration options supported by the Simba Neo4j BI Connector, see Connector Configuration Properties on page 58.

   d. If authentication is required to access the server, then specify the authentication mechanism and your credentials. For more information, see Configuring Authentication on a Non-Windows Machine on page 29.

   e. Optionally, if you want to connect to the server through SSL, then enable SSL and specify the certificate information. For more information, see Configuring SSL Verification on a Non-Windows Machine on page 32.

4. Save the `odbc.ini` configuration file.

   Note:

   If you are storing this file in its default location in the home directory, then prefix the file name with a period (`.`) so that the file becomes hidden. If you are storing this file in another location, then save it as a non-hidden file (without the prefix), and make sure that the ODBCINI environment variable specifies the location. For more information, see Specifying the Locations of the Connector Configuration Files on page 23.

For example, the following is an `odbc.ini` configuration file for macOS containing a DSN that connects to Neo4j without authentication:

```
[ODBC Data Sources]
Sample DSN=Simba Neo4j BI Connector
[Sample DSN]
Driver=/Library/simba/neo4j/lib/libneo4jodbc_sbu.dylib
Host=192.168.222.160
Port=31010
```

As another example, the following is an `odbc.ini` configuration file for a 32-bit driver on a Linux machine, containing a DSN that connects to Neo4j without authentication:

```
[ODBC Data Sources]
Sample DSN=Simba Neo4j BI Connector 32-bit
[Sample DSN]
Driver=/opt/simba/neo4j/lib/32/libneo4jodbc_sb32.so
Host=192.168.222.160
Port=31010
```

You can now use the DSN in an application to connect to the data store.

Configuring a DSN-less Connection on a Non-Windows Machine

To connect to your data store through a DSN-less connection, you need to define the connector in the `odbcinst.ini` file and then provide a DSN-less connection string in your application.

If your machine is already configured to use an existing `odbcinst.ini` file, then update that file by adding the settings described below. Otherwise, copy the `odbcinst.ini` file from the `Setup` subfolder in the connector installation directory to the home directory, and then update the file as described below.

To define a connector on a non-Windows machine:

1. In a text editor, open the `odbcinst.ini` configuration file.

   Note:

   If you are using a hidden copy of the `odbcinst.ini` file, you can remove the period (`.`) from the start of the file name to make the file visible while you are editing it.

2. In the `[ODBC Drivers]` section, add a new entry by typing a name for the connector, an equal sign (=), and then `Installed`.

   For example:

   ```
   [ODBC Drivers]
   Simba Neo4j BI Connector=Installed
   ```

3. Create a section that has the same name as the connector (as specified in the previous step), and then specify the following configuration options as key-value pairs in the section:

   a. Set the `Driver` property to the full path of the connector library file that matches the bitness of the application.

      For example, on a macOS machine:

      ```
      Driver=/Library/simba/neo4j/lib/libneo4jodbc_
      sbu.dylib
      ```

      As another example, for a 32-bit connector on a Linux machine:

      ```
      Driver=/opt/simba/neo4j/lib/32/libneo4jodbc_sb32.so
      ```

   b. Optionally, set the `Description` property to a description of the connector.

      For example:

      ```
      Description=Simba Neo4j BI Connector
      ```

4. Save the `odbcinst.ini` configuration file.

   Note:

   If you are storing this file in its default location in the home directory, then prefix the file name with a period (`.`) so that the file becomes hidden. If you are storing this file in another location, then save it as a non-hidden file (without the prefix), and make sure that the ODBCINSTINI or ODBCSYSINI environment variable specifies the location. For more information, see Specifying the Locations of the Connector Configuration Files on page 23.

For example, the following is an `odbcinst.ini` configuration file for macOS:

```
[ODBC Drivers]
Simba Neo4j BI Connector=Installed
```

```
[Simba Neo4j BI Connector]
Description=Simba Neo4j BI Connector
Driver=/Library/simba/neo4j/lib/libneo4jodbc_sbu.dylib
```

As another example, the following is an `odbcinst.ini` configuration file for both the 32- and 64-bit connectors on Linux:

```
[ODBC Drivers]
Simba Neo4j BI Connector 32-bit=Installed
Simba Neo4j BI Connector 64-bit=Installed
[ 32-bit]
Description=Simba Neo4j BI Connector (32-bit)
Driver=/opt/simba/neo4j/lib/32/libneo4jodbc_sb32.so
[Simba Neo4j BI Connector 64-bit]
Description=Simba Neo4j BI Connector (64-bit)
Driver=/opt/simba/neo4j/lib/64/libneo4jodbc_sb64.so
```

You can now connect to your data store by providing your application with a connection string where the `Driver` property is set to the connector name specified in the `odbcinst.ini` file, and all the other necessary connection properties are also set. For more information, see "DSN-less Connection String Examples" in Using a Connection String on page 35.

For instructions about configuring specific connection features, see the following:

- Configuring Authentication on a Non-Windows Machine on page 29
- Configuring SSL Verification on a Non-Windows Machine on page 32

For detailed information about all the connection properties that the connector supports, see Connector Configuration Properties on page 58.

Configuring Authentication on a Non-Windows Machine

Some Neo4j databases require authentication. You can configure the Simba Neo4j BI Connector to provide your credentials and authenticate the connection to the database using Basic Authentication.

You can set the connection properties described below in a connection string or in a DSN (in the `odbc.ini` file). Settings in the connection string take precedence over settings in the DSN.

Basic 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.

To configure Basic authentication:

1. Set the `Auth_Type` property to `Basic`.
2. Set the `UID` property to an appropriate user name for accessing the Neo4j database.
3. Set the `PWD` property to the password corresponding to the user name you typed above.

Configuring Logging Options on a Non-Windows Machine

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.

You can set the connection properties described below in a connection string, in a DSN (in the `odbc.ini` file), or as a connector-wide setting (in the `simba.neo4jodbc.ini` file). Settings in the connection string take precedence over settings in the DSN, and settings in the DSN take precedence over connector-wide settings.

To enable logging on a non-Windows machine:

1. Open the `simba.neo4jodbc.ini` configuration file in a text editor.
2. To specify the level of information to include in log files, set the `LogLevel` property to one of the following numbers:

LogLevel Value	Description
`0`	Disables all logging.
`1`	Logs severe error events that lead the connector to abort.
`2`	Logs error events that might allow the connector to continue running.

LogLevel Value	Description
`3`	Logs events that might result in an error if action is not taken.
`4`	Logs general information that describes the progress of the connector.
`5`	Logs detailed information that is useful for debugging the connector.
`6`	Logs all connector activity.

3. Set the `LogPath` key to the full path to the folder where you want to save log files.
4. Set the `LogFileCount` key to the maximum number of log files to keep.

   Note:

   After the maximum number of log files is reached, each time an additional file is created, the connector deletes the oldest log file.

5. Set the `LogFileSize` key to the maximum size of each log file in bytes.

   Note:

   After the maximum file size is reached, the connector creates a new file and continues logging.

6. Save the `simba.neo4jodbc.ini` configuration file.
7. Restart your ODBC application to make sure that the new settings take effect.

The Simba Neo4j BI Connector produces the following log files at the location you specify using the `LogPath` key:

- A `simbaneo4jodbcdriver.log` file that logs connector activity that is not specific to a connection.
- A `simbaneo4jodbcdriver_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.

To disable logging on a non-Windows machine:

1. Set the `LogLevel` key to `0`.
2. Save the `simba.neo4jodbc.ini` configuration file.
3. Restart your ODBC application to make sure that the new settings take effect.

Configuring SSL Verification on a Non-Windows Machine

If you are connecting to a Neo4j server that has Secure Sockets Layer (SSL) enabled, then 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.

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.

You can set the connection properties described below in a connection string or in a DSN (in the `odbc.ini` file). Settings in the connection string take precedence over settings in the DSN.

Configuring an SSL Connection without Identity Verification

You can configure a connection that is encrypted by SSL but does not verify the identity of the client or the server.

To configure an SSL Connection without Identity Verification on a non-Windows machine:

1. Set the `SSL` property to `1`.
2. Set the `AllowSelfSignedServerCert` property to `1`.

Configuring One-way SSL Identity Verification

You can configure one-way verification so that the client verifies the identity of the Neo4j server.

To configure one-way SSL verification on a non-Windows machine:

1. Set the `SSL` property to `1`.
2. Set the `TrustedCerts` property to the full path of the `.pem` file containing the certificate for verifying the server.

Testing the Connection on a Non-Windows Machine

To test the connection, you can use an ODBC-enabled client application. For a basic connection test, you can also use the test utilities that are packaged with your driver manager installation. For example, the iODBC driver manager includes simple utilities called iodbctest and iodbctestw. Similarly, the unixODBC driver manager includes simple utilities called isql and iusql.

Using the iODBC Driver Manager

You can use the iodbctest and iodbctestw utilities to establish a test connection with your connector. Use iodbctest to test how your connector works with an ANSI application, or use iodbctestw to test how your connector works with a Unicode application.

Note:

There are 32-bit and 64-bit installations of the iODBC driver manager available. If you have only one or the other installed, then the appropriate version of iodbctest (or iodbctestw) is available. However, if you have both 32- and 64-bit versions installed, then you need to make sure that you are running the version from the correct installation directory.

For more information about using the iODBC driver manager, see http://www.iodbc.org.

To test your connection using the iODBC driver manager:

1. Run **iodbctest** or **iodbctestw**.
2. Optionally, if you do not remember the DSN, then type a question mark (`?`) to see a list of available DSNs.
3. Type the connection string for connecting to your data store, and then press ENTER. For more information, see .

If the connection is successful, then the `SQL>` prompt appears.

Using the unixODBC Driver Manager

You can use the isql and iusql utilities to establish a test connection with your connector and your DSN. isql and iusql can only be used to test connections that use a DSN. Use isql to test how your connector works with an ANSI application, or use iusql to test how your connector works with a Unicode application.

Note:

There are 32-bit and 64-bit installations of the unixODBC driver manager available. If you have only one or the other installed, then the appropriate version of isql (or iusql) is available. However, if you have both 32- and 64-bit versions installed, then you need to make sure that you are running the version from the correct installation directory.

For more information about using the unixODBC driver manager, see http://www.unixodbc.org.

To test your connection using the unixODBC driver manager:

- Run isql or iusql by using the corresponding syntax:
  - `isql` *`[DataSourceName]`*
  - `iusql` *`[DataSourceName]`*

`[DataSourceName]` is the DSN that you are using for the connection.

If the connection is successful, then the `SQL>` prompt appears.

Note:

For information about the available options, run isql or iusql without providing a DSN.

Using a Connection String

For some applications, you might need to use a connection string to connect to your data source. For detailed information about how to use a connection string in an ODBC application, refer to the documentation for the application that you are using.

The connection strings in the following sections are examples showing the minimum set of connection attributes that you must specify to successfully connect to the data source. Depending on the configuration of the data source and the type of connection you are working with, you might need to specify additional connection attributes. For detailed information about all the attributes that you can use in the connection string, see Connector Configuration Properties on page 58.

⚠ **Important:**

When connecting to the data store using a DSN-less connection string, the connector does not encrypt your credentials.

DSN Connection String Example

The following is an example of a connection string for a connection that uses a DSN:

```
DSN=[DataSourceName]
```

[DataSourceName] is the DSN that you are using for the connection.

You can set additional configuration options by appending key-value pairs to the connection string. Configuration options that are passed in using a connection string take precedence over configuration options that are set in the DSN.

DSN-less Connection String Examples

Some applications provide support for connecting to a data source using a driver without a DSN. To connect to a data source without using a DSN, use a connection string instead.

The placeholders in the examples are defined as follows, in alphabetical order:

- *[DatabaseName]* is the name of the database service.
- *[PortNumber]* is the number of the port that the Neo4j server uses to listen for client connections.
- *[Server]* is the IP address or host name of the Neo4j server to which you are connecting.

- *[YourPassword]* is the password corresponding to your user name.
- *[YourUserName]* is the user name that you use to access the Neo4j server.

Connecting to a Standard Neo4j Server Without Authentication

The following is the format of a DSN-less connection string for a standard connection to a Neo4j server that does not require authentication:

```
Driver=Simba Neo4j ODBC Driver;Host=[Server];
Port=[PortNumber];Database=[DatabaseName];Auth_Type=None
```

For example:

```
Driver=Simba Neo4j ODBC
Driver;Host=192.168.222.160;Port=31010;Database=neo4j;Auth_
Type=None
```

Connecting to a Standard Neo4j Server that requires Basic Authentication

The following is the format of a DSN-less connection string for a standard connection to a Neo4j server that requires Basic authentication:

```
Driver=Simba Neo4j ODBC Driver;Port=[PortNumber];Database=
[DatabaseName];Auth_Type=Basic;UID=[YourUserName];PWD=
[YourPassword];
```

For example:

```
Driver=Simba Neo4j ODBC
Driver;Host=192.168.222.160;Port=31010;Database=neo4j;Auth_
Type=Basic;UID=simba;PWD=simba;
```

Features

For more information on the features of the Simba Neo4j BI Connector, see the following:

- Security and Authentication on page 37
- SQL Connector on page 38
- Catalog and Schema Support on page 38
- Schema Definition on page 38
- Nodes and Relationships on page 40
- Aggregate Function Passdown on page 41
- Join Passdown on page 42
- Filter Passdown on page 44
- Cypher-backed Views on page 45
- Data Types on page 56

Security and Authentication

Note:

In this documentation, "SSL" refers to both TLS (Transport Layer Security) and SSL (Secure Sockets Layer). The connector supports . The SSL version used for the connection is the highest version that is supported by both the connector and the server.

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 Windows on page 11 or Configuring Authentication on a Non-Windows Machine on page 29. 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. For information about configuring SSL connections, see Configuring SSL Connections on Windows on page 15 or Configuring SSL Verification on a Non-Windows Machine on page 32.

SQL Connector

The SQL Connector feature of the connector allows applications to use normal SQL queries against Neo4j, translating standard SQL-92 queries into equivalent Neo4j API calls. This translation allows standard queries that BI tools execute to run against your Neo4j instance.

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 ODBC 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 40.

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 ODBC applications.

Mapping Retrieved Metadata

Neo4j is a graph database and therefore does not contain data in the traditional, relational form. Since traditional ODBC toolsets might not support such datasets, the Simba Neo4j BI Connector generates a schema definition that maps the Neo4j data to a ODBC-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 40.
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 61.

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 61 or RelNodeSeparator on page 64.

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 string. 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 string, specify the full path to the View Definition file using the `ViewDefinitionFile` property. For details, see ViewDefinitionFile on page 65.

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

- View Definition File Specification on page 45
- View Definition File Overview on page 45
- Comprehensive View Definition File Example on page 47
- Succinct View Definition File Example on page 55

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 38.
- 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
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
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 50
- Hidden (ref. - "Hidden": false) on page 49
- Views (ref. - "Views": [...]) on page 50

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 50
- Hidden (ref. - "Hidden": false) on page 50
- CypherQuery (ref. - "CypherQuery": "MATCH (node:Player)...") on page 51
- Columns (ref. - "Columns": [...]) on page 51

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.

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 52
- SourceName (ref. - "SourceName": "ID(node)") on page 52
- Neo4jType (ref. - "Neo4jType": ["Long"]) on page 53
- Mandatory (ref. - "Mandatory": true) on page 54
- Hidden (ref. - "Hidden": false) on page 54

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 types, see Data Types on page 56.

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:

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

```
{
   "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"
                  }
               ]
            }
         ]
      }
   ]
}
```

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
Boolean	SQL_BIT
ByteArray	SQL_LONGVARBINARY

Neo4j Type	SQL Type
Date	SQL_DATE
DateTime	SQL_TIMESTAMP
Duration	SQL_VARCHAR
Float	SQL_DOUBLE
Integer	SQL_BIGINT
LocalDateTime	SQL_TIMESTAMP
LocalTime	SQL_TIME
Point	SQL_VARCHAR
String	SQL_VARCHAR
Time	SQL_TIME

Connector Configuration Properties

Connector Configuration Options lists the configuration options available in the Simba Neo4j BI Connector alphabetically by field or button label. Options having only key names, that is, not appearing in the user interface of the connector, are listed alphabetically by key name.

When creating or configuring a connection from , the fields and buttons described below are available in the following dialog boxes:

- Logging Options

When using a connection string, use the key names provided below.

Configuration Options Appearing in the User Interface

The following configuration options are accessible via the Windows user interface for the Simba Neo4j BI Connector, or via the key name when using a connection string or configuring a connection from a Linux or macOS computer:

- Allow Self-Signed Server Certificate on page 58
- Auth_Type on page 59
- Database on page 59
- ExcludeLabels on page 59
- ExcludeRels on page 60
- Host on page 60
- IncludeLabels on page 60
- IncludeRels on page 61
- LabelSeparator on page 61
- LabelsSampleSize on page 61
- Log Level on page 62
- Log Path on page 62
- Max File Size on page 63
- Max Number Files on page 63
- Password on page 63
- PEM Key File on page 64
- Port on page 64
- RelNodeSeparator on page 64
- RelsSampleSize on page 64
- Enable SSL on page 65
- User on page 65
- ViewDefinitionFile on page 65

Allow Self-Signed Server Certificate

Key Name	Default Value	Required
`AllowSelfSigned ServerCert`	Clear (`0`)	No

Description

This option specifies whether the connector allows a connection to a Neo4j server that uses a self-signed certificate.

- Enabled (`1`): The connector authenticates the Neo4j server even if the server is using a self-signed certificate.
- Disabled (`0`): The connector does not allow self-signed certificates from the server.

Note:

This setting is applicable only when SSL is enabled.

Auth_Type

Key Name	Default Value	Required
`Auth_Type`	`Basic`	No

Description

This option specifies the authentication method to use.

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

Database

Key Name	Default Value	Required
`Database`	Retrieved from server.	No

Description

The name of the Neo4j database that you want to access.

ExcludeLabels

Key Name	Default Value	Required
`ExcludeLabels`	None	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

Key Name	Default Value	Required
`ExcludeRels`	None	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'
```

Host

Key Name	Default Value	Required
`Host`	None	Yes

Description

The IP address or host name of the Neo4j server.

IncludeLabels

Key Name	Default Value	Required
`IncludeLabels`	None	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

Key Name	Default Value	Required
`IncludeRels`	None	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

Key Name	Default Value	Required
`LabelSeparator`	Two underscores (__)	No

Description

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

LabelsSampleSize

Key Name	Default Value	Required
`LabelSampleSize`	`1000`	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.

Log Level

Key Name	Default Value	Required
`LogLevel`	OFF (`0`)	No

Description

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

Set the property to one of the following values:

- OFF (`0`): Disable all logging.
- FATAL (`1`): Logs severe error events that lead the connector to abort.
- ERROR (`2`): Logs error events that might allow the connector to continue running.
- WARNING (`3`): Logs events that might result in an error if action is not taken.
- INFO (`4`): Logs general information that describes the progress of the connector.
- DEBUG (`5`): Logs detailed information that is useful for debugging the connector.
- TRACE (`6`): Logs all connector activity.

- A `simbaneo4jodbcdriver.log` file that logs connector activity that is not specific to a connection.
- A `simbaneo4jodbcdriver_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.

Log Path

Key Name	Default Value	Required
`LogPath`	None	Yes, if logging is enabled.

Description

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

Max File Size

Key Name	Default Value	Required
`LogFileSize`	`20971520`	No

Description

The maximum size of each log file in bytes. After the maximum file size is reached, the connector creates a new file and continues logging.

If this property is set using the Windows UI, the entered value is converted from megabytes (MB) to bytes before being set.

Max Number Files

Key Name	Default Value	Required
`LogFileCount`	`50`	No

Description

The maximum number of log files to keep. After the maximum number of log files is reached, each time an additional file is created, the connector deletes the oldest log file.

Password

Key Name	Default Value	Required
`PWD`	None	Yes, unless `Auth_Type` is set to `None`.

Description

The password corresponding to the user name that you provided in the User Name field (the `UID` key).

PEM Key File

Key Name	Default Value	Required
`TrustedCerts`	None	No

Description

The full path of the `.pem` file containing the certificate for verifying the server.

Port

Key Name	Default Value	Required
`Port`	`7687`	Yes

Description

The number of the TCP port that the Neo4j server uses to listen for client connections.

RelNodeSeparator

Key Name	Default Value	Required
`RelNodeSeparator`	Underscore (_)	No

Description

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

RelsSampleSize

Key Name	Default Value	Required
`RelsSampleSize`	`100`	No

Description

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

Enable SSL

Key Name	Default Value	Required
`SSL`	Clear (`0`)	No

Description

This option specifies whether the client uses an SSL encrypted connection to communicate with the Neo4j server.

- Enabled (`1`): The client communicates with the Neo4j server using SSL.
- Disabled (`0`): SSL is disabled.

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

User

Key Name	Default Value	Required
`UID`	None	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.

Configuration Options Having Only Key Names

The following configuration options do not appear in the Windows user interface for the Simba Neo4j BI Connector. They are accessible only when you use a connection string or configure a connection on macOS or Linux.

- Driver on page 66
- StrictlyUseBoltScheme on page 66

Driver

Key Name	Default Value	Required
`Driver`	`Simba Neo4j ODBC Driver` when installed on Windows, or the absolute path of the connector shared object file when installed on a non-Windows machine.	Yes

Description

On Windows, the name of the installed connector (`Simba Neo4j ODBC Driver;`).

On other platforms, the name of the installed connector as specified in `odbcinst.ini`, or the absolute path of the connector shared object file.

StrictlyUseBoltScheme

Key Name	Default Value	Required
`StrictlyUseBoltScheme`	`0`	No

Description

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

- `1`: The connector only attempts to connect using the bolt:// scheme.
- `0`: 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.

Third-Party Trademarks