Progress DataDirect for JDBC for Apache Spark SQL

User's Guide

Release 6.0.1

Copyright

Updated: 2025/10/23

Table of Contents

1

Welcome to the Progress DataDirect for JDBC for Apache Spark SQL Driver

The Progress® DataDirect® *for* JDBC™ for Apache Spark SQL™ driver supports standard SQL query language for read-write access to the following Apache Spark SQL servers:

- Apache Spark SQL 2.0 and higher
- Apache Spark SQL 1.2 and higher

The driver is designed to access Spark SQL via the Thrift JDBC server. For more information on this implementation, refer to Spark SQL and DataFrame Guide: Distributed SQL Engine.

The documentation for the driver also includes the *Progress DataDirect for JDBC Drivers Reference*. The reference provides general reference information for all DataDirect drivers for JDBC, including content on troubleshooting, supported SQL escapes, and DataDirect tools.

For the complete documentation set, visit the Progress DataDirect Connectors Documentation Hub: https://docs.progress.com/category/datadirect-apache-spark-sql

For details, see the following topics:

What's New in this Release?

Support and certification

Visit the following web pages for the latest support and certification information.

- Release Notes: https://www.progress.com/datadirect-connectors/whats-new#jdbc
- DataDirect Product Compatibility Guide: https://docs.progress.com/bundle/datadirect-product-compatibility/resource/datadirect-product-compatibility.pdf

Changes Since the 6.0.1 Release

- **Enhancements**
  - The driver has been enhanced to comply with FIPS standards for data encryption. As part of this enhancement, the driver was tested with FIPS 140-3 enabled using a Red Hat OpenJDK 21 on a Red Hat Universal Base Image 9 instance. See FIPS (Federal Information Processing Standard) on page 72 for details.
  - The driver has been enhanced to allow you to override the default value of the User-Agent header when required by a service. Using the new UserAgent property, you can specify the string value of the User-Agent header to be used in HTTP requests. See UserAgent on page 122 for details.
  - The driver has been enhanced to include timestamp in the Spy and JDBC packet logs by default. See SpyAttributes on page 114 for details.
  - Interactive SQL for JDBC (JDBCISQL) is now installed with the product. JDBCISQL is a command-line interface that supports connecting your driver to a data source, executing SQL statements and retrieving results in a terminal. This tool provides a method to quickly test your drivers in an environment that does not support GUIs. See Interactive SQL for JDBC (JDBCISQL) on page 49 for details.
  - The driver has been enhanced to support the Statement.cancel API, which allows you to cancel running queries. The Statement.cancel API is supported only on Apache Spark SQL 2.0 and higher.
  - The driver has been enhanced to support the Binary data type for Apache Spark SQL 2.0 and higher, including the following two new connection properties:
    - MaxBinarySize allows you to specify the maximum length of fields of the Binary data type that the driver describes through result set descriptions and metadata methods.
    - BinaryDescribeType allows you to specify whether Binary columns are described as VARBINARY or LONGVARBINARY.

    See Data Types on page 14, BinaryDescribeType on page 89, and MaxBinarySize on page 105 for details.

- The driver has been enhanced to support HTTP mode, which allows you to access Apache Spark SQL data stores using HTTP/HTTPS requests. HTTP mode can be configured using the new TransportMode and HTTPPath connection properties. See HTTP Mode on page 48, TransportMode on page 118, and HTTPPath on page 98 for details.
- The driver has been enhanced to support cookie based authentication for HTTP connections. Cookie based authentication can be configured using the new EnableCookieAuthentication and CookieName connection properties. See Configuring User ID/Password Authentication on page 63, EnableCookieAuthentication on page 95, and CookieName on page 93.

- **Changed Behavior**
  - The connection property `SpyAttributes` has been updated to exclude the attribute `load=classname`, which was previously used to load the driver specified by the given class name. See SpyAttributes on page 114 for details.
  - Java SE 7 has reached the end of its product life cycle and will no longer receive generally available security updates. As a result, the drivers will no longer support JVMs that are version Java SE 7 or earlier. Support for distributed versions of Java SE 7 and earlier will also end, including IBM SDK (Java Edition).

Highlights of the 6.0.1 Release

- **Certifications**
  - Certified with Apache Spark SQL 1.4.*x* and 1.5.*x*.
- **Enhancements**
  - Enhanced to support the Decimal and Varchar data types. See Data Types on page 14 and getTypeInfo on page 15 for details.
  - Added ArrayFetchSize connection property to improve performance and reduce out of memory errors. ArrayFetchSize can be used to increase throughput or, alternately, improve response time in Web-based applications. See ArrayFetchSize on page 87 and Performance Considerations on page 61 for details.
- **Changed Behavior**
  - The driver no longer registers the Statement Pool Monitor as a JMX MBean by default. To register the Statement Pool Monitor and manage statement pooling with standard JMX API calls, the new RegisterStatementPoolMonitorMBean connection property must be set to true. For details, see RegisterStatementPoolMonitorMBean on page 111.

Highlights of the 6.0.0 Release

- Supports Apache Spark 1.2 and higher.
- Supports SSL protocol for sending encrypted data. See Data encryption on page 70 for details.
- Supports Kerberos authentication. See Using Authentication on page 62 for details.
- Returns result set metadata for parameterized statements that have been prepared but not yet executed. See Parameter metadata support on page 75 for details.
- Supports connection pooling. For details, refer to "Connection Pool Manager" in the *Progress DataDirect for JDBC Drivers Reference*.
- Supports statement pooling. For details, refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference*.

- Includes a set of timeout connection properties which allow you to limit the duration of active sessions and how long the driver waits to establish a connection before timing out. See Timeout Properties on page 56 for details.
- Includes the TransactionMode connection property which allows you to configure the driver to report that it supports transactions, although Spark SQL does not support transactions. This provides a workaround for applications that do not operate with a driver that reports transactions are not supported. See TransactionMode on page 117 for details.
- The driver provides support for the following standard SQL functionality:
  - Insert
  - Create Table and Create View
  - Drop Table and Drop View
  - Batches

  See Supported SQL Functionality on page 125 for details.

Requirements

The driver is compatible with JDBC 2.0, 3.0, and 4.0.

The driver requires a Java Virtual Machine (JVM) that is Java SE 8 or higher, including Oracle JDK, OpenJDK, and IBM SDK (Java) distributions.

Data Source and Driver Classes

The driver class for the driver is:

com.ddtek.jdbc.sparksql.SparkSQLDriver

Two data source classes are provided with the driver. Which data source class you use depends on the JDBC functionality your application requires. The following table shows the recommended data source class to use with different JDBC specifications.

Table 1: Choosing a Data Source Class

If your application requires...	JVM Version	Data Source Class
JDBC 4.0 functionality and higher	Java SE 8 or higher	com.ddtek.jdbcx.sparksql.SparkSQLDataSource40
JDBC 3.x functionality and earlier specifications	Java SE 8 or higher	com.ddtek.jdbcx.sparksql.SparkSQLDataSource

See also

Connecting Using Data Sources on page 28

Connection URL

After setting the CLASSPATH, the required connection information needs to be passed in the form of a connection URL. The form of the connection URL differs depending on whether you are using a binary or HTTP connection.

Note:

- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: {*value*}. For example: `User={hello }` or `Password={{hello}}`.

For binary connections (the default):

```
jdbc:datadirect:sparksql://servername:port[;property=value[;...]]
```

For HTTP connections (TransportMode=http):

```
jdbc:datadirect:sparksql://servername:port;DatabaseName=database;
TransportMode=http;[property=value[;...]]
```

where:

servername

specifies the name or the IP address of the server to which you want to connect.

port

specifies the port number of the server listener. The default is `10000`.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

This examples show how to establish a connection to a server with user ID/password authentication.

For binary connections:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:sparksql://Server3:10000;
  DatabaseName=Test;User=admin;Password=adminpass");
```

For HTTP connections:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:sparksql://Server3:10000;DatabaseName=MyDB;
  TransportMode=http;User=admin;Password=adminpass";)
```

See also

Using Connection Properties on page 50

Version String Information

The DatabaseMetaData.getDriverVersion() method returns a Driver Version string in the format:

M.m.s.bbbbbb(F*YYYYYY*.U*ZZZZZZ*)

where:

M is the major version number.

m is the minor version number.

s is the service pack number.

bbbbbb is the driver build number.

YYYYYY is the framework build number.

ZZZZZZ is the utl build number.

For example:

```
6.0.0.000002(F000001.U000002)
      |____| |_____| |_____|
      Driver  Frame    Utl
```

Connection Properties

The driver includes over 35 connection properties. You can use these connection properties to customize the driver for your environment. Connection properties can be used to accomplish different tasks, such as implementing driver functionality and optimizing performance. You can specify connection properties in a connection URL or within a JDBC data source object.

See also

Using Connection Properties on page 50
Connection Property Descriptions on page 81

Data Types

The following table shows how the driver maps Apache Spark SQL data types to standard JDBC data types.

Table 2: Data Types

Apache Spark SQL	JDBC
Bigint	BIGINT

Apache Spark SQL	JDBC
Binary	VARBINARY or LONGVARBINARY[1, 2]
Boolean	BOOLEAN
Date	DATE
Decimal	DECIMAL
Double	DOUBLE
Float	REAL
Int	INTEGER
Smallint	SMALLINT
String[3, 4]	VARCHAR *or* LONGVARCHAR
Timestamp	TIMESTAMP
Tinyint	TINYINT
Varchar[5]	VARCHAR

See also

getTypeInfo on page 15

getTypeInfo

The following table provides getTypeInfo results for all sources supported by the driver.

[1] If the BinaryDescribeType property is set to `varbinary` (the default), this data type maps to VARBINARY. If set to `longvarbinary`, this data type maps to LONGVARBINARY.

[2] Supported only for Apache Spark SQL 2.0 and higher.

[3] Maximum of 2 GB

[4] If the StringDescribeType connection property is set to `varchar` (the default), the String data type maps to VARCHAR. If StringDescribeType is set to `longvarchar`, String maps to LONGVARCHAR. StringDescribeType affects all columns reported as String, even columns that were originally cast as Varchar. This is important to note because the Spark Thrift server, when returning result metadata for Varchar columns, reports column type as (12) STRING and precision as 2147483647.

[5] When returning result set metadata for Varchar columns, the Spark Thrift server reports the column type as (12) STRING and the precision as 2147483647. For the latest information about this issue, refer to the Apache JIRA SPARK-5918 issue Web page.

Table 3: getTypeInfo()

TYPE_NAME = bigint	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = L	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bigint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
TYPE_NAME = boolean	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 16 (BOOLEAN)	PRECISION = 1
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = boolean	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = date	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = true	SEARCHABLE = 3
LITERAL_PREFIX = {d'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = decimal AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = decimal MAXIMUM_SCALE = 38	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = double AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 8 (DOUBLE) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = double MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 15 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = float AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 7 (REAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = float MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 7 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

TYPE_NAME = int AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = int MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = smallint AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 5 (SMALLINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = S LOCAL_TYPE_NAME = smallint MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 5 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = string[6, 7] AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) *or* -1 (LONGVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = string MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

[6] Maximum of 2 GB

[7] If the StringDescribeType connection property is set to `varchar` (the default), the String data type maps to VARCHAR. If StringDescribeType is set to `longvarchar`, String maps to LONGVARCHAR. StringDescribeType affects all columns reported as String, even columns that were originally cast as Varchar. This is important to note because the Spark Thrift server, when returning result metadata for Varchar columns, reports column type as (12) STRING and precision as 2147483647.

TYPE_NAME = timestamp AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = {ts' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = timestamp MAXIMUM_SCALE = 9	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 29 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = tinyint AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -6 (TINYINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = Y LOCAL_TYPE_NAME = tinyint MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 3 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = varchar[8] AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = varchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 65355 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

[8] When returning result set metadata for Varchar columns, the Spark Thrift server reports the column type as (12) STRING and the precision as 2147483647. For the latest information about this issue, refer to the Apache JIRA SPARK-5918 issue Web page.

DataDirect tools

Progress DataDirect for JDBC drivers install the set of tools described in this section. For detailed instructions on using these tools, refer to the corresponding topics in the *Progress DataDirect for JDBC Drivers Reference*.

- DataDirect Test allows you to test your JDBC driver and learn the JDBC API.
- DataDirect Connection Pool Manager allows you to pool connections when accessing databases. When your applications use connection pooling, connections are reused rather than created each time a connection is requested. Because establishing a connection is among the most costly operations an application may perform, using Connection Pool Manager to implement connection pooling can significantly improve performance.
- Statement Pool Monitor loads statements into and remove statements from the statement pool as well as generate information to help you troubleshoot statement pooling performance.
- DataDirect Spy logs detailed information about calls your driver makes that can be used for troubleshooting.

Troubleshooting

The *Progress DataDirect for JDBC Drivers Reference* provides information on troubleshooting problems should they occur. Refer to the "Troubleshooting" section in the *Reference* for details.

Additional information

In addition to the content provided in this guide, the documentation set also contains detailed conceptual and reference information that applies to all the drivers. For more information in these topics, refer the *Progress DataDirect for JDBC Drivers Reference* or use the links below to view some common topics:

- "JDBC support" describes support for JDBC interfaces and methods for the Progress DataDirect for JDBC drivers.
- "JDBC extensions" describes the JDBC extensions provided by the com.ddtek.jdbc.extensions package.
- "SQL escape sequences for JDBC" provides an overview of SQL escape sequences for JDBC. In addition, it documents the scalar functions that you use in SQL statements.
- "Security best practices for JDBC applications" describes the security best practices you should employ when developing and deploying your application with the driver.

Contacting Technical Support

Progress DataDirect offers a variety of options to meet your support needs. Please visit our Web site for more details and for contact information:

https://www.progress.com/support

The Progress DataDirect Web site provides the latest support information through our global service network. The SupportLink program provides access to support contact details, tools, patches, and valuable information, including a list of FAQs for each product. In addition, you can search our Knowledgebase for technical bulletins and other information.

When you contact us for assistance, please provide the following information:

- Your number or the serial number that corresponds to the product for which you are seeking support, or a case number if you have been provided one for your issue. If you do not have a SupportLink contract, the SupportLink representative assisting you will connect you with our Sales team.
- Your name, phone number, email address, and organization. For a first-time call, you may be asked for full information, including location.
- The Progress DataDirect product and the version that you are using.
- The type and version of the operating system where you have installed your product.
- Any database, database version, third-party software, or other environment information required to understand the problem.
- A brief description of the problem, including, but not limited to, any error messages you have received, what steps you followed prior to the initial occurrence of the problem, any trace logs capturing the issue, and so on. Depending on the complexity of the problem, you may be asked to submit an example or reproducible application so that the issue can be re-created.
- A description of what you have attempted to resolve the issue. If you have researched your issue on Web search engines, our Knowledgebase, or have tested additional configurations, applications, or other vendor products, you will want to carefully note everything you have already attempted.
- A simple assessment of how the severity of the issue is impacting your organization.

2

Getting started

After the driver has been installed and defined on your class path, you can connect from your application to your database in either of the following ways.

- Using the JDBC `DriverManager`, by specifying the connection URL in the `DriverManager.getConnection()` method.
- Creating a JDBC `DataSource` that can be accessed through the Java Naming Directory Interface (JNDI).

For details, see the following topics:

- Data Source and Driver Classes
- Setting the Classpath
- Connecting with the JDBC Driver Manager
- Connecting Using Data Sources

Data Source and Driver Classes

The driver class for the driver is:

com.ddtek.jdbc.sparksql.SparkSQLDriver

Two data source classes are provided with the driver. Which data source class you use depends on the JDBC functionality your application requires. The following table shows the recommended data source class to use with different JDBC specifications.

Table 4: Choosing a Data Source Class

If your application requires...	JVM Version	Data Source Class
JDBC 4.0 functionality and higher	Java SE 8 or higher	com.ddtek.jdbcx.sparksql.SparkSQLDataSource40
JDBC 3.x functionality and earlier specifications	Java SE 8 or higher	com.ddtek.jdbcx.sparksql.SparkSQLDataSource

See also

Connecting Using Data Sources on page 28

Setting the Classpath

The driver must be defined in your CLASSPATH variable. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. If the driver is not defined on your CLASSPATH, you will receive a `class not found` exception when trying to load the driver. Set your system CLASSPATH to include the `sparksql.jar` file as shown, where *`install_dir`* is the path to your product installation directory:

```
install_dir/lib/60/sparksql.jar
```

Windows Example

```
CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\sparksql.jar
```

UNIX Example

```
CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/sparksql.jar
```

Connecting with the JDBC Driver Manager

One way to connect to a Spark SQL database is through the JDBC DriverManager using the DriverManager.getConnection() method. As the following example shows, this method specifies a string containing a connection URL.

```
Connection conn = DriverManager.getConnection
   ("jdbc:datadirect:sparksql://Server3:10000;
   DatabaseName=Test;User=admin;Password=adminpass");
```

See also

Connecting Using Data Sources on page 28

Connection URL

After setting the CLASSPATH, the required connection information needs to be passed in the form of a connection URL. The form of the connection URL differs depending on whether you are using a binary or HTTP connection.

Note:

- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: {*value*}. For example: `User={hello }` or `Password={{hello}}`.

For binary connections (the default):

```
jdbc:datadirect:sparksql://servername:port[;property=value[;...]]
```

For HTTP connections (TransportMode=http):

```
jdbc:datadirect:sparksql://servername:port;DatabaseName=database;
TransportMode=http;[property=value[;...]]
```

where:

servername

specifies the name or the IP address of the server to which you want to connect.

port

specifies the port number of the server listener. The default is `10000`.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

This examples show how to establish a connection to a server with user ID/password authentication.

For binary connections:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:sparksql://Server3:10000;
  DatabaseName=Test;User=admin;Password=adminpass");
```

For HTTP connections:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:sparksql://Server3:10000;DatabaseName=MyDB;
  TransportMode=http;User=admin;Password=adminpass";)
```

See also

Using Connection Properties on page 50

Testing the Connection

You can use DataDirect Test™ to verify your connection.

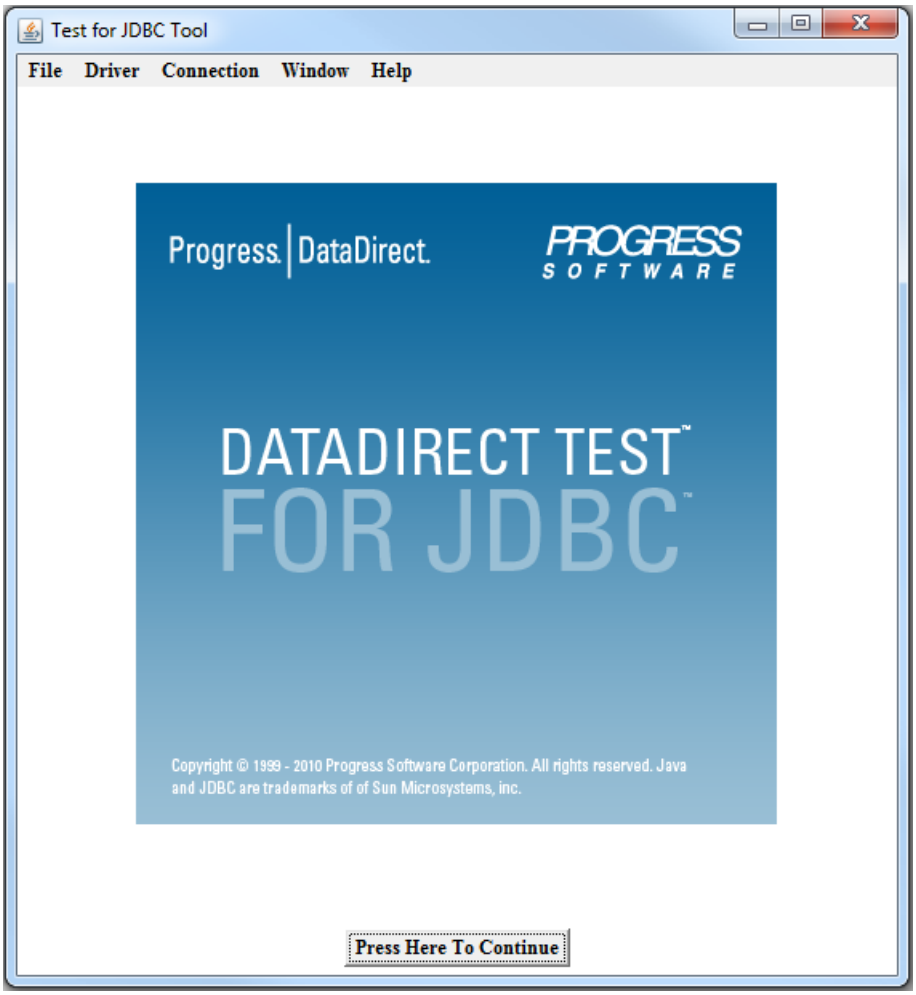
To test the Driver Manager connection, follow these steps:

1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

   Note: For UNIX/Linux, if you do not have access to `/opt`, your home directory will be used in its place.

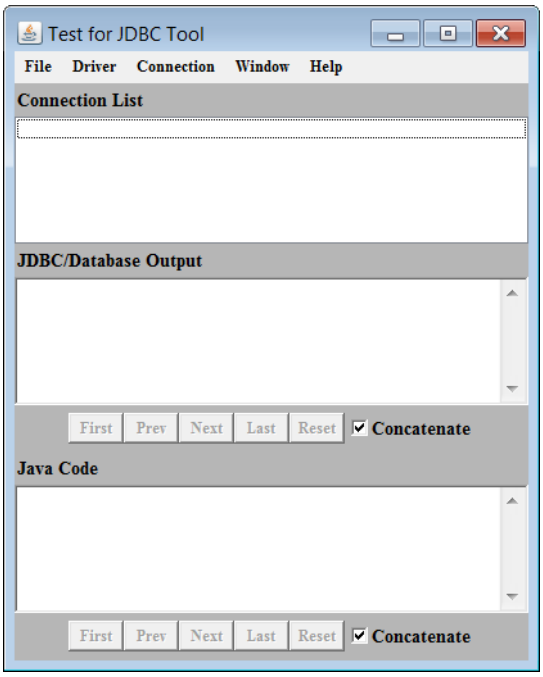
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

   The **Test for JDBC Tool** window appears:



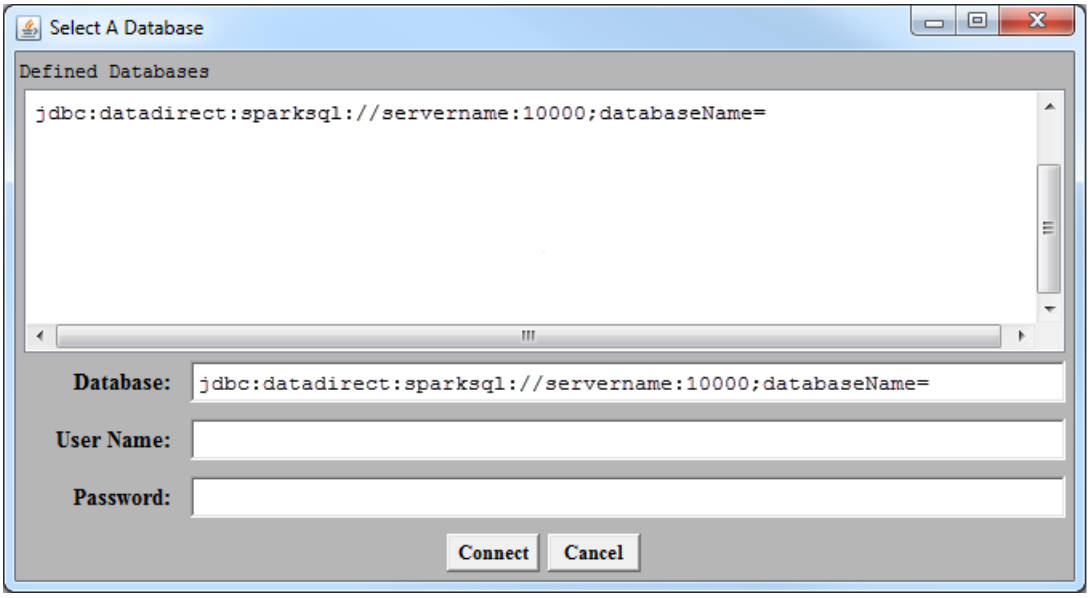
3. Click **Press Here to Continue**.

   The main dialog appears:



4. From the menu bar, select **Connection > Connect to DB**.

   The **Select A Database** dialog appears:



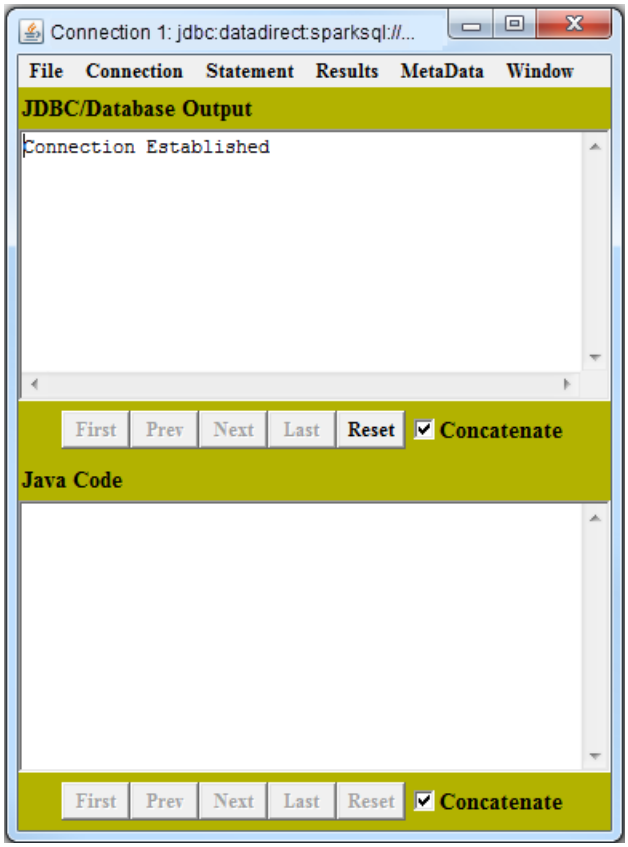
5. Select the appropriate database template from the Defined Databases field.
6. In the Database field, specify the correct ServerName and PortNumber for your Apache Spark SQL data source.

   For example:

   ```
   jdbc:datadirect:sparksql://Server3:10000;databaseName=Test
   ```

7. If required, enter your user name and password in the fields provided.
8. Click **Connect**.

If the connection is successful, the **JDBC/Database Output** window reports that a connection has been established. (If a connection is not established, the window reports an error.)



Refer to "DataDirect Test" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using DataDirect Test.

Connecting Using Data Sources

A *JDBC data source* is a Java object, specifically a DataSource object, that defines connection information required for a JDBC driver to connect to the database. Each JDBC driver vendor provides their own data source implementation for this purpose. A Progress DataDirect data source is Progress DataDirect's implementation of a DataSource object that provides the connection information needed for the driver to connect to a database.

Because data sources work with the Java Naming Directory Interface (JNDI) naming service, data sources can be created and managed separately from the applications that use them. Because the connection information is defined outside of the application, the effort to reconfigure your infrastructure when a change is made is minimized. For example, if the database is moved to another database server, the administrator need only change the relevant properties of the data source (DataSource object). The applications using the database do not need to change because they only refer to the name of the data source.

How Data Sources Are Implemented

Data sources are implemented through a data source class. A data source class implements the following interfaces:

- `javax.sql.DataSource`
- `javax.sql.ConnectionPoolDataSource` (allows applications to use connection pooling)

Refer to "Connection Pool Manager" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

See also

Data Source and Driver Classes on page 12

Creating Data Sources

The following examples show how to create and use Progress DataDirect data sources:

- `JNDI_LDAP_Example.java` can be used to create a JDBC data source and save it in your LDAP directory using the JNDI Provider for LDAP.
- `JNDI_FILESYSTEM_Example.java` can be used to create a JDBC data source and save it in your local file system using the File System JNDI Provider.

You can use these examples as templates to create your own data sources. These examples are in the *`install_dir`*`/examples/JNDI` directory, where *`install_dir`* is your product installation directory.

Note: To connect using a data source, the driver needs to access a JNDI data store to persist the data source information. To download the JNDI File System Service Provider, go to the Oracle Technology Network Java SE Support downloads page and make sure that the fscontext.jar and providerutil.jar files from the download are on your classpath.

Note: You must include the javax.sql.* and javax.naming.* classes to create and use Progress DataDirect data sources. The driver provides the necessary JAR files, which contain the required classes and interfaces. If you plan to connect using a JDBC data source, the fscontext.jar and providerutil.jar files, which are shipped with the JNDI File System Service Provider, must be on your classpath. To download the JNDI File System Service Provider, go to the Oracle Technology Network Java SE Support downloads page and select a JNDI version.

Example Data Source

To configure a data source using the example files, you will need to create a data source definition. The content required to create a data source definition is divided into three sections.

First, you will need to import the data source class. For example:

```
import com.ddtek.jdbcx.sparksql.SparkSQLDataSource;
```

Next, you will need to set the values and define the data source. For example, the following definition contains the minimum properties required for a binary connection:

Note:

- Setting the password using a data source is generally not recommended. The data source persists all properties, including the Password property, in clear text.
- In a JDBC data source, string values must be enclosed in double quotation marks, for example, `setUser("abc@defcorp.com")`.

```
SparkSQLDataSource mds = new SparkSQLDataSource();
mds.setDescription("My Spark SQL Server");
mds.setServerName("MyServer");
mds.setPortNumber(10000);
mds.setDatabaseName("myDB");
```

The following example contains the minimum properties for a connection in HTTP mode:

```
SparkSQLDataSource mds = new SparkSQLDataSource();
mds.setDescription("My Spark SQL Server");
mds.setServerName("MyServer");
mds.setPortNumber(10000);
mds.setDatabaseName("myDB");
mds.setTransportMode("http");
```

Finally, you will need to configure the example application to print out the data source attributes. Note that this code is specific to the driver and should only be used in the example application. For example, you would add the following section for a binary connection using only the minimum properties:

```
if (ds instanceof SparkSQLDataSource)
{
SparkSQLDataSource jmds = (SparkSQLDataSource) ds;
System.out.println("description=" + jmds.getDescription());
System.out.println("serverName=" + jmds.getServerName());
System.out.println("portNumber=" + jmds.getPortNumber());
System.out.println("databaseName=" + jmds.getDatabaseName());
System.out.println();
}
```

Calling a Data Source in an Application

Applications can call a Progress DataDirect data source using a logical name to retrieve the javax.sql.DataSource object. This object loads the specified driver and can be used to establish a connection to the database.

Once the data source has been registered with JNDI, it can be used by your JDBC application as shown in the following code example:

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("EmployeeDB");
Connection con = ds.getConnection("domino", "spark");
```

In this example, the JNDI environment is first initialized. Next, the initial naming context is used to find the logical name of the data source (EmployeeDB). The Context.lookup() method returns a reference to a Java object, which is narrowed to a javax.sql.DataSource object. Finally, the DataSource.getConnection() method is called to establish a connection.

Testing a data source connection

You can use DataDirect Test™ to establish and test a data source connection. The screen shots in this section were taken on a Windows system.

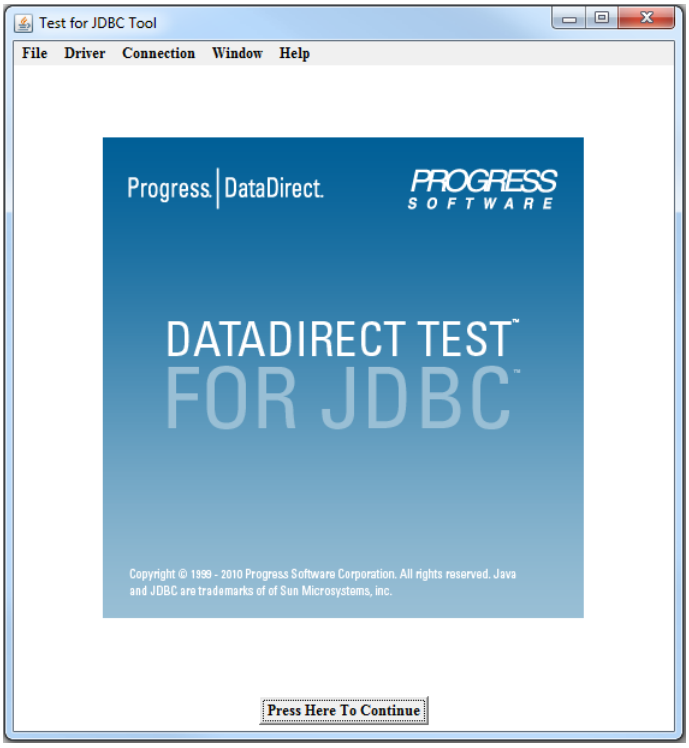
Take the following steps to establish a connection.

1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

   Note: For UNIX/Linux, if you do not have access to `/opt`, your home directory will be used in its place.

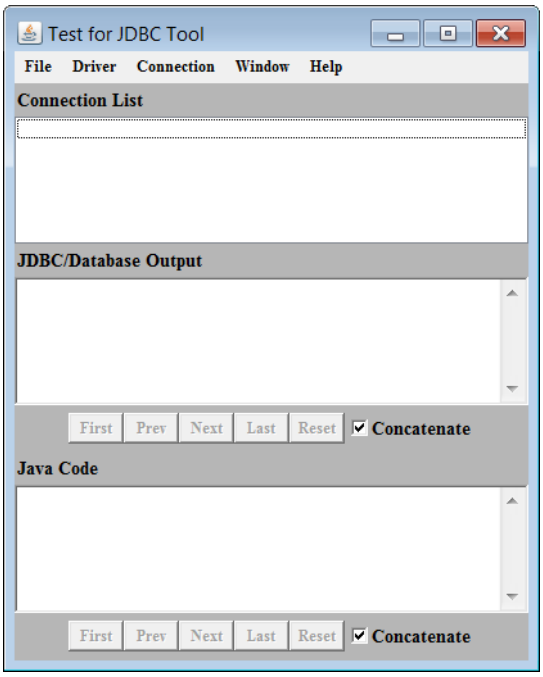
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

   The **Test for JDBC Tool** window appears:



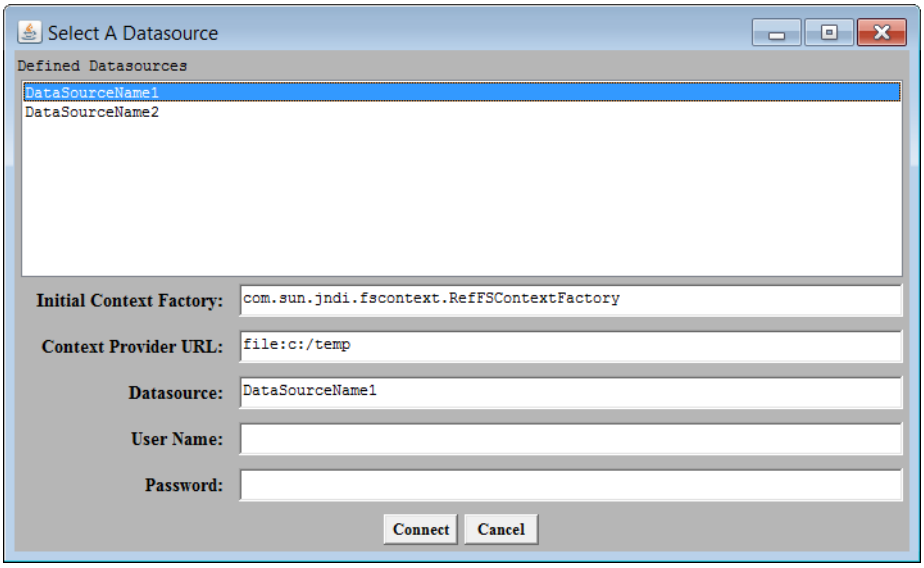
3. Click **Press Here to Continue**.

   The main dialog appears:



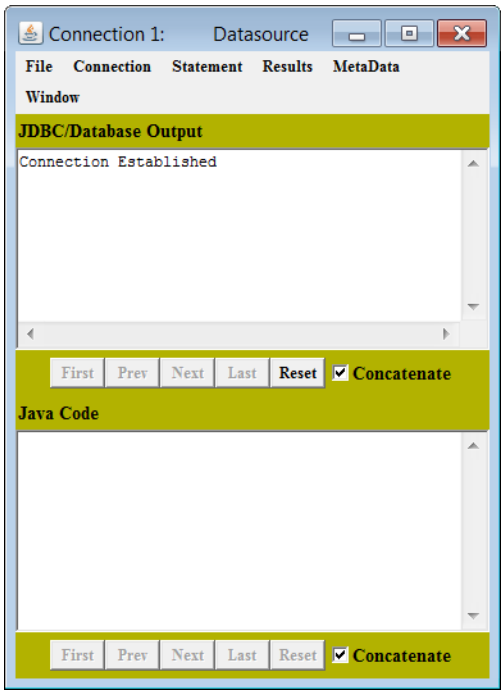
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   The **Select A Database** dialog appears:



5. Select a datasource template from the **Defined Datasources** field.
6. Provide the following information:
   a) In the **Initial Context Factory**, specify the location of the initial context provider for your application.
   b) In the **Context Provider URL**, specify the location of the context provider for your application.
   c) In the **Datasource** field, specify the name of your datasource.
7. If you are using user ID/password authentication, enter your user ID and password in the corresponding fields.
8. Click **Connect**.

If the connection information is entered correctly, the **JDBC/Database Output** window reports that a connection has been established. If a connection is not established, the window reports an error.



3

Using the driver

This section provides information on how to connect to your data store using either the JDBC Driver Manager or DataDirect JDBC data sources, as well as information on how to implement and use functionality supported by the driver.

For details, see the following topics:

- Required permissions for Java SE with the standard Security Manager enabled
- Connecting from an Application
- Using Connection Properties
- Performance Considerations
- Using Authentication
- Data encryption
- Proxy server support
- Using Client Information
- IP Addresses
- Parameter metadata support
- ResultSet metadata support
- Isolation Levels
- Unicode support
- Error Handling

- Large Object Support
- Rowset Support
- Timeouts
- Views
- SQL Escape Sequences
- Using Scrollable Cursors
- Spark SQL Compatibility with Apache Hive
- Stored Procedures

Required permissions for Java SE with the standard Security Manager enabled

Using the driver on a Java platform with the standard Security Manager enabled requires certain permissions to be set in the Java SE security policy file `java.policy`. The default location of this file is *`java_install_dir`*`/jre/lib/security`.

Note: Security manager may be enabled by default in certain scenarios, such as running on an application server or in a Web browser applet.

To run an application on a Java platform with the standard Security Manager, use the following command:

```
"java -Djava.security.manager application_class_name"
```

where *`application_class_name`* is the class name of the application.

Refer to your Java documentation for more information about setting permissions in the security policy file.

Permissions for establishing connections

To establish a connection to the database server, the driver must be granted the permissions as shown in the following example:

```
grant codeBase "file:/install_dir/lib/60/-" {
   permission java.net.SocketPermission "*", "connect";
};
```

where:

`install_dir`

is the product installation directory.

Granting access to Java properties

To allow the driver to read the value of various Java properties to perform certain operations, permissions must be granted as shown in the following example:

```
grant codeBase "file:/install_dir/lib/60/-" {
   permission java.util.PropertyPermission "*", "read, write";
};
```

where:

install_dir

is the product installation directory.

Granting access to temporary files

Access to the temporary directory specified by the JVM configuration must be granted in the Java SE security policy file to use insensitive scrollable cursors or to perform client-side sorting of DatabaseMetaData result sets. The following example shows permissions that have been granted for the `C:\TEMP` directory:

```
grant codeBase "file:/install_dir/lib/60/-" {
// Permission to create and delete temporary files.
// Adjust the temporary directory for your environment.
   permission java.io.FilePermission "C:\\TEMP\\-", "read,write,delete";
};
```

where:

install_dir

is the product installation directory.

Permissions for Kerberos Authentication

To use Kerberos authentication, the application and driver code bases must be granted security permissions in the security policy file of the Java Platform as shown in the following code example.

```
grant codeBase "file:/install_dir/lib/60/-" {
   permission javax.security.auth.AuthPermission
      "createLoginContext.DDTEK-JDBC";
   permission javax.security.auth.AuthPermission "doAs";
   permission javax.security.auth.kerberos.ServicePermission
      "krbtgt/your_realm@your_realm", "initiate";
   permission javax.security.auth.kerberos.ServicePermission
      "principal_name/db_hostname@your_realm", "initiate";
};
```

where:

install_dir

is the product installation directory.

your_realm

is the Kerberos realm (or Windows Domain) to which the database host machine belongs.

principal_name

is the service principal name registered with the Key Distribution Center (KDC) that identifies the database service.

db_hostname

is the host name of the machine running the database.

Connecting from an Application

Once the driver is installed and configured, you can connect to your database in either of the following ways:

- Using the JDBC Driver Manager, by specifying the connection URL in the `DriverManager.getConnection()` method.
- Creating a JDBC data source that can be accessed through the Java Naming Directory Interface (JNDI).

Data Source and Driver Classes

The driver class for the driver is:

com.ddtek.jdbc.sparksql.SparkSQLDriver

Two data source classes are provided with the driver. Which data source class you use depends on the JDBC functionality your application requires. The following table shows the recommended data source class to use with different JDBC specifications.

Table 5: Choosing a Data Source Class

If your application requires...	JVM Version	Data Source Class
JDBC 4.0 functionality and higher	Java SE 8 or higher	com.ddtek.jdbcx.sparksql.SparkSQLDataSource40
JDBC 3.x functionality and earlier specifications	Java SE 8 or higher	com.ddtek.jdbcx.sparksql.SparkSQLDataSource

See also

Connecting Using Data Sources on page 28

Setting the Classpath

The driver must be defined in your CLASSPATH variable. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. If the driver is not defined on your CLASSPATH, you will receive a `class not found` exception when trying to load the driver. Set your system CLASSPATH to include the `sparksql.jar` file as shown, where *`install_dir`* is the path to your product installation directory:

`install_dir``/lib/60/sparksql.jar`

Windows Example

```
CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\sparksql.jar
```

UNIX Example

```
CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/sparksql.jar
```

Connecting with the JDBC Driver Manager

One way to connect to a Spark SQL database is through the JDBC DriverManager using the DriverManager.getConnection() method. As the following example shows, this method specifies a string containing a connection URL.

```
Connection conn = DriverManager.getConnection
   ("jdbc:datadirect:sparksql://Server3:10000;
   DatabaseName=Test;User=admin;Password=adminpass");
```

See also

Connecting Using Data Sources on page 28

Connection URL

After setting the CLASSPATH, the required connection information needs to be passed in the form of a connection URL. The form of the connection URL differs depending on whether you are using a binary or HTTP connection.

Note:

- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: {*`value`*}. For example: `User={hello }` or `Password={{hello}}`.

For binary connections (the default):

`jdbc:datadirect:sparksql://`*`servername`*`:`*`port`*`[;`*`property`*`=`*`value`*`[;...]]`

For HTTP connections (TransportMode=http):

```
jdbc:datadirect:sparksql://servername:port;DatabaseName=database;
TransportMode=http;[property=value[;...]]
```

where:

servername

specifies the name or the IP address of the server to which you want to connect.

port

specifies the port number of the server listener. The default is `10000`.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

This examples show how to establish a connection to a server with user ID/password authentication.

For binary connections:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:sparksql://Server3:10000;
  DatabaseName=Test;User=admin;Password=adminpass");
```

For HTTP connections:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:sparksql://Server3:10000;DatabaseName=MyDB;
  TransportMode=http;User=admin;Password=adminpass";)
```

See also

Using Connection Properties on page 50

Testing the Connection

You can use DataDirect Test™ to verify your connection.

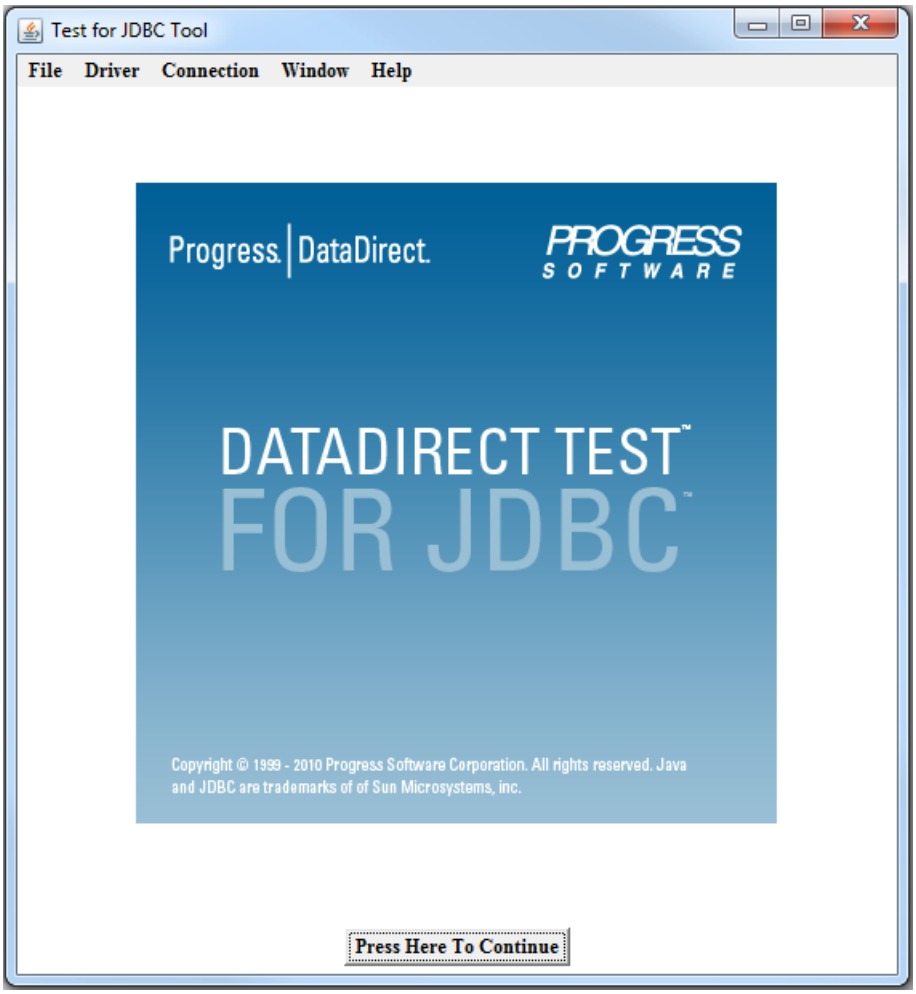
To test the Driver Manager connection, follow these steps:

1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

   Note: For UNIX/Linux, if you do not have access to `/opt`, your home directory will be used in its place.

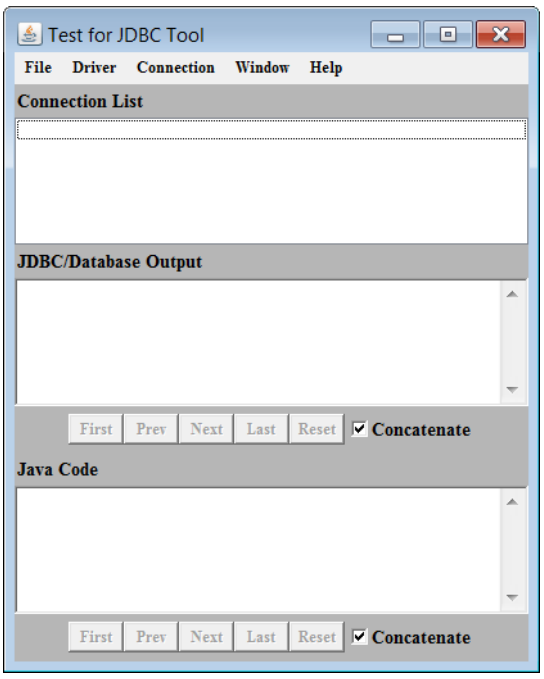
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

   The **Test for JDBC Tool** window appears:



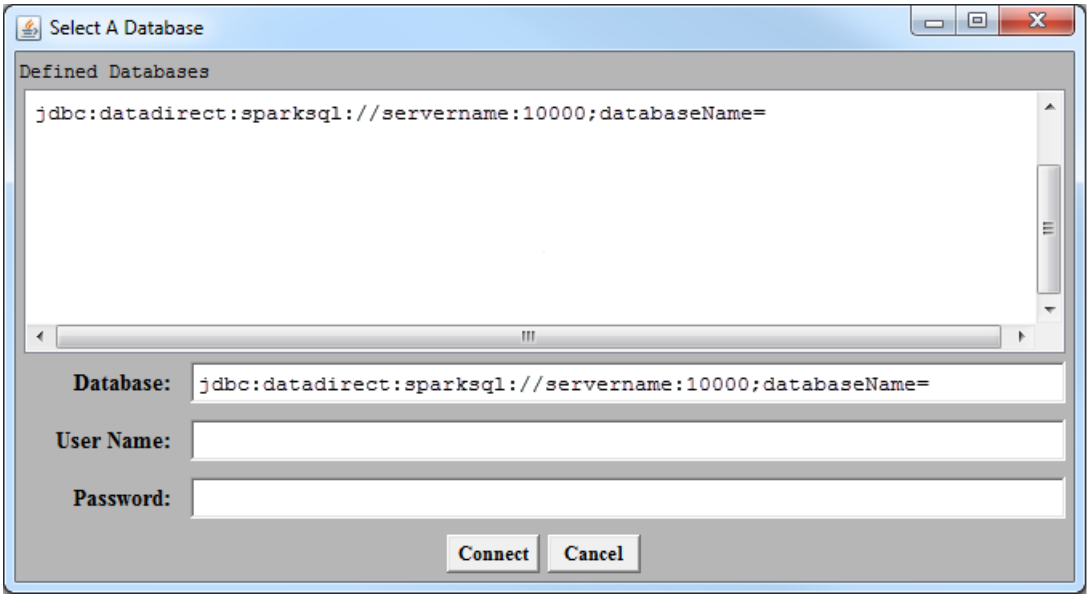
3. Click **Press Here to Continue**.

   The main dialog appears:



4. From the menu bar, select **Connection > Connect to DB**.

   The **Select A Database** dialog appears:



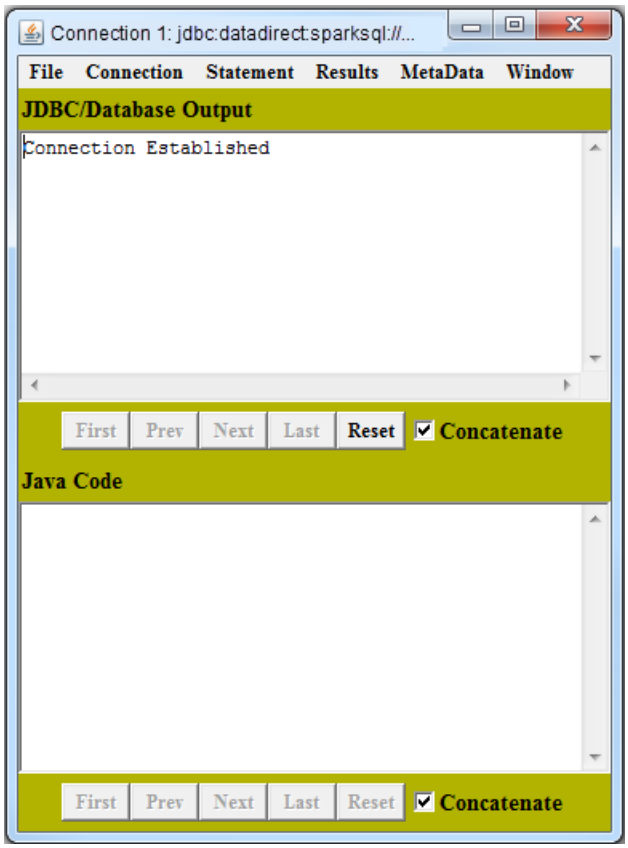
5. Select the appropriate database template from the Defined Databases field.
6. In the Database field, specify the correct ServerName and PortNumber for your Apache Spark SQL data source.

   For example:

   ```
   jdbc:datadirect:sparksql://Server3:10000;databaseName=Test
   ```

7. If required, enter your user name and password in the fields provided.
8. Click **Connect**.

If the connection is successful, the **JDBC/Database Output** window reports that a connection has been established. (If a connection is not established, the window reports an error.)



Refer to "DataDirect Test" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using DataDirect Test.

Connecting Using Data Sources

A *JDBC data source* is a Java object, specifically a DataSource object, that defines connection information required for a JDBC driver to connect to the database. Each JDBC driver vendor provides their own data source implementation for this purpose. A Progress DataDirect data source is Progress DataDirect's implementation of a DataSource object that provides the connection information needed for the driver to connect to a database.

Because data sources work with the Java Naming Directory Interface (JNDI) naming service, data sources can be created and managed separately from the applications that use them. Because the connection information is defined outside of the application, the effort to reconfigure your infrastructure when a change is made is minimized. For example, if the database is moved to another database server, the administrator need only change the relevant properties of the data source (DataSource object). The applications using the database do not need to change because they only refer to the name of the data source.

How Data Sources Are Implemented

Data sources are implemented through a data source class. A data source class implements the following interfaces:

- `javax.sql.DataSource`
- `javax.sql.ConnectionPoolDataSource` (allows applications to use connection pooling)

Refer to "Connection Pool Manager" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

See also

Data Source and Driver Classes on page 12

Creating Data Sources

The following examples show how to create and use Progress DataDirect data sources:

- `JNDI_LDAP_Example.java` can be used to create a JDBC data source and save it in your LDAP directory using the JNDI Provider for LDAP.
- `JNDI_FILESYSTEM_Example.java` can be used to create a JDBC data source and save it in your local file system using the File System JNDI Provider.

You can use these examples as templates to create your own data sources. These examples are in the *`install_dir`*`/examples/JNDI` directory, where *`install_dir`* is your product installation directory.

Note: To connect using a data source, the driver needs to access a JNDI data store to persist the data source information. To download the JNDI File System Service Provider, go to the Oracle Technology Network Java SE Support downloads page and make sure that the fscontext.jar and providerutil.jar files from the download are on your classpath.

Note: You must include the javax.sql.* and javax.naming.* classes to create and use Progress DataDirect data sources. The driver provides the necessary JAR files, which contain the required classes and interfaces. If you plan to connect using a JDBC data source, the fscontext.jar and providerutil.jar files, which are shipped with the JNDI File System Service Provider, must be on your classpath. To download the JNDI File System Service Provider, go to the Oracle Technology Network Java SE Support downloads page and select a JNDI version.

Example Data Source

To configure a data source using the example files, you will need to create a data source definition. The content required to create a data source definition is divided into three sections.

First, you will need to import the data source class. For example:

```
import com.ddtek.jdbcx.sparksql.SparkSQLDataSource;
```

Next, you will need to set the values and define the data source. For example, the following definition contains the minimum properties required for a binary connection:

Note:

- Setting the password using a data source is generally not recommended. The data source persists all properties, including the Password property, in clear text.
- In a JDBC data source, string values must be enclosed in double quotation marks, for example, `setUser("abc@defcorp.com")`.

```
SparkSQLDataSource mds = new SparkSQLDataSource();
mds.setDescription("My Spark SQL Server");
mds.setServerName("MyServer");
mds.setPortNumber(10000);
mds.setDatabaseName("myDB");
```

The following example contains the minimum properties for a connection in HTTP mode:

```
SparkSQLDataSource mds = new SparkSQLDataSource();
mds.setDescription("My Spark SQL Server");
mds.setServerName("MyServer");
mds.setPortNumber(10000);
mds.setDatabaseName("myDB");
mds.setTransportMode("http");
```

Finally, you will need to configure the example application to print out the data source attributes. Note that this code is specific to the driver and should only be used in the example application. For example, you would add the following section for a binary connection using only the minimum properties:

```
if (ds instanceof SparkSQLDataSource)
{
SparkSQLDataSource jmds = (SparkSQLDataSource) ds;
System.out.println("description=" + jmds.getDescription());
System.out.println("serverName=" + jmds.getServerName());
System.out.println("portNumber=" + jmds.getPortNumber());
System.out.println("databaseName=" + jmds.getDatabaseName());
System.out.println();
}
```

Calling a Data Source in an Application

Applications can call a Progress DataDirect data source using a logical name to retrieve the javax.sql.DataSource object. This object loads the specified driver and can be used to establish a connection to the database.

Once the data source has been registered with JNDI, it can be used by your JDBC application as shown in the following code example:

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("EmployeeDB");
Connection con = ds.getConnection("domino", "spark");
```

In this example, the JNDI environment is first initialized. Next, the initial naming context is used to find the logical name of the data source (EmployeeDB). The Context.lookup() method returns a reference to a Java object, which is narrowed to a javax.sql.DataSource object. Finally, the DataSource.getConnection() method is called to establish a connection.

Testing a data source connection

You can use DataDirect Test™ to establish and test a data source connection. The screen shots in this section were taken on a Windows system.

Take the following steps to establish a connection.

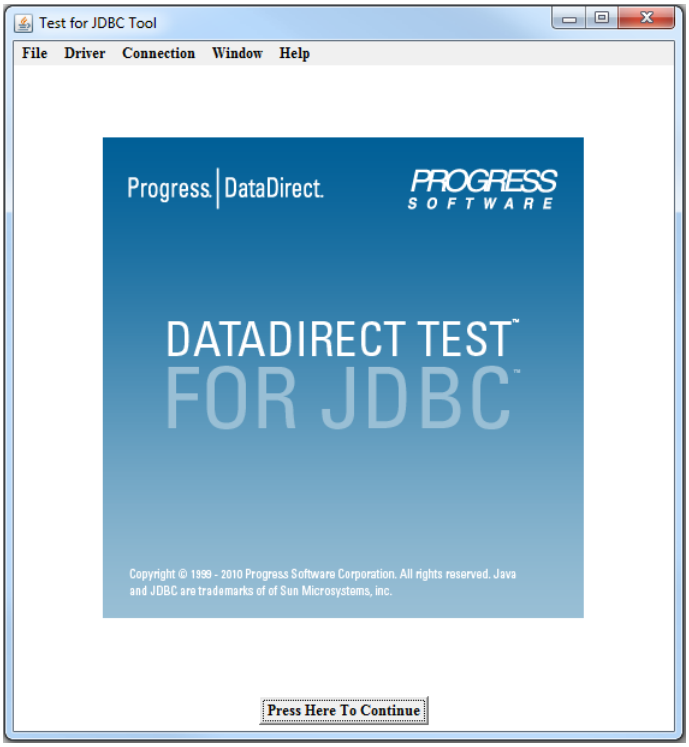
1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

   Note: For UNIX/Linux, if you do not have access to `/opt`, your home directory will be used in its place.

2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)

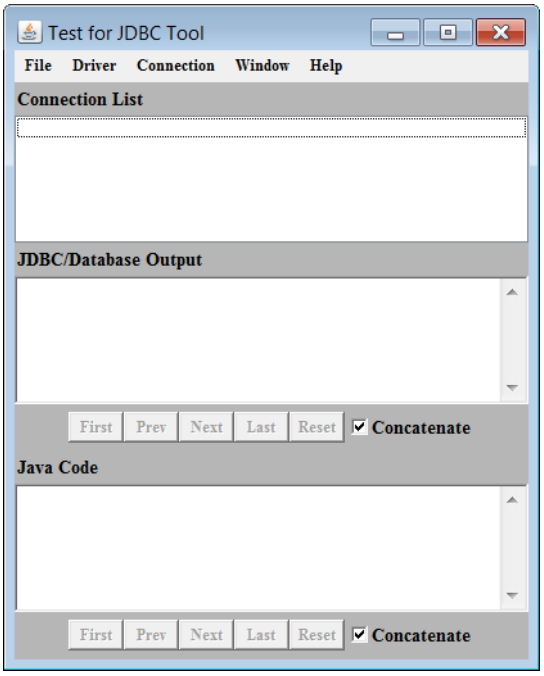
- `testforjdbc.sh` (on UNIX and Linux systems)

The **Test for JDBC Tool** window appears:



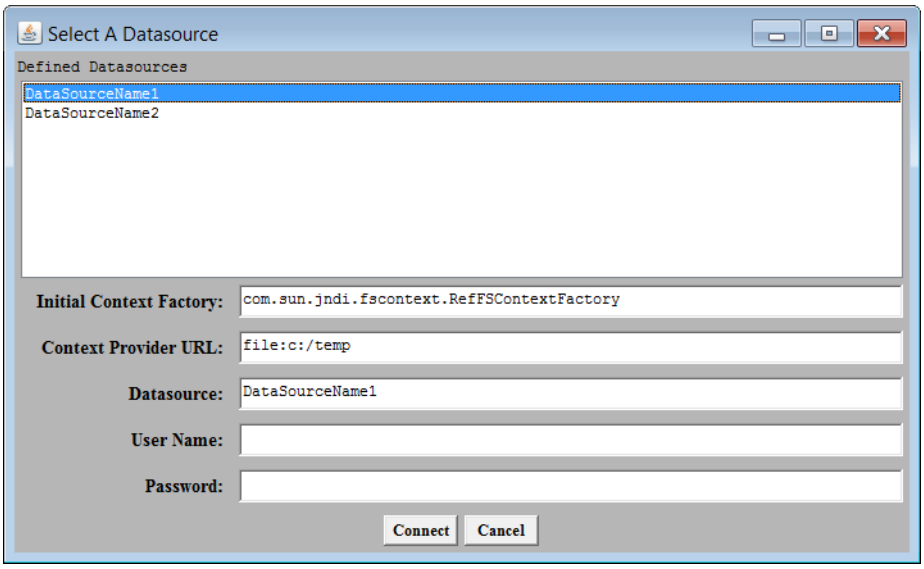
3. Click **Press Here to Continue**.

   The main dialog appears:



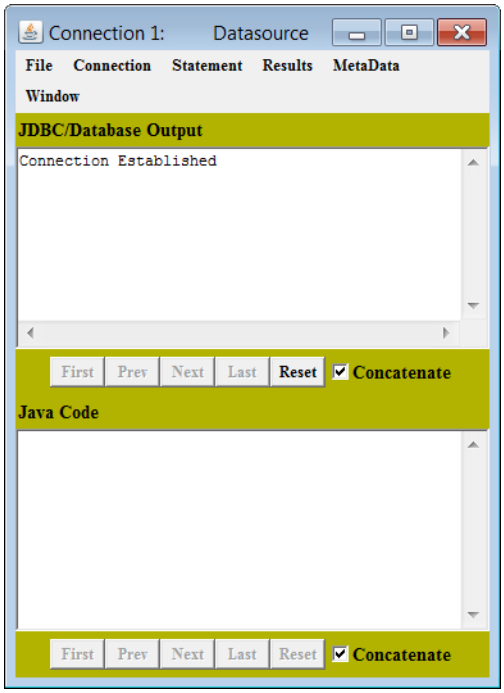
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   The **Select A Database** dialog appears:



5. Select a datasource template from the **Defined Datasources** field.
6. Provide the following information:
   a) In the **Initial Context Factory**, specify the location of the initial context provider for your application.
   b) In the **Context Provider URL**, specify the location of the context provider for your application.
   c) In the **Datasource** field, specify the name of your datasource.
7. If you are using user ID/password authentication, enter your user ID and password in the corresponding fields.
8. Click **Connect**.

If the connection information is entered correctly, the **JDBC/Database Output** window reports that a connection has been established. If a connection is not established, the window reports an error.



HTTP Mode

In addition to the default thrift protocol (binary mode), the driver also supports HTTP mode, which allows you to access Apache Spark SQL data stores using HTTP/HTTPS requests. When HTTP mode is enabled, thrift RPC messages are sent to an endpoint using HTTP transport. HTTP mode is typically employed when there is a need to access data through a proxy server, such as when connecting to a load balancer or a gateway server. Unless otherwise noted, the same features and functionality are supported for both the thrift and HTTP protocols.

To connect to a server using HTTP Mode:

1. Configure the minimum required options required for a connection:
   - Set the DatabaseName property to provide the name of the Apache Spark SQL database to which you want to connect.
   - Set the ServerName property to provide the name or the IP address of the server to which you want to connect.
   - Set the PortNumber property to provide the TCP port of the primary database server that is listening for connections to the Apache Spark SQL database. The default is `10000`.
2. Set the TransportMode property to `http`.
3. Optionally, if not using the default HTTP endpoint, set the HTTPPath property to provide the path of the endpoint to be used for HTTP/HTTPS requests. The default is `cliservice`.
4. Optionally, if you need to override the default value of the User-Agent header to one used by your service, set the UserAgent property to specify the string value of the User-Agent header to be used in HTTP requests. If no value is specified, the following header value is used: `Progress/8.0 (SparkSQL ODBC driver)`.
5. Optionally, if you are sending requests to HTTPS endpoints, set the EncryptionMethod property to `ssl` to enable SSL data encryption. Data encryption behavior can be further configured using the connection properties described in "Data Encryption Properties" in the user's guide.

The following example demonstrates a connection URL with HTTP mode and SSL enabled.

```
jdbc:datadirect:sparksql://myserver:10000;DatabaseName=mydb1;EncryptionMethod=ssl;
HTTPPath=warehouse;TransportMode=http;
```

See also

DatabaseName on page 95
ServerName on page 113
PortNumber on page 108
TransportMode on page 118
HTTPPath on page 98
EncryptionMethod on page 96

Interactive SQL for JDBC (JDBCISQL)

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with Interactive SQL for JDBC (JDBCISQL). JDBCISQL supports a command line interface that allows you to connect to a data source, execute SQL statements and retrieve results for display on a terminal.

To execute commands with JDBCISQL:

1. Start the ISQL tool. Do one of the following:
   - On Windows, double-click the`jdbcisql.bat` file in the *`install_dir`*`\jdbcisql` folder. Or, from a command prompt, navigate to the *`install_dir`*`\jdbcisql` directory and run the `jdbcisql.bat` file.
   - On Linux and UNIX, change to the *`install_dir`*`\isql` directory and run `jdbcisql.sh`.

   The Interactive SQL prompt appears.
2. Type the driver name class; then, press **Enter**:

   ```
   com.ddtek.jdbc.aha.AhaDriver
   ```

3. Type `connect` followed by the connection URL for the driver; then, press **Enter**. For example:

   ```
   connect jdbc:datadirect:aha://mycompany.aha.io;user=jsmith@progress.com;password=secret;
   ```

   If successful, the tool will return the time required to connect.
4. At the `ISQL>` prompt, issue a SQL command to query or modify the data source; then, press **Enter**. For example:

   ```
   SELECT * FROM FEATURES;
   ```

   Note: SQL commands must be terminated by a semi-colon.

   Note: In addition to SQL commands, JDBCISQL supports a set of proprietary commands. Type `Help` at the prompt for a list of supported commands and syntax.

   The results of the command are displayed in the terminal.
5. After you are finished executing queries and commands, you can disconnect from the data source by typing the following; then, pressing **Enter**:

   ```
   DISCONNECT;
   ```

6. Press any key to end the session.

Using Connection Properties

You can use connection properties to customize the driver for your environment. This section lists the connection properties supported by the driver and describes each property. You can use these connection properties with either the JDBC Driver Manager or a JDBC data source. For a Driver Manager connection, a property is expressed as a key value pair and takes the form *property*=*value*. For a data source connection, a property is expressed as a JDBC method and takes the form `set`*property*`(`*value*`)`.

The following sections describe the connection properties supported by the driver for Apache Spark SQL.

Note: All connection property names are case-insensitive. For example, Password is the same as password. Required properties are noted as such.

Note: The data type listed for each connection property is the Java data type used for the property value in a JDBC data source.

See also

Connecting with the JDBC Driver Manager on page 24
Connecting Using Data Sources on page 28
Connection Property Descriptions on page 81

Required Properties

The following table summarizes connection properties required to connect to a database. The first section describes options required for both binary (TCP) mode and HTTP modes, while the second documents additional options required to establish an HTTP connection.

Table 6: Required Properties

Property	Characteristic
Required properties for all connections	
DatabaseName on page 95	Specifies the name of the database. The database must exist, or the connection attempt will fail.
PortNumber on page 108	The TCP port of the primary database server that is listening for connections to the database. The default is `10000`.
ServerName on page 113	Specifies either the IP address or the server name (if your network supports named servers) of the primary database server.
Additional properties required for enabling HTTP mode	

Property	Characteristic
HTTPPath on page 98	Specifies the path of the HTTP/HTTPS endpoint used for connections when HTTP mode is enabled (`TransportMode=http`). The default is `cliservice`.
TransportMode on page 118	Specifies whether binary (TCP) mode or HTTP mode is used to access Apache Spark SQL data sources. If set to `binary`, Thrift RPC requests are sent to directly to data sources using a binary connection (TCP mode). If set to `http`, Thrift RPC requests are sent using HTTP transport (HTTP mode). HTTP mode is typically used when connecting to a reverse-proxy server, such as a gateway, for improved security, or a load balancer. **Note:** To configure the driver to use HTTPS end points, set `TransportMode=http` and `EncryptionMethod=SSL`. The default is `binary`.

See also

Connection Property Descriptions on page 81

User ID/Password Authentication Properties

The following table describes the connection properties used to configure user ID/Password authentication. The first section describes the properties required for user ID/Password authentication, while the second lists the additional properties required to enable the cookie-based authentication feature.

Table 7: User ID/Password Authentication Properties

Property	Characteristic
User ID/Password Authentication	
AuthenticationMethod on page 88	Determines which authentication method the driver uses when establishing a connection. If set to `kerberos`, the driver uses Kerberos authentication. If set to `userIdPassword`, the driver uses user ID/password authentication. The default is `userIdPassword`.
Password on page 107	Specifies a password that is used to connect to your database.
User on page 121	Specifies the user name that is used to connect to the database.

Property	Characteristic
Additional Properties for Cookie Based Authentication (HTTP Mode only)	
CookieName on page 93	Specifies the name of the cookie used for authenticating HTTP requests when HTTP mode `TransportMode=http` and cookie based authentication are enabled (`EnableCookieAuthentication=true`). The default is `hive.server2.auth`.
EnableCookieAuthentication on page 95	Determines whether the driver attempts to use cookie based authentication for requests to an HTTP endpoint after the initial authentication to the server. Cookie based authentication improves response time by eliminating the need to re-authenticate with the server for each request. The default is `true`.

See also

Connection Property Descriptions on page 81
Using Authentication on page 62

Kerberos Authentication Properties

This section describes the connection properties used to configure Kerberos authentication.

Table 8: Authentication Properties

Property	Characteristic
AuthenticationMethod on page 88	Determines which authentication method the driver uses when establishing a connection. If set to `kerberos`, the driver uses Kerberos authentication. If set to `userIdPassword`, the driver uses user ID/password authentication. The default is `userIdPassword`.
Password on page 107	Specifies a password that is used to connect to your database.
ServicePrincipalName on page 113	Specifies the service principal name to be used for Kerberos authentication.
User on page 121	Specifies the user name that is used to connect to the database.

See also

Connection Property Descriptions on page 81
Using Authentication on page 62

Data Encryption Properties

The following table summarizes connection properties which can be used in the implementation of SSL data encryption, including server and client authentication.

Table 9: Data Encryption Properties

Property	Characteristic
CryptoProtocolVersion on page 94	Specifies a cryptographic protocol or comma-separated list of cryptographic protocols that can be used when TLS/SSL is enabled using the EncryptionMethod connection property. The default is determined by the settings of the JRE.
EncryptionMethod on page 96	Determines whether data is encrypted and decrypted when transmitted over the network between the driver and database server. If set to `noEncryption`, data is not encrypted or decrypted. If set to `SSL`, data is encrypted using SSL. If the database server does not support SSL, the connection fails and the driver throws an exception. If set to `requestSSL`, the login request and data is encrypted using SSL. If the database server does not support SSL, the driver establishes an unencrypted connection. **Note:** Enabling SSL `EncryptionMethod=SSL` and Transport mode (`TransportMode=http`) configures the driver to use HTTPS end points instead of HTTP end points. The default is `noEncryption`.
HostNameInCertificate on page 97	Specifies a host name for certificate validation when SSL encryption is enabled (`EncryptionMethod=SSL`) and validation is enabled (`ValidateServerCertificate=true`). This property is optional and provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.
KeyPassword on page 102	Specifies the password that is used to access the individual keys in the keystore file when SSL is enabled (`EncryptionMethod=SSL`) and SSL client authentication is enabled on the database server. This property is useful when individual keys in the keystore file have a different password than the keystore file.
KeyStore on page 103	Specifies the directory of the keystore file to be used when SSL is enabled (`EncryptionMethod=SSL`) and SSL client authentication is enabled on the database server. The keystore file contains the certificates that the client sends to the server in response to the server's certificate request.

Property	Characteristic
KeyStorePassword on page 104	Specifies the password that is used to access the keystore file when SSL is enabled (`EncryptionMethod=SSL`) and SSL client authentication is enabled on the database server. The keystore file contains the certificates that the client sends to the server in response to the server's certificate request.
TrustStore on page 119	Specifies the directory of the truststore file to be used when SSL is enabled (`EncryptionMethod=SSL`) and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.
TrustStorePassword on page 120	Specifies the password that is used to access the truststore file when SSL is enabled (`EncryptionMethod=SSL`) and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.
ValidateServerCertificate on page 122	Determines whether the driver validates the certificate that is sent by the database server when SSL encryption is enabled (`EncryptionMethod=SSL`). When using SSL server authentication, any certificate that is sent by the server must be issued by a trusted Certificate Authority (CA). The default is `true`.

See also

Connection Property Descriptions on page 81
Data encryption on page 70

Data Type Handling Properties

The following table summarizes connection properties which can be used to handle data types.

Table 10: Data Type Handling Properties

Property	Characteristic
BinaryDescribeType on page 89	Specifies how columns of the Binary type are described. This property affects ResultSetMetaData calls. getTypeInfo() calls are affected only when the property is set to `longvarbinary`. In this configuration, an additional LONGVARBINARY entry is added to the getTypeInfo results. If set to `varbinary`, Binary columns are described as VARBINARY. If set to `longvarbinary`, Binary columns are described as LONGVARBINARY. The default is `varbinary`.

Property	Characteristic
ConvertNull on page 92	Controls how data conversions are handled for null values. If set to `0`, the driver does not perform the data type check if the value of the column is null. This allows null values to be returned even though a conversion between the requested type and the column type is undefined. If set to `1`, the driver checks the data type being requested against the data type of the table column that stores the data. If a conversion between the requested type and column type is not defined, the driver generates an "unsupported data conversion" exception regardless of whether the column value is NULL. The default is `1`.
JavaDoubleToString on page 102	Determines which algorithm the driver uses when converting a double or float value to a string value. By default, the driver uses its own internal conversion algorithm, which improves performance. If set to `true`, the driver uses the JVM algorithm when converting a double or float value to a string value. If your application cannot accept rounding differences and you are willing to sacrifice performance, set this value to `true` to use the JVM conversion algorithm. If set to `false`, the driver uses its own internal algorithm when converting a double or float value to a string value. This value improves performance, but slight rounding differences within the allowable error of the double and float data types can occur when compared to the same conversion using the JVM algorithm. The default is `false`.
MaxBinarySize on page 105	Specifies the maximum length, in bytes, of fields of the Binary data type when described by the driver through result set descriptions and metadata methods. If a field exceeds the specified length, the driver truncates that field. The default is `2147483647`.
StringDescribeType on page 117	Specifies whether String columns are described as VARCHAR or LONGVARCHAR. This property affects resultSetMetaData() calls; it does not affect getTypeInfo() calls. If set to `varchar`, String columns are described as VARCHAR. If set to `longvarchar`, String columns are described as LONGVARCHAR. The default is `varchar`.

See also

Connection Property Descriptions on page 81

HTTP Properties

The following table summarizes connection properties which can be used to configure the driver to use requests using HTTP transport (HTTP mode).

Table 11: Data Type Handling Properties

Property	Characteristic
HTTPPath on page 98	Specifies the path of the HTTP/HTTPS endpoint used for connections when HTTP mode is enabled (`TransportMode=http`). The default is `cliservice`.
TransportMode on page 118	Specifies whether binary (TCP) mode or HTTP mode is used to access Apache Spark SQL data sources. If set to `binary`, Thrift RPC requests are sent to directly to data sources using a binary connection (TCP mode). If set to `http`, Thrift RPC requests are sent using HTTP transport (HTTP mode). HTTP mode is typically used when connecting to a proxy server, such as a gateway, for improved security, or a load balancer. The default is `binary`.
UserAgent on page 122	Specifies the string value of the User-Agent header to be used in HTTP requests. This property provides a method to override the default value of the User-Agent header when required by a service. No default value, which means the driver uses `Progress/8.0 (SparkSQL ODBC driver)` for the value of the User-Agent header.

See also

Connection Property Descriptions on page 81

Timeout Properties

The following table summarizes timeout connection properties.

Table 12: Timeout Properties

Property	Characteristic
LoginTimeout on page 105	Specifies the amount of time, in seconds, that the driver waits for a connection to be established before timing out the connection request. If set to `0`, the driver does not time out a connection request. If set to *x*, the driver waits for the specified number of seconds before returning control to the application and throwing a timeout exception. The default is `0`.

See also

- Connection Property Descriptions on page 81

Client Information Properties

The following table summarizes connection properties which can be used to return client information.

Table 13: Client Information Properties

Property	Characteristic
AccountingInfo on page 85	Defines accounting information. This value is stored locally and is used for database administration/monitoring purposes.
ApplicationName on page 86	Specifies the name of the application. This value is stored locally and is used for database administration/monitoring purposes.
ClientHostName on page 89	Specifies the host name of the client machine. This value is stored locally and is used for database administration/monitoring purposes.
ClientUser on page 90	Specifies the user ID. This value is stored locally and is used for database administration/monitoring purposes.
ProgramID on page 108	Specifies the driver and version information on the client. This value is stored locally and is used for database administration/monitoring purposes.

See also

Connection Property Descriptions on page 81

Statement Pooling Properties

The following table summarizes statement pooling connection properties.

Table 14: Statement Pooling Properties

Property	Characteristic
ImportStatementPool on page 99	Specifies the path and file name of the file to be used to load the contents of the statement pool. When this property is specified, statements are imported into the statement pool from the specified file.

Property	Characteristic
MaxPooledStatements on page 106	Specifies the maximum number of prepared statements to be pooled for each connection and enables the driver's internal prepared statement pooling when set to an integer greater than zero (0). The driver's internal prepared statement pooling provides performance benefits when the driver is not running from within an application server or another application that provides its own statement pooling. If set to `0`, the driver's internal prepared statement pooling is not enabled. If set to *x*, the driver's internal prepared statement pooling is enabled and the driver uses the specified value to cache up to that many prepared statements created by an application. If the value set for this property is greater than the number of prepared statements that are used by the application, all prepared statements that are created by the application are cached. Because CallableStatement is a sub-class of PreparedStatement, CallableStatements also are cached. The default is `0`.
RegisterStatementPoolMonitorMBean on page 111	Registers the Statement Pool Monitor as a JMX MBean when statement pooling has been enabled with MaxPooledStatements. This allows you to manage statement pooling with standard JMX API calls and to use JMX-compliant tools, such as JConsole. If set to `true`, the driver registers an MBean for the statement pool monitor for each statement pool. This gives applications access to the Statement Pool Monitor through JMX when statement pooling is enabled. If set to `false`, the driver does not register an MBean for the Statement Pool Monitor for any statement pool. The default is `false`.

See also

Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

See also

Connection Property Descriptions on page 81
Performance Considerations on page 61

Proxy Server Properties

The following table summarizes proxy server connection properties.

Table 15: Proxy Server Properties

Property	Characteristic
ProxyHost on page 109	Specifies a proxy server.
ProxyPassword on page 110	Specifies the password needed to connect to a proxy server.
ProxyPort on page 110	Specifies the port number where the proxy server is listening for HTTP or HTTPS requests.
ProxyUser on page 111	Specifies the user name needed to connect to a proxy server.

See also

Connection Property Descriptions on page 81

Additional Properties

The following table summarizes additional connection properties.

Table 16: Additional Properties

Property	Characteristic
ArrayFetchSize on page 87	Specifies the number of fields the driver uses to calculate the maximum number of rows for a fetch. When executing a fetch, the driver divides the ArrayFetchSize value by the number of columns in a particular table to determine the number of rows to retrieve. By determining the fetch size based on the number of fields, the driver can avoid out of memory errors when fetching from tables containing a large number of columns while continuing to provide improved performance when fetching from tables containing a small number of columns. If set to *-x*, the driver overrides any settings on the statement level and uses the number of fields specified by the absolute value of *-x* to calculate the number of rows to retrieve. If set to *x*, the driver uses the number of fields specified by the value of *x* to calculate the number of rows to retrieve. However, the driver will not override settings, such as setFetchSize(), on the statement level. The default is `20000`.

Property	Characteristic
ConnectionRetryCount on page 91	Specifies the number of times the driver retries connection attempts to the server until a successful connection is established. . If set to `0`, the driver does not try to reconnect after the initial unsuccessful attempt. If set to *x*, the driver retries connection attempts the specified number of times. If a connection is not established during the retry attempts, the driver returns an exception that is generated by the last server to which it tried to connect. The default is `5`.
ConnectionRetryDelay on page 92	Specifies the number of seconds the driver waits between connection retry attempts when ConnectionRetryCount is set to a positive integer. If set to `0`, the driver does not delay between retries. If set to *x*, the driver waits between connection retry attempts the specified number of seconds. The default is `1`.
InitializationString on page 100	Specifies one or multiple SQL commands to be executed by the driver after it has established the connection to the database and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver throws an exception indicating which SQL command or commands failed.
InsensitiveResultSetBufferSize on page 101	Determines the amount of memory used by the driver to cache insensitive result set data. If set to `-1`, the driver caches insensitive result set data in memory. If set to `0`, the driver caches insensitive result set data in memory, up to a maximum of 2 MB. If set to *x*, the driver caches insensitive result set data in memory and uses this value to set the size (in KB) of the memory buffer for caching insensitive result set data. The default is `2048`.
RemoveColumnQualifiers on page 112	Specifies whether the driver removes 3-part column qualifiers and replaces them with alias.column qualifiers. If set to `true` (enabled), the driver removes 3-part column qualifiers and replaces them with alias.column qualifiers. If set to `false`, the driver does not modify the request. The default is `false`.

Property	Characteristic
SpyAttributes on page 114	Enables DataDirect Spy to log detailed information about calls issued by the driver on behalf of the application. DataDirect Spy is not enabled by default.
TransactionMode on page 117	Specifies how the driver handles manual transactions. If set to `ignore`, the data source does not support transactions and the driver always operates in auto-commit mode. If set to `noTransactions`, the data source and the driver do not support transactions. The default is `noTransactions`.
UseCurrentSchema on page 120	Specifies whether results are restricted to the tables and views in the current schema if a call is made without specifying a schema or if the schema is specified as the wildcard character %. Restricting results to the tables and views in the current schema improves performance of calls that do not specify a schema. If set to `true`, the results that are returned from getTables() and getColumns() methods are restricted to the tables and views in the current schema. If set to `false`, the results that are returned from getTables() and getColumns() methods are not restricted. The default is `false`.

See also

Connection Property Descriptions on page 81

Performance Considerations

You can optimize application performance by considering the impact of the following connection properties.

ArrayFetchSize: To improve throughput, consider increasing the value of ArrayFetchSize. By increasing the value of ArrayFetchSize, you increase the number of rows the driver will retrieve from the server for a fetch. In turn, increasing the number of rows that the driver can retrieve reduces the number, and expense, of network round trips. For example, if an application attempts to fetch 100,000 rows, it is more efficient for the driver to retrieve 2000 rows over the course of 50 round trips than to retrieve 500 rows over the course of 200 round trips. Note that improved throughput does come at the expense of increased demands on memory and slower response time. Furthermore, if the fetch size exceeds the available buffer memory of the server, an out of memory error is returned when attempting to execute a fetch. If you receive this error, decrease the value specified until fetches are successfully executed.

For many applications, throughput is the primary performance measure. In contrast, response time (how fast the first set of data is returned) can be of greater importance for interactive or Web-based applications. Since smaller fetch sizes are generally returned more quickly than larger ones, you can improve response time by decreasing the value of ArrayFetchSize.

BinaryDescribeType: When BinaryDescribeType is set to `longvarbinary`, the driver not only maps Binary to Longvarbinary, but also allocates more space to cache the long data. Because more space is allocated for the long data, your application will incur a performance penalty.

EncryptionMethod: Data encryption may adversely affect performance because of the additional overhead (mainly CPU usage) required to encrypt and decrypt data.

InsensitiveResultSetBufferSize: To improve performance when using scroll-insensitive result sets, the driver can cache the result set data in memory instead of writing it to disk. By default, the driver caches 2 MB of insensitive result set data in memory and writes any remaining result set data to disk. Performance can be improved by increasing the amount of memory used by the driver before writing data to disk or by forcing the driver to never write insensitive result set data to disk. The maximum cache size setting is 2 GB.

MaxPooledStatements: To improve performance, the driver's own internal prepared statement pooling should be enabled when the driver does not run from within an application server or from within another application that provides its own prepared statement pooling. When the driver's internal prepared statement pooling is enabled, the driver caches a certain number of prepared statements created by an application. For example, if the MaxPooledStatements property is set to `20`, the driver caches the last 20 prepared statements created by the application. If the value set for this property is greater than the number of prepared statements used by the application, all prepared statements are cached.

Refer to "Designing JDBC Applications for Performance Optimization" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using prepared statement pooling to optimize performance.

StringDescribeType: To obtain data from String columns with the getClob() method, the StringDescribeType connection property must be set to `longvarchar`. (Otherwise, calling getClob() results in an "unsupported data conversion" exception.) When StringDescribeType is set to `longvarchar`, the driver not only maps String to Longvarchar but also allocates more space to cache the long data. Because more space is allocated for the long data, your application will incur a performance penalty.

UseCurrentSchema: If your application needs to access tables and views owned only by the current user, performance of your application can be improved by setting this property to `true`. When this property is set to `true`, the driver returns only tables and views owned by the current user when executing getTables() and getColumns() methods. Setting this property to `true` is equivalent to passing the user ID used on the connection as the schemaPattern argument to the getTables() or getColumns() call.

See also

ArrayFetchSize on page 87
BinaryDescribeType on page 89
EncryptionMethod on page 96
InsensitiveResultSetBufferSize on page 101
MaxPooledStatements on page 106
StringDescribeType on page 117
UseCurrentSchema on page 120

Using Authentication

The driver supports the following authentication methods:

- *User ID/password authentication* authenticates the user to the database using a database user name and password.
- *Kerberos* is a trusted third-party authentication service. The driver supports Windows Active Directory Kerberos and MIT Kerberos implementations.

Using the AuthenticationMethod Property

The AuthenticationMethod connection property controls which authentication mechanism the driver uses when establishing connections.

When AuthenticationMethod=`kerberos`, the driver uses Kerberos authentication when establishing a connection. The driver ignores any values specified by the User and Password properties.

When AuthenticationMethod=`userIdPassword` (default), the driver uses user ID/password authentication when establishing a connection. The User property provides the user ID. The Password property provides the password. If user ID or password is not specified, the driver passes the word "anonymous" for the missing item(s) when submitting requests to the server.

Configuring User ID/Password Authentication

Take the following steps to configure User ID/Password authentication:

1. Set the AuthenticationMethod property to `userIdPassword`.
2. Set the DatabaseName connection property.
3. Set the ServerName connection property.
4. Set the User property to provide the user ID.
5. Set the Password property to provide the password.
6. Optionally, if the driver is configured to use HTTP mode (`TransportMode=http`), set the CookieName property to provide the name of the cookie used for authenticating HTTP requests. The default is `hive.server2.auth`.

See also

AuthenticationMethod on page 88
DatabaseName on page 95
User on page 121
Password on page 107
CookieName on page 93

Configuring the Driver for Kerberos Authentication

To configure the driver for Kerberos authentication, take the following steps.

1. Verify that your environment meets the requirements outlined in "Kerberos Authentication Requirements."
2. Modify your JAAS login configuration file to include the JAAS login module information needed for your environment. You can create your own login configuration file, or you can use the `JDBCDriverLogin.conf` file installed with the driver. This file is installed in the `/lib` directory of the product installation directory.

Note: For more information on the JAAS login configuration file, see "Java Authentication and Authorization Service (JAAS) Login Configuration File."

Whether you are using the JDBCDriverLogin.conf file or another file, the login configuration file must contain the entry JDBC_DRIVER_01 with JAAS login module information. The following examples show that the JAAS login module information depends on your JRE.

Oracle JRE

```
JDBC_DRIVER_01 {
  com.sun.security.auth.module.Krb5LoginModule required useTicketCache=true;
};
```

IBM JRE

```
JDBC_DRIVER_01 {
  com.ibm.security.auth.module.Krb5LoginModule required useDefaultCcache=true;
};
```

3. Once the JAAS configuration file includes the JAAS login module information, the file must be referenced using one of the following methods.

   - **Option 1.** Specify a login configuration file directly in your application with the java.security.auth.login.config system property. For example:

     ```
     System.setProperty("java.security.auth.login.config","install_dir/lib/JDBCDriverLogin.conf");
     ```

   - **Option 2.** Set up a default configuration. Modify the Java security properties file to indicate the URL of the login configuration file with the login.config.url.*n* property where *n* is an integer connoting separate, consecutive login configuration files.

     Note: When more than one login configuration file is specified, then the files are read and concatenated into a single configuration. At least one of the login configuration files must include the JDBC_DRIVER_01 entry with the JAAS login module.

     1. Open the Java security properties file. The security properties file is the java.security file in the /jre/lib/security directory of your Java installation.
     2. Find the line # Default login configuration file in the security properties file.
     3. Below the # Default login configuration file line, add the URL of the login configuration file as the value for a login.config.url.*n* property. For example:

        ```
        # Default login configuration file
        login.config.url.1=file:${user.home}/.java.login.config
        login.config.url.2=file:install_dir/lib/JDBCDriverLogin.conf
        ```

4. Set the Kerberos realm name and the KDC name for that realm using either of the following methods.

   Note: If using Windows Active Directory, the Kerberos realm name is the Windows domain name and the KDC name is the Windows domain controller name.

   - **Option 1.** Modify the krb5.conf file to include the default realm name and the KDC name for that realm. (See "The krb5.conf File" for details about using and locating the krb5.conf file.)

For example, if the realm name is *XYZ.COM* and the KDC name is *kdc1*, your `krb5.conf` file would include the following entries.

```
[libdefaults]
default_realm = XYZ.COM

[realms]
XYZ.COM = {
kdc = kdc1
}
```

- **Option 2.** Specify the Java system properties, `java.security.krb5.realm` and `java.security.krb5.kdc`, in your application. For example, if the realm name is *XYZ.COM* and the KDC name is *kdc1*, your application would include the following settings.

```
System.setProperty("java.security.krb5.realm","XYZ.COM");
System.setProperty("java.security.krb5.kdc","kdc1")
```

Note: Even if you do not use the `krb5.conf` file to specify the realm and KDC names, you may need to modify your `krb5.conf` file to suit your environment. Refer to your database vendor documentation for information.

If you do not specify a valid Kerberos realm and a valid KDC name, the following exception is thrown.

```
Message:[DataDirect][MongoDB JDBC Driver]Could not establish a connection using
integrated security: No valid credentials provided
```

5. If you want the driver to use user credentials other than the server user's operating system credentials, include code in your application to obtain and pass a `javax.security.auth.Subject` used for authentication. (See "Specifying User Credentials for Kerberos Authentication (Delegation of Credentials)" for details.)
6. Set the driver's AuthenticationMethod connection property to `kerberos`. (See "AuthenticationMethod" and "Using the AuthenticationMethod Property" for details.)

Note: When Kerberos authentication is enabled through the driver (`AuthenticationMethod=kerberos`), the driver automatically detects and abides by the server's SASL-QOP configuration at connection time. See "Kerberos SASL-QOP" for details.

---

7. Specify the service principal name with the ServicePrincipalName connection property. (See "ServicePrincipalName" for details on how to formulate and specify the service principal name.)
8. If using Kerberos authentication with a Security Manager on a Java Platform, you must grant security permissions to the application and driver. See "Permissions for Kerberos Authentication" for an example.
9. Establish a procedure for obtaining a Kerberos Ticket Granting Ticket (TGT) for your environment. (See "Obtaining a Kerberos Ticket Granting Ticket" for details.)
   - **Scenario 1.** If an application uses Kerberos authentication from a Windows client and Kerberos authentication is provided by Windows Active Directory, Windows Active Directory automatically obtains a TGT.
   - **Scenario 2.** When Kerberos authentication is provided by MIT Kerberos, you can allow the application to obtain a TGT in either of the following ways.
     1. Automate the method of obtaining the TGT as with a keytab. (See your Kerberos documentation for details.)
     2. Require the application user to obtain the TGT with a `kinit` command when logging on.

A TGT can be obtained with a `kinit` command to the Kerberos server. For example, the following command requests a TGT from the server with a lifetime of 10 hours, which is renewable for 5 days.

```
kinit -l 10h -r 5d user
```

Note: The `klist` command can be used on Windows or UNIX/Linux systems to verify that a TGT has been obtained.

See also

Kerberos Authentication Requirements on page 66
The JAAS Login Configuration File on page 67
The krb5.conf File on page 67
Specifying User Credentials for Kerberos Authentication (Delegation of Credentials) on page 68
AuthenticationMethod on page 88
Using the AuthenticationMethod Property on page 63
Kerberos SASL-QOP on page 69
ServicePrincipalName on page 113
Permissions for Kerberos Authentication on page 37
Obtaining a Kerberos Ticket Granting Ticket on page 69

Kerberos Authentication Requirements

Verify that your environment meets the requirements listed in the following table before you configure the driver for Kerberos authentication.

Note: The domain controller must administer both the database server and the client.

Table 17: Kerberos Configuration Requirements

Component	Requirements
Database server	The database server must be running Apache Spark SQL 1.2 or higher.

Kerberos server	The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos KDC. Network authentication must be provided by one of the following methods:
	• Windows Active Directory on one of the following operating systems:
	  • Windows Server 2003 or higher
	  • Windows 2000 Server Service Pack 3 or higher
	• MIT Kerberos 1.5 or higher

Client	Java SE 8 or higher must be installed.

See also

Configuring the Driver for Kerberos Authentication on page 63

The JAAS Login Configuration File

A Kerberos environment must include a Java Authentication and Authorization Service (JAAS) login configuration file. This login configuration file specifies the JAAS login module for Kerberos authentication.

You can create your own JAAS login configuration file, or you can use the `JDBCDriverLogin.conf` file installed with the driver. This file is installed in the `/lib` directory of the product installation directory.

Whether you are using the `JDBCDriverLogin.conf` file or another file, the login configuration file must contain the entry `JDBC_DRIVER_01` with JAAS login module information. The following examples show that the JAAS login module information depends on your JRE.

Oracle JRE

```
JDBC_DRIVER_01 {
  com.sun.security.auth.module.Krb5LoginModule required useTicketCache=true;
};
```

IBM JRE

```
JDBC_DRIVER_01 {
  com.ibm.security.auth.module.Krb5LoginModule required useDefaultCcache=true;
};
```

Once the JAAS configuration file includes the `JDBC_DRIVER_01` entry and JAAS login module information, the file must be referenced either directly in the application with the `java.security.auth.login.config` system property or through the Java security properties file with the `login.config.url.`*n* property. See "Configuring the Driver for Kerberos Authentication" for step-by-step instructions.

Refer to "JAAS Login Configuration File" in your Java documentation for information about setting options in the login configuration file.

See also

Configuring the Driver for Kerberos Authentication on page 63

The krb5.conf File

The `krb5.conf` file contains Kerberos configuration information. Typically, the default realm name and the KDC name for that realm are specified in the `krb5.conf` file. However, you can specify the realm and KDC names directly in your application with the `java.security.krb5.realm` and `java.security.krb5.kdc` system properties. Setting these system properties will override the settings in the `krb5.conf` file.

When a client application does not use the `java.security.krb5.realm` and `java.security.krb5.kdc` system properties, the JVM looks for a `krb5.conf` file that contains the realm and KDC names. The JVM first looks for the `krb5.conf` file in the location specified with the `java.security.krb5.conf` system property. If this system property has not been used, then the JVM continues looking for the `krb5.conf` file using an internal algorithm. Refer to your vendor's JVM documentation for the list of directories that the JVM searches in order to find the `krb5.conf` file.

During installation, a `krb5.conf` file is installed in the `/lib` directory of the product installation directory. The installed `krb5.conf` file contains generic syntax for setting the default realm name and the KDC name for that realm. If you are not already using another `krb5.conf` file for your Kerberos implementation, you can modify it to suit your environment. However, you will either need to specify the location of this file using the `java.security.krb5.conf` system property, or you will need to add the file to a directory where it may be found by your JVM.

Depending on your environment, other modifications may need to be made to your `krb5.conf` file. Refer to the following resources for more information on the Kerberos configuration and the `krb5.conf` file.

- Your database vendor documentation
- "Keberos Requirements" in *Java™ Documentation*
- "krb5.conf" in *MIT Kerberos Documentation*

See also

Configuring the Driver for Kerberos Authentication on page 63

Specifying User Credentials for Kerberos Authentication (Delegation of Credentials)

By default, when Kerberos authentication is used, the driver takes advantage of the user name and password maintained by the operating system to authenticate users to the database. By allowing the database to share the user name and password used for the operating system, users with a valid operating system account can log into the database without supplying a user name and password.

Many application servers or Web servers act on behalf of the client user logged on the machine on which the application is running, rather than the server user. If you want the driver to use user credentials other than the server the operating system user name and password, include code in your application to obtain and pass a javax.security.auth.Subject used for authentication as shown in the following example.

```
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import java.sql.*;
//  The following code creates a javax.security.auth.Subject instance
//  used for authentication. Refer to the Java Authentication
//  and Authorization Service documentation for details on using a
//  LoginContext to obtain a Subject.
LoginContext lc = null;
Subject subject = null;
try {
    lc = new LoginContext("JaasSample", new TextCallbackHandler());
    lc.login();
    subject = lc.getSubject();
}
catch (Exception le) {
    ... // display login error
}//  This application passes the javax.security.auth.Subject
//  to the driver by executing the driver code as the subject
Connection con =
   (Connection) Subject.doAs(subject, new PrivilegedExceptionAction() {

   public Object run() {

        Connection con = null;
    try {
        Class.forName("com.ddtek.jdbc.sparksql.SparkSQLDriver");
        String url = "jdbc:datadirect:sparksql://myServer:10000";
        con = DriverManager.getConnection(url);
       }
```

```
      catch (Exception except) {
      ... //log the connection error
             Return null;
         }

          return con;
     }
});
//  This application now has a connection that was authenticated with
//  the subject. The application can now use the connection.
Statement   stmt = con.createStatement();
String      sql = "SELECT * FROM employee";
ResultSet   rs = stmt.executeQuery(sql);
... // do something with the results
```

Obtaining a Kerberos Ticket Granting Ticket

Kerberos uses the credentials in a Ticket Granting Ticket (TGT) to verify the identity of users and control access to services. Depending on your environment, you will need to establish a procedure for obtaining a TGT.

If an application uses Kerberos authentication from a Windows client and Kerberos authentication is provided by Windows Active Directory, Windows Active Directory automatically obtains a TGT for the user.

In contrast, if Kerberos authentication is provided by MIT Kerberos, you can allow the application to obtain a TGT in one of two ways. First, you can automate the method of obtaining the TGT as with a keytab. Second, you can require the application user to obtain the TGT with a `kinit` command when logging on.

A TGT can be obtained directly with a `kinit` command to the Kerberos server. For example, the following command requests a TGT from the server with a lifetime of 10 hours, which is renewable for 5 days.

```
kinit -l 10h -r 5d user
```

Note: The `klist` command can be used on Windows or UNIX/Linux systems to verify that a TGT has been obtained.

Refer to your Kerberos documentation for more information on using a keytab file to automate the process of obtaining a TGT.

Kerberos SASL-QOP

The driver supports Kerberos SASL-QOP data integrity and confidentiality. SASL-QOP is configured on the server side as a part of SparkSQL Kerberos configuration. When Kerberos authentication is enabled through the driver (`AuthenticationMethod=kerberos`), the driver automatically detects and abides by the server's SASL-QOP configuration at connection time. In addition to supporting the auth value, the driver supports the auth-int and auth-conf SASL-QOP values. SASL-QOP values are defined as follows:

- `auth`: authentication only (default)
- `auth-int`: authentication with integrity protection
- `auth-conf`: authentication with confidentiality protection

See also

Configuring the Driver for Kerberos Authentication on page 63
AuthenticationMethod on page 88

Data encryption

The driver supports Secure Sockets Layer (SSL) data encryption. SSL works by allowing the client and server to send each other encrypted data that only they can decrypt. SSL negotiates the terms of the encryption in a sequence of events known as the *SSL handshake*. The handshake involves the following types of authentication:

- *SSL server authentication* requires the server to authenticate itself to the client.
- *SSL client authentication* is optional and requires the client to authenticate itself to the server after the server has authenticated itself to the client.

Configuring SSL encryption

The following steps outline how to configure SSL encryption.

Note: Connection hangs can occur when the driver is configured for SSL and the database server does not support SSL. You may want to set a login timeout using the LoginTimeout property to avoid problems when connecting to a server that does not support SSL.

To configure SSL encryption:

Important: The driver complies with FIPS when FIPS mode is enabled with the client JVM. See "FIPS (Federal Information Processing Standard)" for more information.

---

1. Set the EncryptionMethod property to `SSL`.
2. Use the CryptoProtocolVersion property to specify acceptable cryptographic protocol versions (for example, TLSv1.2) supported by your server.
3. Specify the location and password of the truststore file used for SSL server authentication. Either set the TrustStore and TrustStorePassword properties or their corresponding Java system properties (javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword, respectively).
4. To validate certificates sent by the database server, set the ValidateServerCertificate property to `true`.
5. Optionally, set the HostNameInCertificate property to a host name to be used to validate the certificate. The HostNameInCertificate property provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.
6. If your database server is configured for SSL client authentication, configure your keystore information:
   a) Specify the location and password of the keystore file. Either set the KeyStore and KeyStorePassword properties or their corresponding Java system properties (javax.net.ssl.keyStore and javax.net.ssl.keyStorePassword, respectively).
   b) If any key entry in the keystore file is password-protected, set the KeyPassword property to the key password.

Configuring SSL server authentication

When the client makes a connection request, the server presents its public certificate for the client to accept or deny. The client checks the issuer of the certificate against a list of trusted Certificate Authorities (CAs) that resides in an encrypted file on the client known as a *truststore*. Optionally, the client may check the subject (owner) of the certificate. If the certificate matches a trusted CA in the truststore (and the certificate's subject matches the value that the application expects), an encrypted connection is established between the client and server. If the certificate does not match, the connection fails and the driver throws an exception.

To check the issuer of the certificate against the contents of the truststore, the driver must be able to locate the truststore and unlock the truststore with the appropriate password. You can specify truststore information in either of the following ways:

- Specify values for the Java system properties javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword. For example:

  ```
  java -Djavax.net.ssl.trustStore=C:\Certificates\MyTruststore
       -Djavax.net.ssl.trustStorePassword=MyTruststorePassword
  ```

  This method sets values for all SSL sockets created in the JVM.

- Specify values for the connection properties TrustStore and TrustStorePassword in the connection URL. For example:

  ```
  jdbc:datadirect:sparksql://Server3:10000;DatabaseName=Test;
     Password=secr3t;TrustStore=C:\Certficates\MyTruststore.jks;
     TrustStorePassword=MyTruststorePassword;User=jsmith;
  ```

  Any values specified by the TrustStore and TrustStorePassword properties override values specified by the Java system properties. This allows you to choose which truststore file you want to use for a particular connection.

Alternatively, you can configure the drivers to trust any certificate sent by the server, even if the issuer is not a trusted CA. Allowing a driver to trust any certificate sent from the server is useful in test environments because it eliminates the need to specify truststore information on each client in the test environment. If the driver is configured to trust any certificate sent from the server, the issuer information in the certificate is ignored.

Configuring SSL client authentication

If the server is configured for SSL client authentication, the server asks the client to verify its identity after the server has proved its identity. Similar to SSL server authentication, the client sends a public certificate to the server to accept or deny. The client stores its public certificate in an encrypted file known as a *keystore*.

The driver must be able to locate the keystore and unlock the keystore with the appropriate keystore password. Depending on the type of keystore used, the driver also may need to unlock the keystore entry with a password to gain access to the certificate and its private key.

The drivers can use the following types of keystores:

- Java Keystore (JKS) contains a collection of certificates. Each entry is identified by an alias. The value of each entry is a certificate and the certificate's private key. Each keystore entry can have the same password as the keystore password or a different password. If a keystore entry has a password different than the keystore password, the driver must provide this password to unlock the entry and gain access to the certificate and its private key.
- PKCS #12 keystores. To gain access to the certificate and its private key, the driver must provide the keystore password. The file extension of the keystore must be .pfx or .p12.

You can specify this information in either of the following ways:

- Specify values for the Java system properties javax.net.ssl.keyStore and javax.net.ssl.keyStorePassword. For example:

```
java -Djavax.net.ssl.keyStore=C:\Certificates\MyKeystore
     -Djavax.net.ssl.keyStorePassword=MyKeystorePassword
```

  This method sets values for all SSL sockets created in the JVM.

Note: If the keystore specified by the javax.net.ssl.keyStore Java system property is a JKS and the keystore entry has a password different than the keystore password, the KeyPassword connection property must specify the password of the keystore entry (for example, `KeyPassword=MyKeyPassword`).

FIPS (Federal Information Processing Standard)

The Federal Information Processing Standard (or FIPS) is a cryptography standard created by the U.S. government. FIPS specifications require certain secure algorithms, cryptographic modules, and random number generation. The driver is FIPS compliant for data encryption when FIPS is enabled for the JVM on the client machine.

The following applies when the driver is running in a FIPS environment:

- The driver complies with 140-3 and 140-2 standards.
- The driver uses PKCS #11 providers to access keystores.

The driver was tested with FIPS 140-3 enabled using Red Hat OpenJDK 21 on a Red Hat Universal Base Image 9 instance.

Proxy server support

In some environments, your application may need to connect through a proxy server, for example, if your application accesses an external resource such as a Web service. At a minimum, your application needs to provide the following connection information when you invoke the JVM if the application connects through a proxy server:

- Server name or IP address of the proxy server
- Port number on which the proxy server is listening for HTTP/HTTPS requests

In addition, if authentication is required, your application may need to provide a valid user ID and password for the proxy server. Consult with your system administrator for the required information.

For example, the following command invokes the JVM while specifying a proxy server named `pserver`, a port of `8888`, and provides a user ID and password for authentication:

```
java -Dhttp.proxyHost=pserver -Dhttp.proxyPort=8888 -Dhttp.proxyUser=smith
-Dhttp.proxyPassword=secret -cp sparksql.jar com.acme.myapp.Main
```

Alternatively, you can use the ProxyHost, ProxyPort, ProxyUser, and ProxyPassword connection properties. See Connection Property Descriptions on page 81 and Proxy Server Properties on page 59 for details about these properties.

Using Client Information

Many databases allow applications to store client information associated with a connection. For example, the following types of information can be useful for database administration and monitoring purposes:

- Name of the application currently using the connection.
- User ID for whom the application using the connection is performing work. The user ID may be different than the user ID that was used to establish the connection.
- Host name of the client on which the application using the connection is running.
- Product name and version of the driver on the client.
- Additional information that may be used for accounting or troubleshooting purposes, such as an accounting ID.

How Databases Store Client Information

Typically, databases that support storing client information do so by providing a register, a variable, or a column in a system table in which the information is stored. If an application attempts to store information and the database does not provide a mechanism for storing that information, the driver caches the information locally. Similarly, if an application returns client information and the database does not provide a mechanism for storing that information, the driver returns the locally cached value.

For example, let's assume that the following code returns a pooled connection to a database and sets a client application name for that connection. In this example, the application sets the application name SALES157 using the driver property ApplicationName.

```
// Get Database Connection
Connection con = DriverManager.getConnection(
   "jdbc:datadirect:sparksql://Server3:10000;DatabaseName=MyDB;
   ApplicationName=SALES157","TEST","secret");
...
```

The application name SALES157 is stored locally by the database. When the connection to the database is closed, the client information on the connection is reset to an empty string.

Storing Client Information

Your application can store client information associated with a connection using any of the following methods:

- Using the driver connection properties listed in the following table.
- Using the following JDBC methods:
  - Connection.setClientInfo(*properties*)
  - Connection.setClientInfo(*property_name*, *value*)
- Using the JDBC extension methods provided in the com.ddtek.jdbc.extensions package.

Table 18: Client Information Properties

Property	Characteristic
AccountingInfo on page 85	Defines accounting information. This value is stored locally and is used for database administration/monitoring purposes.
ApplicationName on page 86	Specifies the name of the application. This value is stored locally and is used for database administration/monitoring purposes.
ClientHostName on page 89	Specifies the host name of the client machine. This value is stored locally and is used for database administration/monitoring purposes.
ClientUser on page 90	Specifies the user ID. This value is stored locally and is used for database administration/monitoring purposes.
ProgramID on page 108	Specifies the driver and version information on the client. This value is stored locally and is used for database administration/monitoring purposes.

Refer to "JDBC support" in the *Progress DataDirect for JDBC Drivers Reference* for more information about JDBC methods.

See also

Connection Property Descriptions on page 81

Returning client information

Your application can return client information in the following ways:

- Using the following JDBC methods:
  - Connection.getClientInfo()
  - Connection.getClientInfo(*property_name*)
  - DatabaseMetaData.getClientInfoProperties()
- Using the JDBC extension methods provided in the com.ddtek.jdbc.extensions package.

Refer to "JDBC support" in the *Progress DataDirect for JDBC Drivers Reference* for more information about JDBC methods.

Returning MetaData About Client Information Locations

You may want to return metadata about the register, variable, or column in which the database stores client information. For example, you may want to determine the maximum length allowed for a client information value before you store that information. If your application attempts to set a client information value that exceeds the maximum length allowed by the database, that value is truncated and the driver generates a warning. Determining the maximum length of the value beforehand can avoid this situation.

To return metadata about client information, call the DatabaseMetaData.getClientInfoProperties() method:

```
// Get Database Connection
Connection con = DriverManager.getConnection(
   "jdbc:datadirect:sparksql://Server3:10000;DatabaseName=jdbc", "test", "secret");
DatabaseMetaData metaData = con.getMetaData();
ResultSet rs = metaData.getClientInfoProperties();
...
```

The driver returns a result set that provides the following information for each client information property supported by the database:

- Property name
- Maximum length of the property value
- Default property value
- Property description

IP Addresses

The driver supports Internet Protocol (IP) addresses in IPv4 format.

The server name specified in a connection URL, or data source, can resolve to an IPv4 address. In the following example, the server name Server3 can resolve to an IPv4 address:

```
jdbc:datadirect:sparksql://Server3:10000;
   DatabaseName=Test;User=admin;Password=secret
```

Alternately, you can specify addresses using IPv4 format in the server portion of the connection URL. For example, the following connection URL specifies the server using an IPv4 address:

```
jdbc:datadirect:sparksql://123.456.78.90:10000;
   DatabaseName=Test;User=admin;Password=secret
```

You also can specify addresses using the ServerName data source property. The following example shows a data source definition that specifies the server name using an IPv4 address:

```
SparkSQLDataSource mds = new SparkSQLDataSource();
mds.setDescription("My SparkSQLDataSource");
mds.setServerName("123.456.78.90");
...
```

Parameter metadata support

The driver supports returning parameter metadata as described in this section.

Insert, Update, and Delete Statements

The driver supports returning parameter metadata for the following forms of Insert, Update, and Delete statements [9]:

[9] While the driver supports returning parameter metadata for these statements, the SQL commands themselves may not be supported for some versions of Apache Spark SQL.

- `INSERT INTO foo VALUES (?, ?, ?)`
- `INSERT INTO foo (col1, col2, col3) VALUES (?, ?, ?)`
- `UPDATE foo SET col1=?, col2=?, col3=? WHERE col1 operator ? [{AND | OR} col2 operator ?]`
- `DELETE FROM foo WHERE col1 operator ?`

where *operator* is any of the following SQL operators: =, <, >, <=, >=, and <>.

Select Statements

The driver supports returning parameter metadata for Select statements that contain parameters in ANSI SQL-92 entry-level predicates, for example, such as COMPARISON, BETWEEN, IN, LIKE, and EXISTS predicate constructs. Refer to the ANSI SQL reference for detailed syntax.

Parameter metadata can be returned for a Select statement if the statement contains a predicate value expression that can be targeted against the source tables in the associated FROM clause. For example:

```
SELECT * FROM foo WHERE bar > ?
```

In this case, the value expression "bar" can be targeted against the table "foo" to determine the appropriate metadata for the parameter.

The following Select statements show further examples for which parameter metadata can be returned:

```
SELECT col1, col2 FROM foo WHERE col1 = ? AND col2 > ?
SELECT ... WHERE colname LIKE ?
SELECT ... WHERE colname BETWEEN ? AND ?
SELECT ... WHERE colname IN (?, ?, ?)
```

Subqueries are supported, but they can only exist in the From clause. In the following example, the second Select statement is a subquery:

```
SELECT * FROM (SELECT * FROM T1 UNION ALL SELECT * FROM T2) sq
```

ANSI SQL-92 entry-level predicates in a WHERE clause containing GROUP BY, HAVING, or ORDER BY statements are supported. For example:

```
SELECT * FROM t1 WHERE col = ? ORDER BY 1
```

Joins are supported. For example:

```
SELECT * FROM t1,t2 WHERE t1.col1 = ?
```

Fully qualified names and aliases are supported. For example:

```
SELECT A.a, B.b, FROM T1 AS A, T2 AS B WHERE A.a = ? AND B.b = ?"
```

ResultSet metadata support

If your application requires table name information, the driver can return table name information in ResultSet metadata for Select statements. The Select statements for which ResultSet metadata is returned may contain aliases, joins, and fully qualified names. The following queries are examples of Select statements for which the ResultSetMetaData.getTableName() method returns the correct table name for columns in the Select list:

```
SELECT id, name FROM Employee
SELECT E.id, E.name FROM Employee E
SELECT E.id, E.name AS EmployeeName FROM Employee E
SELECT E.id, E.name, I.location, I.phone FROM Employee E, EmployeeInfo I
   WHERE E.id = I.id
SELECT id, name, location, phone FROM Employee, EmployeeInfo WHERE id = empId
SELECT Employee.id, Employee.name, EmployeeInfo.location, EmployeeInfo.phone
   FROM Employee, EmployeeInfo WHERE Employee.id = EmployeeInfo.id
```

The table name returned by the driver for generated columns is an empty string. The following query is an example of a Select statement that returns a result set that contains a generated column (the column named "upper").

```
SELECT E.id, E.name as EmployeeName, {fn UCASE(E.name)} AS upper
   FROM Employee E
```

The driver also can return catalog name information when the ResultSetMetaData.getCatalogName() method is called if the driver can determine that information. For example, for the following statement, the driver returns "test" for the catalog name and "foo" for the table name:

```
SELECT * FROM test.foo
```

The additional processing required to return table name and catalog name information is only performed if the ResultSetMetaData.getTableName() or ResultSetMetaData.getCatalogName() methods are called.

Isolation Levels

When the TransactionMode connection property is set to `ignore`, the driver supports the READ COMMITTED isolation level. When TransactionMode is set to `noTransactions`, the driver supports no isolation levels.

See also

Using Connection Properties on page 50
TransactionMode on page 117

Unicode support

Multilingual JDBC applications can be developed on any operating system using the driver to access both Unicode and non-Unicode enabled databases. Internally, Java applications use UTF-16 Unicode encoding for string data. When fetching data, the driver automatically performs the conversion from the character encoding used by the database to UTF-16. Similarly, when inserting or updating data in the database, the driver automatically converts UTF-16 encoding to the character encoding used by the database.

The JDBC API provides mechanisms for retrieving and storing character data encoded as Unicode (UTF-16) or ASCII. Additionally, the Java String object contains methods for converting UTF-16 encoding of string data to or from many popular character encodings.

Error Handling

SQLExceptions

The driver reports errors to the application by throwing SQLExceptions. Each SQLException contains the following information:

- Description of the probable cause of the error, prefixed by the component that generated the error
- Native error code (if applicable)
- String containing the XOPEN SQLstate

Driver Errors

An error generated by the driver has the format shown in the following example:

```
[DataDirect][SparkSQL JDBC Driver]Timeout expired.
```

You may need to check the last JDBC call your application made and refer to the JDBC specification for the recommended action.

Database Errors

An error generated by the database has the format shown in the following example:

```
[DataDirect][SparkSQL JDBC Driver][SparkSQL]Invalid Object Name.
```

If you need additional information, use the native error code to look up details in your database documentation.

Large Object Support

Although Apache Spark SQL does not define a Clob data type, the driver allows you to retrieve and update long data, specifically LONGVARCHAR data, using JDBC methods designed for Clobs. To do this, the StringDescribeType property must be set to `longvarchar`.

When using these methods to update long data as Clobs, the updates are made to the local copy of the data contained in the Clob object.

Note: The driver does not support retrieving and updating long data using JDBC methods designed for Blobs.

Retrieving and updating long data using JDBC methods designed for Clobs provides some of the same benefits as retrieving and updating Clobs, such as:

- Provides random access to data
- Allows searching for patterns in the data, such as retrieving long data that begins with a specific character string

To provide these benefits normally associated with Clobs, data must be cached. Because data is cached, your application will incur a performance penalty.

See also

Using Connection Properties on page 50
Data Type Handling Properties on page 54
StringDescribeType on page 117

Rowset Support

The driver supports any JSR 114 implementation of the RowSet interface, including:

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

Visit the Java Community Process JSR 114: JDBC Rowset Implementations page for more information about JSR 114.

Timeouts

The LoginTimeout connection property allows you to specify how long the driver waits for a connection to be established before timing out the connection request.

See also

Using Connection Properties on page 50
Timeout Properties on page 56

Views

Apache Spark SQL supports views, but as logical objects with no associated storage. Because Apache Spark SQL does not support materialized views, the driver does not support materialized views.

SQL Escape Sequences

Refer to "SQL escape sequences" in the *Progress DataDirect for JDBC Drivers Reference* for information about SQL escape sequences.

Using Scrollable Cursors

The driver supports only scroll-insensitive, read-only result sets.

Note: When the driver cannot support the requested result set type or concurrency, it automatically downgrades the cursor and generates one or more SQLWarnings with detailed information.

Spark SQL Compatibility with Apache Hive

Spark SQL currently supports most Hive features and additional features are continuously being added. For the latest information, refer to Spark SQL and DataFrame Guide: Compatibility with Apache Hive.

Stored Procedures

Apache Spark SQL has no concept of stored procedures. Therefore, the driver does not support stored procedures or returning parameter metadata for stored procedure arguments.

4

Connection Property Descriptions

You can use connection properties to customize the driver for your environment. This section lists the connection properties supported by the driver and describes each property. You can use these connection properties with either the JDBC Driver Manager or a JDBC data source. For a Driver Manager connection, a property is expressed as a key value pair and takes the form *property*=*value*. For a data source connection, a property is expressed as a JDBC method and takes the form `setproperty(value)`.

Note:

- In a JDBC data source, string values must be enclosed in double quotation marks, for example, `setUser("abc@defcorp.com")`.
- The data type listed for each connection property is the Java data type used for the property value in a JDBC data source.
- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: {*value*}. For example: `User={hello }` or `Password={{hello}}`.

The following table provides a summary of the connection properties supported by the driver and their default values.

Table 19: Connection Properties

Property	Data Source Method	Default
AccountingInfo on page 85	setAccountingInfo	empty string

Property	Data Source Method	Default
ApplicationName on page 86	setApplicationName	empty string
ArrayFetchSize on page 87	setArrayFetchSize	`20000` (cells)
AuthenticationMethod on page 88	setAuthenticationMethod	`userIdPassword`
BinaryDescribeType on page 89	setBinaryDescribeType	`varbinary`
ClientHostName on page 89	setClientHostName	empty string
ClientUser on page 90	setClientUser	empty string
ConnectionRetryCount on page 91	setConnectionRetryCount	`5`
ConnectionRetryDelay on page 92	setConnectionRetryDelay	`1` (second)
ConvertNull on page 92	setConvertNull	`1`
CookieName on page 93	setCookieName	`hive.server2.auth`
CryptoProtocolVersion on page 94	setCryptoProtocolVersion	None
DatabaseName on page 95	setDatabaseName	The default database
EnableCookieAuthentication on page 95	setEnableCookieAuthentication	`true`
EncryptionMethod on page 96	setEncryptionMethod	`noEncryption`
HostNameInCertificate on page 97	setHostNameInCertificate	empty string
HTTPPath on page 98	setHTTPPath	`cliservice`
ImportStatementPool on page 99	setImportStatementPool	empty string
InitializationString on page 100	setInitializationString	None
InsensitiveResultSetBufferSize on page 101	setInsensitiveResultSetBufferSize	`2048`
JavaDoubleToString on page 102	setJavaDoubleToString	`false`
KeyPassword on page 102	setKeyPassword	None
KeyStore on page 103	setKeyStore	None
KeyStorePassword on page 104	setKeyStorePassword	None
LoginTimeout on page 105	setLoginTimeout	`0`
MaxBinarySize on page 105	setMaxBinarySize	`2147483647`

Property	Data Source Method	Default
MaxPooledStatements on page 106	setMaxPooledStatements	None
Password on page 107	setPassword	None
PortNumber on page 108	setPortNumber	`10000`
ProgramID on page 108	setProgramID	empty string
ProxyHost on page 109	setProxyHost	empty string
ProxyPassword on page 110	setProxyPassword	empty string
ProxyPort on page 110	setProxyPort	`0`
ProxyUser on page 111	setProxyUser	empty string
RegisterStatementPoolMonitorMBean on page 111	setRegisterStatementPoolMonitorMBean	`false`
RemoveColumnQualifiers on page 112	setRemoveColumnQualifiers	`false`
ServerName on page 113	setServerName	None
ServicePrincipalName on page 113	setServicePrincipalName	None
SpyAttributes on page 114	setSpyAttributes	None
StringDescribeType on page 117	setStringDescribeType	`varchar`
TransactionMode on page 117	setTransactionMode	`noTransactions`
TransportMode on page 118	setTransportMode	`binary`
TrustStore on page 119	setTrustStore	None
TrustStorePassword on page 120	setTrustStorePassword	None
UseCurrentSchema on page 120	setUseCurrentSchema	`false` (results are not restricted to the tables and views in the current schema)
User on page 121	setUser	None

Property	Data Source Method	Default
UserAgent on page 122	setUserAgent	No default value, which means the driver uses `Progress/8.0 (SparkSQL ODBC driver)` for the value of the User-Agent header.
ValidateServerCertificate on page 122	setValidateServerCertificate	`true`

For details, see the following topics:

- AccountingInfo
- ApplicationName
- ArrayFetchSize
- AuthenticationMethod
- BinaryDescribeType
- ClientHostName
- ClientUser
- ConnectionRetryCount
- ConnectionRetryDelay
- ConvertNull
- CookieName
- CryptoProtocolVersion
- DatabaseName
- EnableCookieAuthentication
- EncryptionMethod
- HostNameInCertificate
- HTTPPath
- ImportStatementPool
- InitializationString
- InsensitiveResultSetBufferSize
- JavaDoubleToString
- KeyPassword
- KeyStore
- KeyStorePassword
- LoginTimeout

- MaxBinarySize
- MaxPooledStatements
- Password
- PortNumber
- ProgramID
- ProxyHost
- ProxyPassword
- ProxyPort
- ProxyUser
- RegisterStatementPoolMonitorMBean
- RemoveColumnQualifiers
- ServerName
- ServicePrincipalName
- SpyAttributes
- StringDescribeType
- TransactionMode
- TransportMode
- TrustStore
- TrustStorePassword
- UseCurrentSchema
- User
- UserAgent
- ValidateServerCertificate

AccountingInfo

Purpose

Defines accounting information. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is the accounting information.

Data Source Method

setAccountingInfo

Default

empty string

Data Type

String

See also

- Using Client Information on page 73
- Client Information Properties on page 57

ApplicationName

Purpose

Specifies the name of the application. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is the name of the application.

Data Source Method

setApplicationName

Default

empty string

Data Type

String

See also

- Using Client Information on page 73
- Client Information Properties on page 57

ArrayFetchSize

Purpose

Specifies the number of fields the driver uses to calculate the maximum number of rows for a fetch. When executing a fetch, the driver divides the ArrayFetchSize value by the number of columns in a particular table to determine the number of rows to retrieve. By determining the fetch size based on the number of fields, the driver can avoid out of memory errors when fetching from tables containing a large number of columns while continuing to provide improved performance when fetching from tables containing a small number of columns.

Valid Values

-x | *x*

where:

-x

is a negative integer.

x

is a positive integer.

Behavior

If set to *-x*, the driver overrides any settings on the statement level and uses the number of fields specified by the absolute value of *-x* to calculate the number of rows to retrieve.

If set to *x*, the driver uses the number of fields specified by the value of *x* to calculate the number of rows to retrieve. However, the driver will not override settings, such as setFetchSize(), on the statement level.

Example

If this property is set to `20000` fields and you are querying a table with 19 columns, the driver divides the number of fields by the number of columns to calculate the number of rows to retrieve. In this example, the driver would retrieve approximately 1053 rows per fetch.

Notes

You can improve performance by increasing the value specified for this option. However, if the number of fields specified exceeds the available buffer memory on the server, an out of memory error will be returned. If you receive this error, decrease the value specified until fetches are successfully executed.

Data Source Method

setArrayFetchSize

Default

`20000` (cells)

Data Type

Int

See also

Performance Considerations on page 61

AuthenticationMethod

Purpose

Determines which authentication method the driver uses when establishing a connection.

Valid Values

`kerberos` | `userIdPassword`

Behavior

If set to `kerberos`, the driver uses Kerberos authentication. The driver ignores any user ID or password that is specified. If you set this value, you also must set the ServicePrincipalName property.

If set to `userIdPassword`, the driver uses user ID/password authentication. If user ID or password is not specified, the driver passes the word "anonymous" for the missing item(s) when submitting requests to the server.

Notes

- The User property provides the user ID. The Password property provides the password.

Data Source Method

setAuthenticationMethod

Default

`userIdPassword`

Data Type

String

See also

- Using Authentication on page 62
- User ID/Password Authentication Properties on page 51

BinaryDescribeType

Purpose

Specifies how columns of the Binary type are described. This property affects ResultSetMetaData calls. getTypeInfo() calls are affected only when the property is set to `longvarbinary`. In this configuration, an additional LONGVARBINARY entry is added to the getTypeInfo results.

Valid Values

`varbinary` | `longvarbinary`

Behavior

If set to `varbinary`, Binary columns are described as VARBINARY.

If set to `longvarbinary`, Binary columns are described as LONGVARBINARY.

Data Source Method

setBinaryDescribeType

Default

`varbinary`

Data Type

String

See also

Performance Considerations

ClientHostName

Purpose

Specifies the host name of the client machine. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

`string`

where:

`string`

is the host name of the client machine.

Data Source Method

setClientHostName

Default

empty string

Data Type

String

See also

- Using Client Information on page 73
- Client Information Properties on page 57

ClientUser

Purpose

Specifies the user ID. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is a valid user ID.

Data Source Method

setClientUser

Default

empty string

Data Type

String

See also

- Using Client Information on page 73
- Client Information Properties on page 57

ConnectionRetryCount

Purpose

Specifies the number of times the driver retries connection attempts to the server until a successful connection is established.

Valid Values

`0` | *x*

where:

x

is a positive integer that represents the number of retries.

Behavior

If set to `0`, the driver does not try to reconnect after the initial unsuccessful attempt.

If set to *x*, the driver retries connection attempts the specified number of times. If a connection is not established during the retry attempts, the driver returns an exception that is generated by the last server to which it tried to connect.

Example

If this property is set to 2, the driver retries the server twice after the initial retry attempt.

Notes

- If an application sets a login timeout value (for example, using DataSource.loginTimeout or DriverManager.loginTimeout) and the login timeout expires, the driver ceases connection attempts.
- The ConnectionRetryDelay property specifies the wait interval, in seconds, to occur between retry attempts.

Data Source Method

setConnectionRetryCount

Default

5

Data Type

int

See also

ConnectionRetryDelay on page 92

ConnectionRetryDelay

Purpose

Specifies the number of seconds the driver waits between connection retry attempts when ConnectionRetryCount is set to a positive integer.

Valid Values

`0` | *x*

where:

x

is a number of seconds.

Behavior

If set to `0`, the driver does not delay between retries.

If set to *x*, the driver waits between connection retry attempts the specified number of seconds.

Example

If ConnectionRetryCount is set to `2` and this property is set to `3`, the driver retries the server twice after the initial retry attempt. The driver waits 3 seconds between retry attempts.

Data Source Method

setConnectionRetryDelay

Default

`1` (second)

Data Type

int

See also

ConnectionRetryCount on page 91

ConvertNull

Purpose

Controls how data conversions are handled for null values.

Valid Values

`0` | `1`

Behavior

If set to `0`, the driver does not perform the data type check if the value of the column is null. This allows null values to be returned even though a conversion between the requested type and the column type is undefined.

If set to `1`, the driver checks the data type being requested against the data type of the table column that stores the data. If a conversion between the requested type and column type is not defined, the driver generates an "unsupported data conversion" exception regardless of whether the column value is NULL.

Data Source Method

setConvertNull

Default

`1`

Data Type

int

See also

Data Type Handling Properties on page 54

CookieName

Purpose

Specifies the name of the cookie used for authenticating HTTP requests when HTTP mode `TransportMode=http` and cookie based authentication are enabled (`EnableCookieAuthentication=true`). When preparing an HTTP request to the server, the driver will not attempt to reauthenticate if this cookie is present.

Valid Values

`string`

where:

`string`

is a valid cookie name.

Data Source Method

setCookieName

Default

`hive.server2.auth`

Data Type

String

Notes

- If HTTP mode (`TransportMode=binary`) or cookie based authentication (`EnableCookieAuthentication=false`) are disabled, this option is ignored.

See also

- TransportMode on page 118
- EnableCookieAuthentication on page 95
- AuthenticationMethod on page 88

CryptoProtocolVersion

Purpose

Specifies a cryptographic protocol or comma-separated list of cryptographic protocols that can be used when TLS/SSL is enabled using the EncryptionMethod connection property.

Valid Values

`cryptographic_protocol` `[[,` *`cryptographic_protocol`* `]...]`

where:

`cryptographic_protocol`

is one of the following cryptographic protocols:

`TLSv1.2` | `TLSv1.1` | `TLSv1` | `SSLv3` | `SSLv2`

Caution: To avoid vulnerabilities associated with SSLv3 and SSLv2, good security practices recommend using TLSv1 or higher.

Example

If your server supports TLSv1.1 and TLSv1.2, you can specify acceptable cryptographic protocols with the following key-value pair:

```
CryptoProtocolVersion=TLSv1.1,TLSv1.2
```

Notes

- When multiple protocols are specified, the driver uses the highest version supported by the server. If none of the specified protocols are supported by the server, the connection fails and the driver returns an error.
- When no value has been specified for CryptoProtocolVersion, the cryptographic protocol used depends on the highest protocol version supported by the server and the highest protocol version supported by the JDK. Refer to the database management system documentation for information on which cryptographic protocols are supported.

Data Source Method

setCryptoProtocolVersion

Default

No default value

Data Type

String

See also

Data encryption on page 70

DatabaseName

Purpose

Specifies the name of the database to which you want to connect.

Valid Values

database_name

where:

database_name

is the name of a valid database. If the driver cannot find the specified database, the connection fails.

Data Source Method

setDatabaseName

Default

The default database

Data Type

String

EnableCookieAuthentication

Purpose

Determines whether the driver attempts to use cookie based authentication for requests to an HTTP endpoint after the initial authentication to the server. Cookie based authentication improves response time by eliminating the need to re-authenticate with the server for each request.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver attempts to use cookie based authentication for requests to an HTTP endpoint after the initial authentication to the server. The cookie used for authentication is specified by the CookieName property. If the name does not match, or authentication fails, the driver attempts to authenticate according to the setting of the AuthenticationMethod property.

If set to `false`, the driver does not use cookie based authentication for HTTP requests after the initial authentication.

Notes

- If HTTP mode is disabled (`TransportMode=binary`), this option is ignored.

Data Source Method

setEnableCookieAuthentication

Default

`true`

Data Type

String

See also

- TransportMode on page 118
- CookieName on page 93
- AuthenticationMethod on page 88

EncryptionMethod

Purpose

Determines whether data is encrypted and decrypted when transmitted over the network between the driver and database server.

Valid Values

`noEncryption` | `SSL` | `requestSSL`

Behavior

If set to `noEncryption`, data is not encrypted or decrypted.

If set to `SSL`, data is encrypted using SSL. If the database server does not support SSL, the connection fails and the driver throws an exception.

If set to `requestSSL`, the login request and data is encrypted using SSL. If the database server does not support SSL, the driver establishes an unencrypted connection.

Notes

When SSL is enabled, the following properties also apply:

- CryptoProtocolVersion
- HostNameInCertificate
- KeyStore (for SSL client authentication)
- KeyStorePassword (for SSL client authentication)
- KeyPassword (for SSL client authentication)
- TrustStore
- TrustStorePassword
- ValidateServerCertificate

Data Source Method

setEncryptionMethod

Default

`noEncryption`

Data Type

String

See also

- Data encryption on page 70
- Performance Considerations on page 61

HostNameInCertificate

Purpose

Specifies a host name for certificate validation when SSL encryption is enabled (`EncryptionMethod=SSL`) and validation is enabled (`ValidateServerCertificate=true`). This property is optional and provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

Valid Values

`host_name` | `#SERVERNAME#`

where:

`host_name`

is a valid host name.

Behavior

If *host_name* is specified, the driver compares the specified host name to the DNSName value of the SubjectAlternativeName in the certificate. If the certificate does not have a SubjectAlternativeName, the driver compares the host name with the Common Name (CN) part of the certificate. If the values do not match, the connection fails and the driver throws an exception.

If #SERVERNAME# is specified, the driver compares the server name that is specified in the connection URL or data source of the connection to the DNSName value of the SubjectAlternativeName in the certificate. If the certificate does not have a SubjectAlternativeName, the driver compares the host name to the CN part of the certificate's Subject name. If the values do not match, the connection fails and the driver throws an exception. If multiple CN parts are present, the driver validates the host name against each CN part. If any one validation succeeds, a connection is established.

Notes

- If SSL encryption or certificate validation is not enabled, this property is ignored.
- If SSL encryption and validation is enabled and this property is unspecified, the driver uses the server name specified in the connection URL or data source of the connection to validate the certificate.

Data Source Method

setHostNameInCertificate

Default

Empty string

Data Type

String

See also

Data encryption on page 70

HTTPPath

Purpose

Specifies the path of the HTTP/HTTPS endpoint used for connections when HTTP mode is enabled (`TransportMode=http`).

Valid Values

string

where:

string

is the path of the URL endpoint. By default, the value specified must be an HTTP end point. To support HTTPS values, enable SSL using the EncryptionMethod property (`EncryptionMethod=SSL`).

Example

If your server was listening to the following URL:

`https://myserver:10001/cliservice/`

Then the following value should be specified for HTTPPath:

`cliservice`

Notes

- This option is ignored when HTTP mode is disabled (`TransportMode=binary`).

Data Source Method

setHTTPPath

Default

`cliservice`

Data Type

String

See Also

- TransportMode on page 118
- EncryptionMethod on page 96

ImportStatementPool

Purpose

Specifies the path and file name of the file to be used to load the contents of the statement pool. When this property is specified, statements are imported into the statement pool from the specified file.

Valid Values

`string`

where:

`string`

is the path and file name of the file to be used to load the contents of the statement pool.

Notes

If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver throws an exception.

Data Source Method

setImportStatementPool

Default

empty string

Data Type

String

See also

- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.
- MaxPooledStatements on page 106
- RegisterStatementPoolMonitorMBean on page 111

InitializationString

Purpose

Specifies one or multiple SQL commands to be executed by the driver after it has established the connection to the database and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver throws an exception indicating which SQL command or commands failed.

Valid Values

command[[;*command*]...]

where:

command

is a SQL command.

Notes

Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified.

Example

```
jdbc:datadirect:sparksql://SparkServer:10000;User=Admin;
Password=secret;DatabaseName=CompanyDB;
InitializationString=(command1;command2)
```

Data Source Method

setInitializationString

Default

None

Data Type

String

InsensitiveResultSetBufferSize

Purpose

Determines the amount of memory that is used by the driver to cache insensitive result set data.

Valid Values

`-1` | `0` | *x*

where:

x

is a positive integer that represents the amount of memory.

Behavior

If set to `-1`, the driver caches insensitive result set data in memory. If the size of the result set exceeds available memory, an OutOfMemoryException is generated. With no need to write result set data to disk, the driver processes the data efficiently.

If set to `0`, the driver caches insensitive result set data in memory, up to a maximum of 2 MB. If the size of the result set data exceeds available memory, then the driver pages the result set data to disk, which can have a negative performance effect. Because result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk.

If set to *x*, the driver caches insensitive result set data in memory and uses this value to set the size (in KB) of the memory buffer for caching insensitive result set data. If the size of the result set data exceeds available memory, then the driver pages the result set data to disk, which can have a negative performance effect. Because the result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk. Specifying a buffer size that is a power of 2 results in efficient memory use.

Data Source Method

setInsensitiveResultSetBufferSize

Default

`2048`

Data Type

int

See also

Performance Considerations

JavaDoubleToString

Purpose

Determines which algorithm the driver uses when converting a double or float value to a string value. By default, the driver uses its own internal conversion algorithm, which improves performance.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver uses the JVM algorithm when converting a double or float value to a string value. If your application cannot accept rounding differences and you are willing to sacrifice performance, set this value to `true` to use the JVM conversion algorithm.

If set to `false`, the driver uses its own internal algorithm when converting a double or float value to a string value. This value improves performance, but slight rounding differences within the allowable error of the double and float data types can occur when compared to the same conversion using the JVM algorithm.

Data Source Method

setJavaDoubleToString

Default

`false`

Data Type

Boolean

See also

Data Type Handling Properties on page 54

KeyPassword

Purpose

Specifies the password that is used to access the individual keys in the keystore file when SSL is enabled (`EncryptionMethod=SSL`) and SSL client authentication is enabled on the database server. This property is useful when individual keys in the keystore file have a different password than the keystore file.

Valid Values

string

where:

string

is a valid password.

Data Source Method

setKeyPassword

Default

None

Data Type

String

See also

Data encryption on page 70

KeyStore

Purpose

Specifies the directory of the keystore file to be used when SSL is enabled (`EncryptionMethod=SSL`) and SSL client authentication is enabled on the database server. The keystore file contains the certificates that the client sends to the server in response to the server's certificate request.

Valid Values

string

where:

string

is a valid directory of a keystore file.

Notes

- This value overrides the directory of the keystore file that is specified by the javax.net.ssl.keyStore Java system property. If this property is not specified, the keystore directory is specified by the javax.net.ssl.keyStore Java system property.
- The keystore and truststore files can be the same file.

Data Source Method

setKeyStorePassword

Default

None

Data Type

String

See also

Data encryption on page 70

KeyStorePassword

Purpose

Specifies the password that is used to access the keystore file when SSL is enabled (`EncryptionMethod=SSL`) and SSL client authentication is enabled on the database server. The keystore file contains the certificates that the client sends to the server in response to the server's certificate request.

Valid Values

`string`

where:

`string`

is a valid password.

Notes

- This value overrides the password of the keystore file that is specified by the javax.net.ssl.keyStorePassword Java system property. If this property is not specified, the keystore password is specified by the javax.net.ssl.keyStorePassword Java system property.
- The keystore and truststore files can be the same file.

Data Source Method

setKeyStorePassword

Default

None

Data Type

String

See also

Data encryption on page 70

LoginTimeout

Purpose

Specifies the amount of time, in seconds, that the driver waits for a connection to be established before timing out the connection request.

Valid Values

`0` | *x*

where:

x

is a positive integer that represents a number of seconds.

Behavior

If set to `0`, the driver does not time out a connection request.

If set to *x*, the driver waits for the specified number of seconds before returning control to the application and throwing a timeout exception.

Data Source Method

setLoginTimeout

Default

`0`

Data Type

int

See also

Timeout Properties on page 56

MaxBinarySize

Purpose

Specifies the maximum length, in bytes, of fields of the Binary data type when described by the driver through result set descriptions and metadata methods. If a field exceeds the specified length, the driver truncates that field.

Valid Values

x

where:

x

is a positive integer that represents the maximum size of fields of the BINARY data type.

Data Source Method

setMaxBinarySize

Default

2147483647

Data Type

Int

MaxPooledStatements

Purpose

Specifies the maximum number of prepared statements to be pooled for each connection and enables the driver's internal prepared statement pooling when set to an integer greater than zero (0). The driver's internal prepared statement pooling provides performance benefits when the driver is not running from within an application server or another application that provides its own statement pooling.

Valid Values

0 | *x*

where:

x

is a positive integer that represents a number of prepared statements to be cached.

Behavior

If set to 0, the driver's internal prepared statement pooling is not enabled.

If set to *x*, the driver's internal prepared statement pooling is enabled and the driver uses the specified value to cache up to that many prepared statements created by an application. If the value set for this property is greater than the number of prepared statements that are used by the application, all prepared statements that are created by the application are cached. Because CallableStatement is a sub-class of PreparedStatement, CallableStatements also are cached.

Notes

When you enable statement pooling, your applications can access the Statement Pool Monitor directly with DataDirect-specific methods. However, you can also enable the Statement Pool Monitor as a JMX MBean. To enable the Statement Pool Monitor as an MBean, statement pooling must be enabled with MaxPooledStatements and the Statement Pool Monitor MBean must be registered using the RegisterStatementPoolMonitorMBean connection property.

Example

If the value of this property is set to `20`, the driver caches the last 20 prepared statements that are created by the application.

Data Source Method

setMaxPooledStatements

Default

`0`

Data Type

Int

See also

- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.
- Performance Considerations on page 61
- RegisterStatementPoolMonitorMBean on page 111
- ImportStatementPool on page 99

Password

Purpose

Specifies a password that is used to connect to the database.

Valid Values

`string`

where:

`string`

is a valid password. The password is case-sensitive.

Data Source Method

setPassword

Default

None

Data Type

String

See also

- User ID/Password Authentication Properties on page 51
- User on page 121

PortNumber

Purpose

Specifies the TCP port of the primary database server that is listening for connections to the database. This property is supported only for data source connections.

Valid Values

port

where:

port

is the port number.

Data Source Method

setPortNumber

Default

10000

Data Type

int

See also

Required Properties on page 50

ProgramID

Purpose

Specifies the driver and version information on the client. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is a value that identifies the product and version of the driver on the client.

Example

DDJ04200

Default

empty string

Data Type

String

See also

- Using Client Information on page 73
- Client Information Properties on page 57

ProxyHost

Description

Identifies a proxy server to use for the first connection.

Valid Values

server_name | *IP_address*

where:

server_name

is the name of the proxy server, which may be qualified with the domain name.

IP_address

is an IP address, specified in either IPv4 or IPv6 format, or a combination of the two.

Data Source Method

setProxyHost

Default

empty string

See also

- Proxy server support on page 72

ProxyPassword

Purpose

Specifies the password needed to connect to a proxy server for the first connection.

Valid Values

password

where:

password

is a valid password for that server. Contact your system administrator to obtain a valid password.

Data Source Method

`setProxyPassword`

Default

empty string

See also

Proxy server support on page 72

ProxyPort

Purpose

Specifies the port number where the proxy server is listening for HTTP or HTTPS requests for the first connection.

Valid Values

port

where:

port

is the port number on which the proxy server is listening. Contact your system administrator to obtain the correct port.

Data Source Method

`setProxyPort`

Default

0

See also

Proxy server support on page 72

ProxyUser

Purpose

Specifies the specifies the user name needed to connect to a proxy server for the first connection.

Valid Values

user_name

where:

user_name

is a valid user ID for the proxy server.

Data Source Method

setProxyUser

Default

empty string

See also

Proxy server support on page 72

RegisterStatementPoolMonitorMBean

Purpose

Registers the Statement Pool Monitor as a JMX MBean when statement pooling has been enabled with MaxPooledStatements. This allows you to manage statement pooling with standard JMX API calls and to use JMX-compliant tools, such as JConsole.

Valid Values

true | false

Behavior

If set to true, the driver registers an MBean for the statement pool monitor for each statement pool. This gives applications access to the Statement Pool Monitor through JMX when statement pooling is enabled.

If set to `false`, the driver does not register an MBean for the Statement Pool Monitor for any statement pool.

Notes

Registering the MBean exports a reference to the Statement Pool Monitor. The exported reference can prevent garbage collection on connections if the connections are not properly closed. When garbage collection does not take place on these connections, out of memory errors can occur.

Default

`false`

Data Type

Boolean

See also

- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.
- MaxPooledStatements on page 106
- ImportStatementPool on page 99

RemoveColumnQualifiers

Purpose

Specifies whether the driver removes 3-part column qualifiers and replaces them with alias.column qualifiers.

Valid Values

`true` | `false`

Behavior

If set to `true` (enabled), the driver removes 3-part column qualifiers and replaces them with alias.column qualifiers.

If set to `false`, the driver does not modify the request.

Default

`false` (Disabled)

Data Type

boolean

See also

Performance Considerations

ServerName

Purpose

Specifies either the IP address or the server name (if your network supports named servers) of the primary database server. This property is supported only for data source connections.

Valid Values

string

where:

string

is a valid IP address or server name.

Example

`122.23.15.12` or `MySparkServer`

Default

None

Data Type

String

See also

- Required Properties on page 50
- IP Addresses on page 75

ServicePrincipalName

Purpose

Specifies the service principal name to be used for Kerberos authentication.

Valid Values

ServicePrincipalName

where:

ServicePrincipalName

is the three-part service principal name registered with the key distribution center (KDC).

Note: Your service principal name is the value of the `hive.server2.authentication.kerberos.principal` property in the `hive-site.xml` file.

You must specify the service principal name using the following format:

`Service_Name`/*`Fully_Qualified_Domain_Name`*@*`REALM.COM`*

where:

`Service_Name`

is the name of the service hosting the instance. For example, `yourservicename`.

Depending on the distribution you are using, the name of the service is defined either automatically by the server or manually by the user who created the service. For instance, CDH distributions automatically generate a service name, while Apache Hadoop distributions require that the service name be manually defined by the user. Refer to your distribution's documentation for additional information.

`Fully_Qualified_Domain_Name`

is the fully qualified domain name of the host machine. For example, `yourserver.example.com`.

`REALM.COM`

is the domain name of the host machine. This part of the value must be specified in upper-case characters. For example, `EXAMPLE.COM`.

Example

The following is an example of a valid service principal name:

```
yourservicename/yourserver.example.com@EXAMPLE.COM
```

Notes

- If unspecified, the value of the ServerName property is used as the service principal name.
- If AuthenticationMethod is set to `userIdPassword` , the value of the ServicePrincipalName property is ignored.

Default

None

See also

Using Authentication on page 62

SpyAttributes

Purpose

Enables DataDirect Spy to log detailed information about calls issued by the driver on behalf of the application. DataDirect Spy is not enabled by default.

Valid Values

(*spy_attribute*[;*spy_attribute*]...)

where:

spy_attribute

is any valid DataDirect Spy attribute.

Behavior

Attribute	Description
linelimit=*numberofchars*	Sets the maximum number of characters that DataDirect Spy logs on a single line. The default is 0 (no maximum limit).
log=(file)*filename*	Directs logging to the file specified by *filename*. For Windows, if coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: log=(file)C:\\temp\\spy.log;logIS=yes;logTName=yes.
log=(filePrefix)*file_prefix*	Directs logging to a file prefixed by *file_prefix*. The log file is named *file_prefixX*.log where: *X* is an integer that increments by 1 for each connection on which the prefix is specified. For example, if the attribute log=(filePrefix)C:\\temp\\spy_ is specified on multiple connections, the following logs are created: C:\temp\spy_1.log C:\temp\spy_2.log C:\temp\spy_3.log ... If coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: log=(filePrefix)C:\\temp\\spy_;logIS=yes;logTName=yes.
log=System.out	Directs logging to the Java output standard, System.out.

Attribute	Description
logIS={yes | no | nosingleread}	Specifies whether DataDirect Spy logs activity on InputStream and Reader objects. When logIS=nosingleread, logging on InputStream and Reader objects is active; however, logging of the single-byte read InputStream.read or single-character Reader.read is suppressed to prevent generating large log files that contain single-byte or single character read messages. The default is no.
logLobs={yes | no}	Specifies whether DataDirect Spy logs activity on BLOB and CLOB objects.
logTName={yes | no}	Specifies whether DataDirect Spy logs the name of the current thread. The default is no.
timestamp={yes | no}	Specifies whether a timestamp is included on each line of the DataDirect Spy log. The default is yes.

Notes

- If coding a path on Windows to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: log=(file)C:\\temp\\spy.log.
- If a log file name does not include the .log extension, the driver automatically appends it. For example, a file named spy.jsp is renamed to spy.jsp.log by the driver.

Example

The following value instructs the driver to log all JDBC activity to a file using a maximum of 80 characters for each line.

```
(log=(file)/tmp/spy.log;linelimit=80)
```

Default

None

See also

Refer to "Tracking JDBC Calls with DataDirect Spy" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using DataDirect Spy.

StringDescribeType

Purpose

Specifies whether String columns are described as VARCHAR or LONGVARCHAR. This property affects resultSetMetaData() calls; it does not affect getTypeInfo() calls.

Valid Values

`varchar` | `longvarchar`

Behavior

If set to `varchar`, String columns are described as VARCHAR.

If set to `longvarchar`, String columns are described as LONGVARCHAR.

Notes

- To obtain data from String columns with the getClob() method, the StringDescribeType connection property must be set to `longvarchar`. Otherwise, calling getClob() results in an "unsupported data conversion" exception.
- StringDescribeType affects all columns reported as String, even columns that were originally cast as Varchar. This is important to note because the Spark Thrift server, when returning result metadata for Varchar columns, reports column type as (12) STRING and precision as 2147483647. For the latest information on Spark SQL support for Varchar, refer to the Apache JIRA SPARK-5918 issue Web page.

Default

`varchar`

Data Type

String

See also

- Performance Considerations on page 61
- Data Type Handling Properties on page 54

TransactionMode

Purpose

Specifies how the driver handles manual transactions.

Valid Values

`ignore` | `noTransactions`

Behavior

If set to `ignore`, the data source does not support transactions and the driver always operates in auto-commit mode. Calls to set the driver to manual commit mode and to commit transactions are ignored. Calls to rollback a transaction cause the driver to throw an exception indicating that no transaction is started. Metadata indicates that the driver supports transactions and the READ UNCOMMITTED transaction isolation level.

If set to `noTransactions`, the data source and the driver do not support transactions. Metadata indicates that the driver does not support transactions.

Notes

Spark SQL does not support transactions, and by default, the driver reports that transactions are not supported. However, some applications will not operate with a driver that reports transactions are not supported. The TransactionMode connection property allows you to configure the driver to report that it supports transactions. In this mode, the driver ignores requests to enter manual commit mode, start a transaction, or commit a transaction. Requests to rollback a transaction return an error regardless of the transaction mode specified.

Default

`noTransactions`

Data Type

String

TransportMode

Purpose

Specifies whether binary (TCP) mode or HTTP mode is used to access Apache Spark SQL data sources.

Valid Values

`binary` | `http`

Behavior

If set to `binary`, Thrift RPC requests are sent to directly to data sources using a binary connection (TCP mode).

If set to `http`, Thrift RPC requests are sent using HTTP transport (HTTP mode). HTTP mode is typically used when connecting to a proxy server, such as a gateway, for improved security, or a load balancer.

Notes

- The setting of this property corresponds to that of the `hive.server2.transport.mode` property in your `hive-site.xml` file.
- When `TransportMode=http`, the HTTP/HTTPS end point for the Hive server must be specified using the httpPath property.
- To configure the driver to use HTTPS end points, set `TransportMode=http` and `EncryptionMethod=SSL`.
- Apache Spark SQL currently supports using only one protocol mode per server at a time.

Data Source Method

setTransportMode

Default

`binary`

Data Type

String

See also

- EncryptionMethod on page 96
- HTTPPath on page 98

TrustStore

Purpose

Specifies the directory of the truststore file to be used when SSL is enabled (`EncryptionMethod=SSL`) and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.

Valid Values

`string`

where:

`string`

is the directory of the truststore file.

Notes

- This value overrides the directory of the truststore file that is specified by the javax.net.ssl.trustStore Java system property. If this property is not specified, the truststore directory is specified by the javax.net.ssl.trustStore Java system property.
- This property is ignored if ValidateServerCertificate=`false`.

Default

None

Data Type

String

See also

Data encryption on page 70

TrustStorePassword

Purpose

Specifies the password that is used to access the truststore file when SSL is enabled (`EncryptionMethod=SSL`) and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.

Valid Values

`string`

where:

`string`

is a valid password for the truststore file.

Notes

- This value overrides the password of the truststore file that is specified by the javax.net.ssl.trustStorePassword Java system property. If this property is not specified, the truststore password is specified by the javax.net.ssl.trustStorePassword Java system property.
- This property is ignored if ValidateServerCertificate=`false`.

Default

None

Data Type

String

See also

Data encryption on page 70

UseCurrentSchema

Purpose

Specifies whether results are restricted to the tables and views in the current schema if a call is made without specifying a schema or if the schema is specified as the wildcard character %. Restricting results to the tables and views in the current schema improves performance of calls that do not specify a schema.

Valid Values

`true` | `false`

Behavior

If set to `true`, the results that are returned from getTables() and getColumns() methods are restricted to the tables and views in the current schema.

If set to `false`, the results that are returned from getTables() and getColumns() methods are not restricted.

Default

`false`

Data Type

Boolean

See also

Performance Considerations on page 61

User

Purpose

Specifies the user name that is used to connect to the database.

Valid Values

`string`

where:

`string`

is a valid user name. The user name is case-insensitive.

Data Source Method

setUser

Default

None

Data Type

String

See also

- User ID/Password Authentication Properties on page 51
- Password on page 107

UserAgent

Purpose

Specifies the string value of the User-Agent header to be used in HTTP requests. This property provides a method to override the default value of the User-Agent header when required by a service.

Valid Values

string

where:

string

is the string value of the User-Agent header.

Data Source Method

setUserAgent

Default

No default value, which means the driver uses `Progress/8.0 (SparkSQL ODBC driver)` for the value of the User-Agent header.

Data Type

String

See also

- HTTP Mode on page 48

ValidateServerCertificate

Purpose

Determines whether the driver validates the certificate that is sent by the database server when SSL encryption is enabled (`EncryptionMethod=SSL`). When using SSL server authentication, any certificate that is sent by the server must be issued by a trusted Certificate Authority (CA).

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver validates the certificate that is sent by the database server. Any certificate from the server must be issued by a trusted CA in the truststore file. If the HostNameInCertificate property is specified, the driver also validates the certificate using a host name. The HostNameInCertificate property is optional and provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

If set to `false`, the driver does not validate the certificate that is sent by the database server. The driver ignores any truststore information that is specified by the TrustStore and TrustStorePassword properties or Java system properties.

Notes

- Truststore information is specified using the TrustStore and TrustStorePassword properties or by using Java system properties.
- Allowing the driver to trust any certificate that is returned from the server even if the issuer is not a trusted CA is useful in test environments because it eliminates the need to specify truststore information on each client in the test environment.

Default

`true`

Data Type

boolean

See also

Data encryption on page 70

5

Supported SQL Functionality

The driver supports Spark SQL and the core SQL grammar (primarily SQL-92). The following topics describe how the driver handles SQL queries in many circumstances.

Note: Spark SQL uses a subset of SQL and HiveQL. While it provides much of the functionality of SQL, the Spark SQL subset of SQL and HiveQL has some differences and limitations. For the latest information, refer to the Spark SQL and DataFrame Guide.

For details, see the following topics:

- Data Definition Language
- Column Name Qualification
- From Clause
- Group By Clause
- Having Clause
- Order By Clause
- For Update Clause
- Set Operators
- Subqueries
- SQL Expressions

Data Definition Language

The driver supports a broad set of DDL, including (but not limited to) the following:

- CREATE Database and DROP Database
- CREATE Table and DROP Table
- ALTER Table and ALTER Partition statements
- CREATE View and DROP View
- CREATE Function and DROP Function

Refer to the Apache Spark SQL Programming Guide for information about using Spark SQL.

Column Name Qualification

A column can only be qualified with a single name. Furthermore, a table can be qualified with a database (JDBC schema) name in the FROM clause, and in some cases, must also be aliased. Aliasing may not be necessary if the database qualifier is not the current database.

The driver can work around these limitations using the Remove Column Qualifiers connection option.

- If set to *1*, the driver removes three-part column qualifiers and replaces them with alias.column qualifiers.
- If set to *0*, the driver does not do anything with the request.

Suppose you have the following ANSI SQL query:

```
SELECT schema.table1.col1,schema.table2.col2 FROM schema.table1,schema.table2
WHERE schema.table1.col3=schema.table2.col3
```

If the Remove Column Qualifiers connection option is enabled, the driver replaces the three-part column qualifiers:

```
SELECT  table1.col1,
table2.col2 FROM schema.table1 table1 JOIN  schema.table2 table2
WHERE  table1.col3 =  table2.col3
```

From Clause

LEFT, RIGHT, and FULL OUTER JOINs are supported, as are LEFT SEMI JOINs and CROSS JOINs using the equal comparison operator, as shown in the following examples:

```
SELECT a.* FROM a JOIN b ON (a.id = b.id AND a.department = b.department)

SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON
(c.key = b.key2)

SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key=b.key)
  WHERE a.ds='2009-07-07' AND b.ds='2009-07-07'
```

However, the following syntax fails because of the use of non-equal comparison operators.

```
SELECT a.* FROM a JOIN b ON (a.id <> b.id)
```

Spark SQL does not support join syntax in the form of a comma-separated list of tables. The driver, however, overcomes this limitation by translating the SQL into Spark SQL, as shown in the following examples.

ANSI SQL-92 Query	Spark SQL Translation
`SELECT * FROM t1, t2 WHERE a = b`	`SELECT * FROM t1 t1 JOIN t2 t2 WHERE a = b`
`SELECT * FROM t1 y, t2 x WHERE a = b`	`SELECT * FROM t1 y JOIN t2 x WHERE a = b`
`SELECT * FROM t2, (SELECT * FROM t1) x`	`SELECT * FROM t2 t2 JOIN (SELECT * FROM t1 t1) x`

Group By Clause

The Group By clause is supported, with the following Entry SQL level restrictions:

- The COLLATE clause is not supported.
- SELECT DISTINCT is not supported.
- The grouping column reference cannot be an alias. The following queries fail because `fc` is an alias for the `intcol` column:

  ```
  SELECT intcol AS fc, COUNT (*) FROM p_gtable GROUP BY fc
  ```

  ```
  SELECT f(col) as fc, COUNT (*) FROM table_name GROUP BY fc
  ```

Having Clause

The Having Clause is supported, with the following Entry SQL level restriction: a GROUP BY clause is required.

Order By Clause

The Order By clause is supported, with the following Entry SQL level restrictions:

- An integer sort key is not allowed.
- The COLLATE clause is not supported.

For Update Clause

Not supported in this release. If present, the driver strips the For Update clause from the query.

Set Operators

Supported, with the following Entry SQL level restrictions:

- UNION is not supported.

  Therefore, the following query fails:

  ```
  SELECT * FROM t1 UNION SELECT * FROM t2
  ```

- UNION ALL is supported.

  Therefore, the following query works:

  ```
  SELECT * FROM t1 UNION ALL SELECT * FROM t2
  ```

In addition, INTERSECT and EXCEPT are not supported.

Subqueries

A query is an operation that retrieves data from one or more tables or views. In this reference, a top-level query is called a Select statement, and a query nested within a Select statement is called a subquery.

Subqueries are supported, with the following Entry SQL level restriction: subqueries can only exist in the FROM and WHERE clauses, that is, in a derived table. In the following example, the second Select statement is a subquery:

```
SELECT * FROM (SELECT * FROM t1 UNION ALL SELECT * FROM t2) sq
```

Although Apache Spark SQL currently does not support IN or EXISTS subqueries, you can efficiently implement the semantics by rewriting queries to use LEFT SEMI JOIN.

SQL Expressions

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value. You can use expressions in the Where and Having clauses of Select statements.

Expressions enable you to use mathematical operations as well as character string manipulation operators to form complex queries.

Valid expression elements are described in this section.

Constants

Apache Spark SQL uses binary literals for internal functions. Although the driver supports binary literals, no useful information is returned.

Numeric Operators

You can use a numeric operator in an expression to negate, add, subtract, multiply, and divide numeric values. The result of this operation is also a numeric value. The + and - operators are also supported in date/time fields to allow date arithmetic.

The following table lists the supported arithmetic operators.

Table 20: Numeric Operators

Entry SQL Level Operator	Spark SQL Operator
*	Supported
+	Supported
-	Supported
/	Supported
^ (XOR)	N/A
% (Mod)	N/A
& (bitwise AND)	N/A

Character Operator

The concatenation operator (||) is not supported. However, the CONCAT function is supported by Apache Spark SQL. For example, the following statement is supported:

```
SELECT CONCAT('Name is', '(ename FROM emp)')
```

Relational Operators

Relational operators compare one expression to another. The following table lists the supported relational operators.

Table 21: Relational Operators Supported with Spark SQL

Entry SQL Level Operator	Support in Spark SQL
<>	Supported
<	Supported
<=	Supported
=	Supported
<=>	Supported

Entry SQL Level Operator	Support in Spark SQL
>	Supported
>=	Supported
IS [NOT] NULL	Supported
[NOT] BETWEEN x AND y	Supported
[NOT] IN	Supported
EXISTS	Supported
[NOT] LIKE	Supported, except that no collate clause is allowed
RLIKE	Supported
REGEXP	Supported

Logical Operators

A logical operator combines the results of two component conditions to produce a single result or to invert the result of a single condition. The following table lists the supported logical operators.

Table 22: Logical Operators

Operator	Support in Spark SQL
NOT !	Supported
AND &&	Supported
OR ||	Supported

Functions

The following tables show how SQL-92 functions are supported in Apache Spark SQL.

Table 23: Set Functions Supported

Set Function	Support in Spark SQL
Count	Supported
AVG	Supported
MIN	Supported
MAX	Supported

Set Function	Support in Spark SQL
SUM	Supported
DISTINCT	Supported
ALL	Supported

Table 24: Numeric Functions Supported

Numeric Function	Support in Spark SQL
CHAR_LENGTH CHARACTER_LENGTH	Not supported. Use LENGTH(string) instead.
Position...In	Not supported
BIT_LENGTH(s)	Not supported
OCTET_LENGTH(str)	Not supported
EXTRACT...FROM	Not supported

Table 25: String Functions Supported

String Function	Support in Spark SQL
Substring	Supported
Convert … using	Not supported
TRIM	Supported.
Leading	Not supported. Use LTRIM.
Trailing	Not supported. Use RTRIM.
Both	Not supported (default behavior of TRIM)

Table 26: Date/Time Functions Supported

Date/Time Function	Support in Spark SQL
CURRENT_DATE( )	Not supported[10]
CURRENT_TIME( )	Not supported[10]
CURRENT_TIMESTAMP	Not supported. Use UNIX_TIMESTAMP()[10].

[10] Supported by JDBC Escapes.

Table 27: System Functions Supported

System Function	Support in Spark SQL
CASE . . . END	Supported.
COALESCE	Supported.
NULLIF	Not supported.[10]
CAST	Supported.

Note:

Refer to "SQL escape sequences" in the *Progress DataDirect for JDBC Drivers Reference* for information about SQL escape sequences.