Progress DataDirect

Progress DataDirect Connect XE for JDBC for Impala

User's Guide

Release 5.1.4

Copyright

Table of Contents

1

Welcome to the Progress DataDirect Connect XE for JDBC for Impala Driver

The Progress® DataDirect Connect XE® *for* JDBC™ for Impala™ driver supports Cloudera Impala, versions 1.3 and higher.

The Cloudera Impala server is compatible with data stored in various file formats. In addition, Cloudera Impala can work with data stored in other systems through the use of storage handlers. The driver is designed to work seamlessly with the file formats and storage handlers used by the server.

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-cloudera-impala.

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 5.1.4 Release

Driver Enhancements

- The driver has been enhanced to support Kerberos authentication. See Configuring the driver for Kerberos authentication on page 54 for details.
- The driver has been enhanced to support SSL data encryption. See Data Encryption on page 59 and Data Encryption Properties on page 46 for details.
- 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`. See RegisterStatementPoolMonitorMBean on page 92 for details.

  Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for an overview of statement pooling.

Highlights of the 5.1.4 Release

Driver Enhancements

- The driver has been enhanced to support the Char and Vachar data type with Cloudera Impala 2.1 and higher. See Data Types on page 13 for details.
- Support for result set holdability has been added to the driver.

Release Features

- The driver supports standard SQL query language for read-write access to Cloudera Impala data stores. See Supported SQL Functionality on page 103 for details.
- The driver supports JDBC core functions. For details, refer to "JDBC support" in the *Progress DataDirect for JDBC Drivers Reference*.
- Supports Cloudera Impala data types. See Data Types on page 13 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.

Note: To use the driver on a Java Platform with standard Security Manager enabled, certain permissions must be set in the security policy file of the Java Platform. See "Required permissions for Java SE with the standard Security Manager enabled" for details.

See also

Required permissions for Java SE with the standard Security Manager enabled on page 34

Data Source and Driver Classes

The driver class for the Impala driver is:

com.ddtek.jdbc.impala.ImpalaDriver

The driver provides the following data source class that supports the functionality for all JDBC specifications and Java SE 8 or higher:

com.ddtek.jdbcx.impala.ImpalaDataSource

Note: For compatibility purposes, the driver also provides a second data sources class, com.ddtek.jdbcx.impala.ImpalaDataSource40. This data source class functions identically to com.ddtek.jdbcx.impala.ImpalaDataSource; however, it is named after a data source class provided in earlier releases, allowing applications that rely on the older version to use the driver without changes.

See "Connecting Using Data Sources" for information about connecting using data sources.

See also

Connecting Using Data Sources on page 26

Connection URL

After registering the driver, the required connection information needs to be passed in the form of a connection URL. The connection URL takes the form:

```
jdbc:datadirect:impala://servername:[port][;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 21050.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon. For more information on connection properties, see "Using Connection Properties."

The *install_dir*/samples directory, where *install_dir* is your product installation directory, contains a sample application that illustrates the steps of connecting to a Cloudera Impala server.

This example shows how to establish a connection to a Cloudera Impala server with user ID/password authentication:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:impala://Server3:21050;
  DatabaseName=Test;User=admin;Password=adminpass");
```

See also

Using Connection Properties on page 44

Connection Properties

The driver includes over 20 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 "Using Connection Properties" and "Connection Property Descriptions" for more information.

See also

Using Connection Properties on page 44
Connection Property Descriptions on page 69

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:

```
5.1.4.000002(F000001.U000002)
      |____| |_____| |_____|
      Driver  Frame    Utl
```

Data Types

The following table shows how the Impala data types are mapped to the standard JDBC data types.

Table 1: Impala Data Types

Impala	JDBC
Bigint	BIGINT
Boolean	BOOLEAN
Char[1]	CHAR
Decimal[2]	DECIMAL
Double	DOUBLE
Float	REAL
Int	INTEGER
Smallint	SMALLINT
String[3]	VARCHAR *or* LONGVARCHAR[4]
Timestamp	TIMESTAMP
Tinyint	TINYINT
Varchar[5]	VARCHAR

[1] Supported for Cloudera Impala 2.1 and higher.
[2] Supported for Cloudera Impala 1.4 and higher.
[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.
[3] Maximum of 2 GB
[5] Supported for Cloudera Impala 2.1 and higher.

getTypeInfo

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

Table 2: getTypeInfo

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

TYPE_NAME = char[6] AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 1 (CHAR) FIXED_PREC_SCALE = true LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = char MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 255 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = decimal[7] 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

[6] Supported for Cloudera Impala 2.1 and higher.
[7] Supported for Cloudera Impala 1.4 and higher.

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[8] AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) *or* -1 (LONGVARCHAR)[9] 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
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

[8] Maximum of 2 GB

[9] 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.

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[10] 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 = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

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.

[10] Supported for Cloudera Impala 2.1 and higher.

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

The driver class for the Impala driver is:

com.ddtek.jdbc.impala.ImpalaDriver

The driver provides the following data source class that supports the functionality for all JDBC specifications and Java SE 8 or higher:

com.ddtek.jdbcx.impala.ImpalaDataSource

Note: For compatibility purposes, the driver also provides a second data sources class, com.ddtek.jdbcx.impala.ImpalaDataSource40. This data source class functions identically to com.ddtek.jdbcx.impala.ImpalaDataSource; however, it is named after a data source class provided in earlier releases, allowing applications that rely on the older version to use the driver without changes.

See "Connecting Using Data Sources" for information about connecting using data sources.

See also

Connecting Using Data Sources on page 26

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 impala.jar file as shown, where *`install_dir`* is the path to your product installation directory:

```
install_dir/lib/51/impala.jar
```

Windows Example

```
CLASSPATH=.;C:\Program Files\Progress\DataDirect\lib\51\impala.jar
```

UNIX Example

```
CLASSPATH=.:/opt/Progress/DataDirect/lib/51/impala.jar
```

Connecting Using the Driver Manager

One way to connect to an Impala 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:impala://Server3:21050;
  DatabaseName=Test;User=admin;Password=adminpass");
```

Passing the Connection URL

After registering the driver, the required connection information needs to be passed in the form of a connection URL. The connection URL takes the form:

```
jdbc:datadirect:impala://servername:[port][;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 21050.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon. For more information on connection properties, see "Using Connection Properties."

The *install_dir*/samples directory, where *install_dir* is your product installation directory, contains a sample application that illustrates the steps of connecting to a Cloudera Impala server.

This example shows how to establish a connection to a Cloudera Impala server with user ID/password authentication:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:impala://Server3:21050;
  DatabaseName=Test;User=admin;Password=adminpass");
```

See also

Using Connection Properties on page 44

Testing the Connection

You can use DataDirect Test™ to verify your connection. The screen shots in this section were taken on a Windows system.

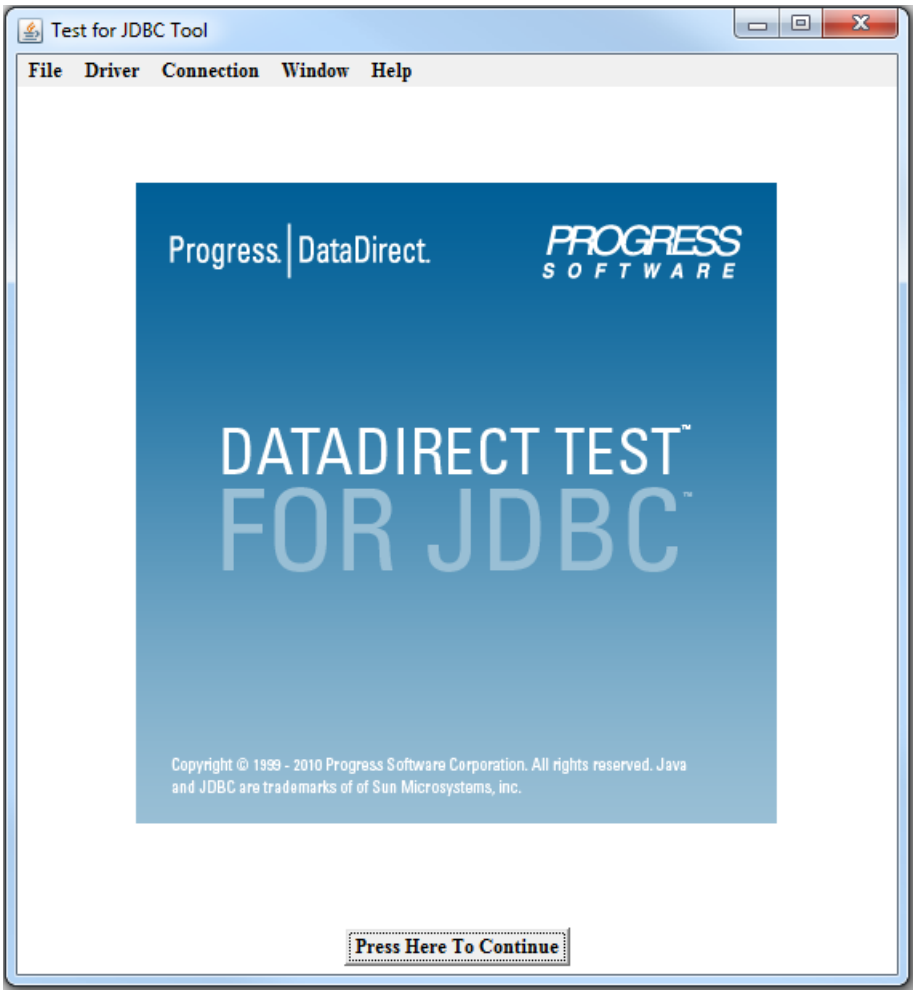
To test the connection from the driver to your data source, follow these steps:

1. Navigate to the installation directory. The default location is:
   - Windows systems: Program Files\Progress\DataDirect\JDBC_51\testforjdbc
   - UNIX and Linux systems: /opt/Progress/DataDirect/JDBC_51/testforjdbc

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

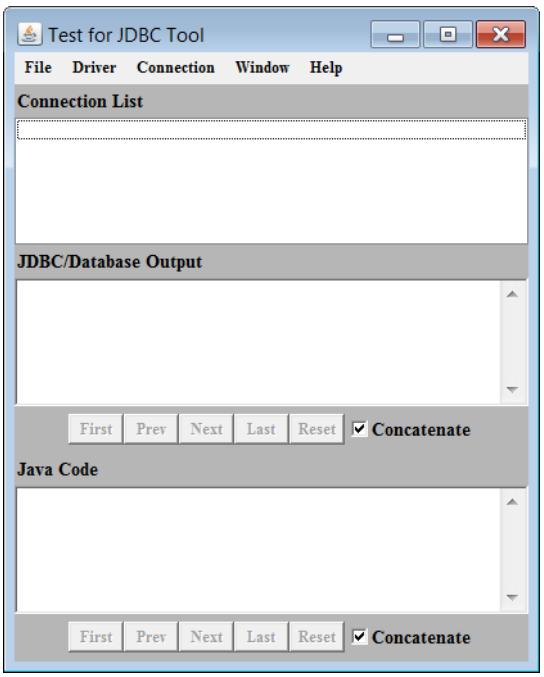
2. From the testforjdbc folder, run the platform-specific tool:
   - testforjdbc.bat (on Windows systems)
   - testforjdbc.sh (on UNIX and Linux systems)

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



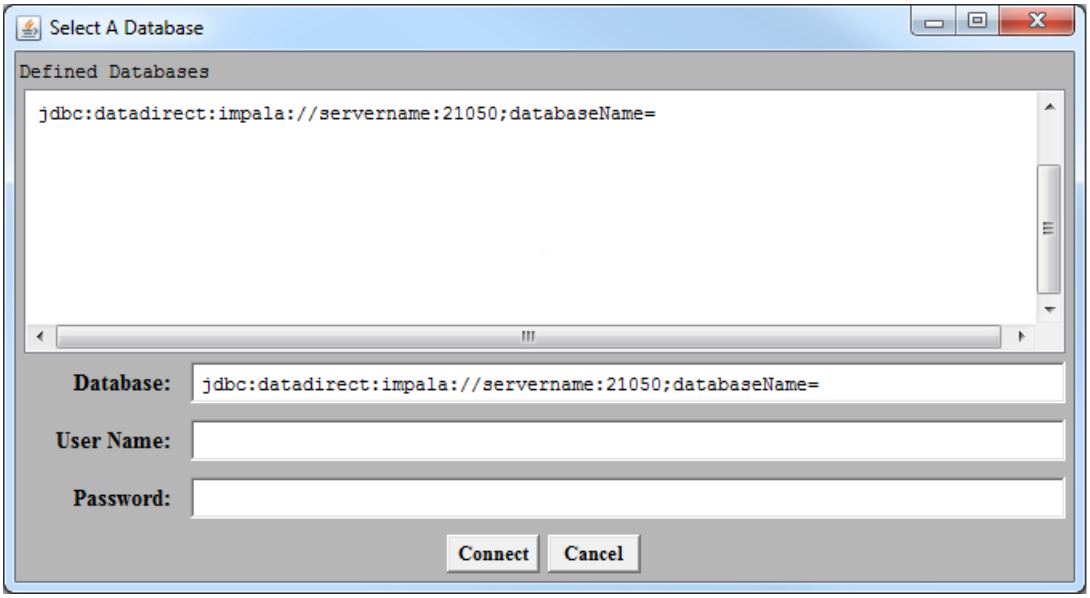
3. Click **Press Here to Continue**.

   The main dialog appears:



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

   The **Select A Database** dialog appears:



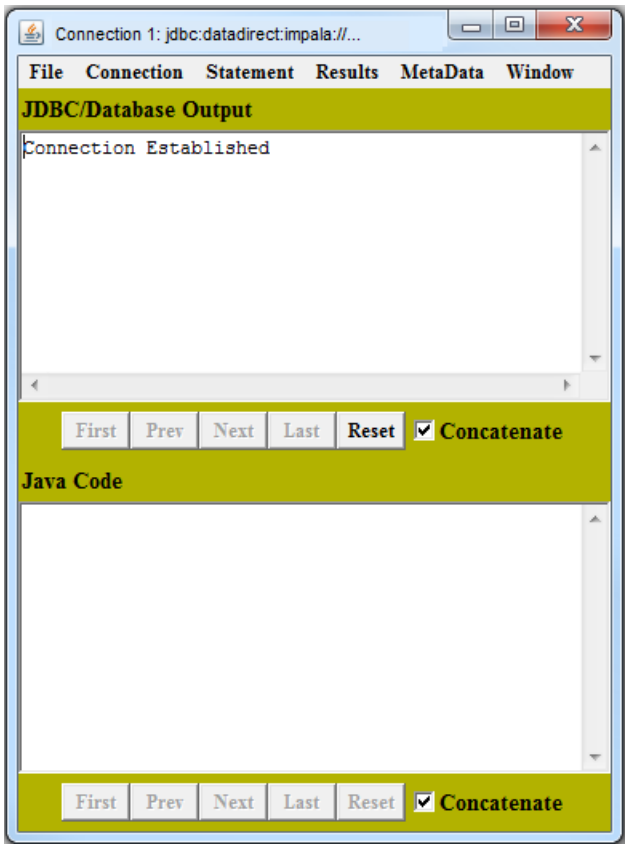
5. Select the appropriate database template from the Defined Databases field.
6. In the Database field, specify the correct ServerName and PortNumber for your Cloudera Impala data source.

   For example:

   ```
   jdbc:datadirect:impala://Server3:21050;databaseName=Test
   ```

7. If required, enter your user name and password in the fields provided.
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.)



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)

See "Data Source and Driver Classes" for data source class information.

See also

Data Source and Driver Classes on page 11

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.

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.

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:

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.impala.ImpalaDataSource;
```

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 connection:

```
ImpalaDataSource mds = new ImpalaDataSource();
mds.setDescription("My Impala Server");
mds.setServerName("MyServer");
mds.setPortNumber(21050);
mds.setDatabaseName("myDB");
```

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 ImpalaDataSource)
{
ImpalaDataSource jmds = (ImpalaDataSource) 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 DataSource Connection

You can use DataDirect Test™ to verify your connection. The screen shots in this section were taken on a Windows system.

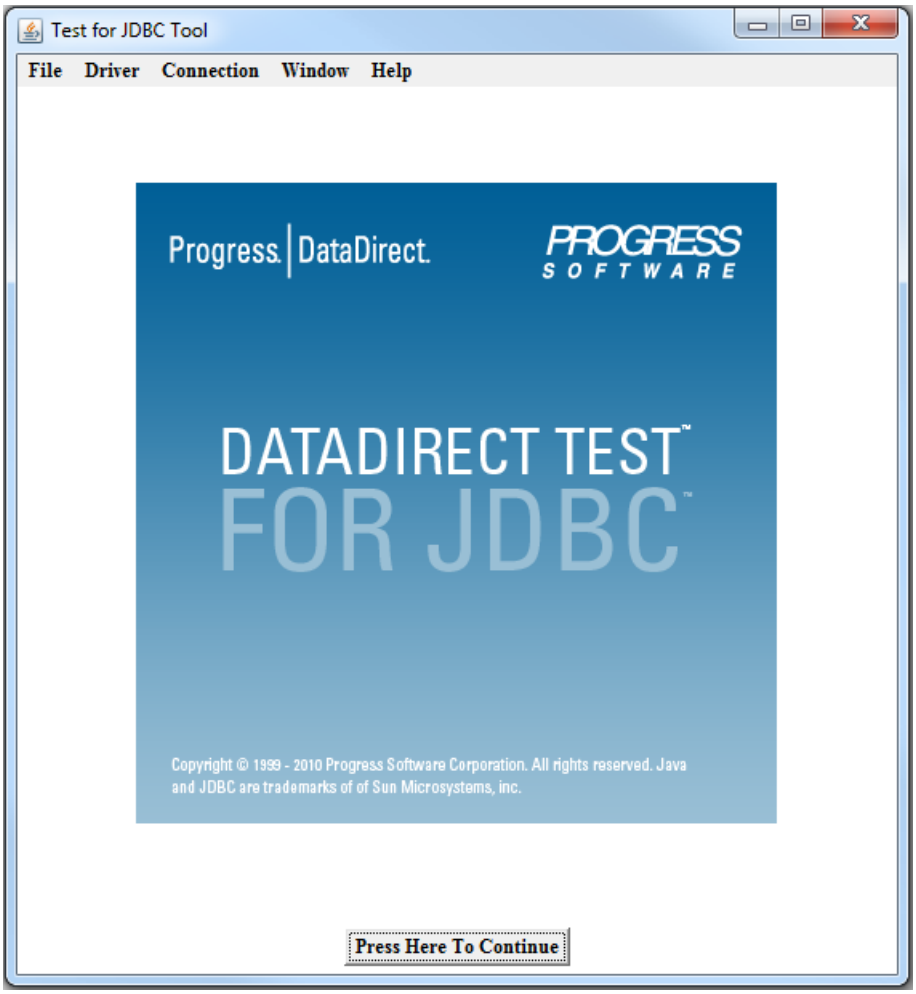
To test the connection from the driver to your data source, follow these steps:

1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC_51\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC_51/testforjdbc`

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

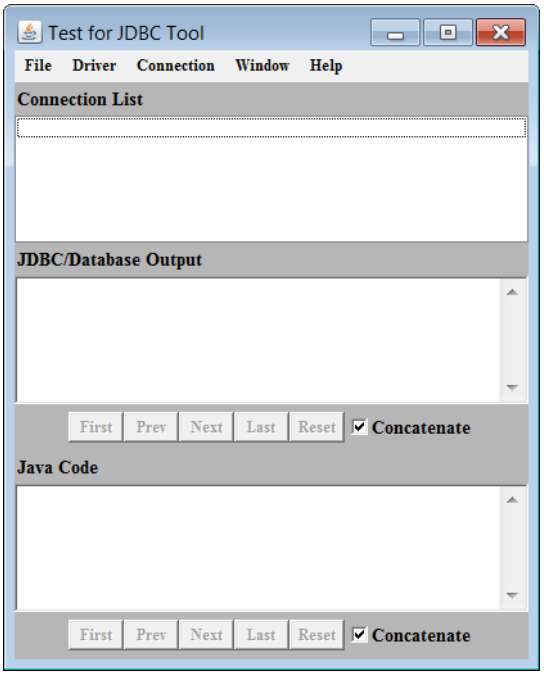
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

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



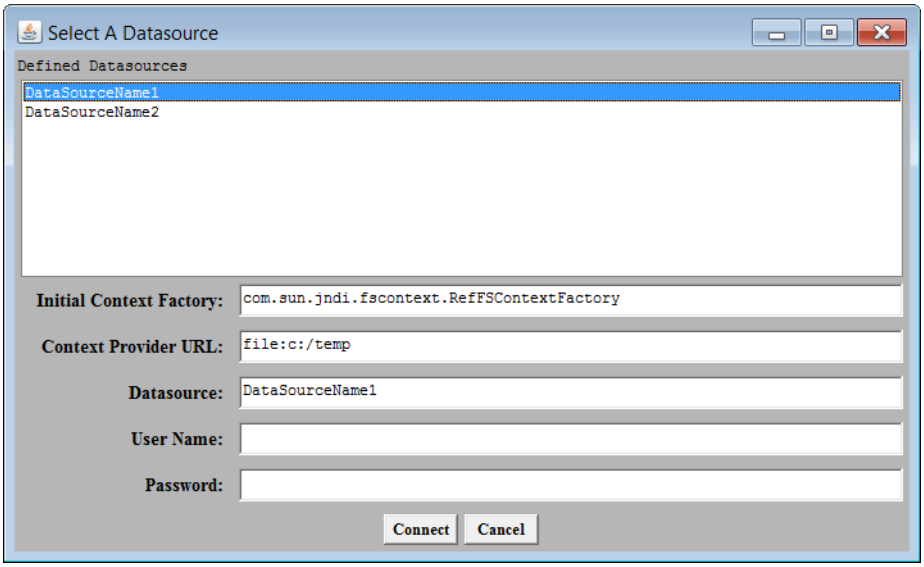
3. Click **Press Here to Continue**.

   The main dialog appears:



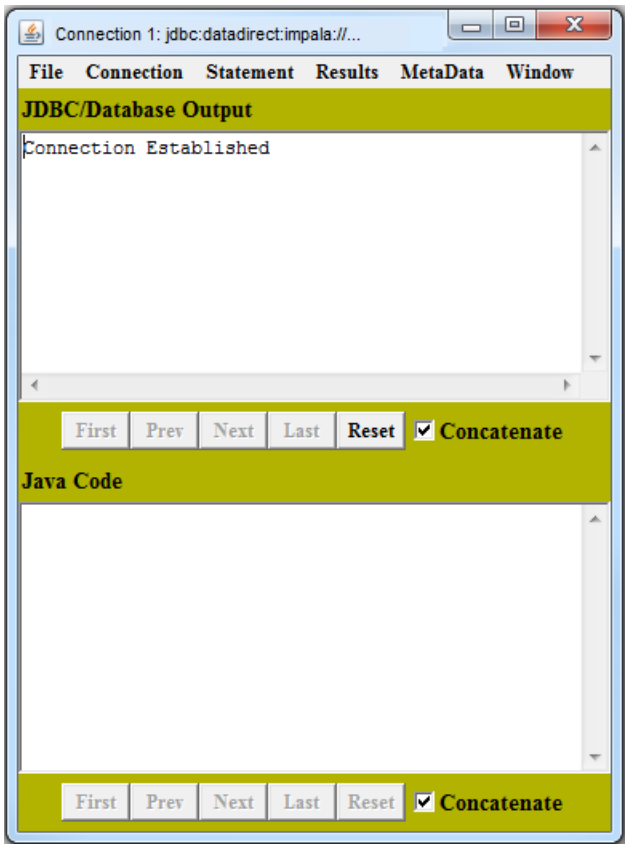
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   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.)


Connection 1: jdbc:datadirect:impala://...
File Connection Statement Results MetaData Window
JDBC/Database Output
Connection Established
First Prev Next Last Reset Concatenate
Java Code
First Prev Next Last Reset Concatenate

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
- Authentication
- Data Encryption
- Client Information
- IP Addresses
- Parameter Metadata Support
- ResultSet Metadata Support
- Isolation Levels
- Unicode support
- Error Handling
- Large Object Support

- Rowset Support
- Timeouts
- Using Scrollable Cursors
- Limitations on Cloudera Impala Functionality

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/51/-" {
   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/51/-" {
   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/51/-" {
// 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.

Connecting from an Application

Once the driver is installed and configured, you can connect from your application 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).

Connecting Using the JDBC Driver Manager

One way to connect to a database is through the JDBC Driver Manager using the DriverManager.getConnection() method. This method specifies a string containing a connection URL. This example shows how to establish a connection to a data source:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:impala://Server3:21050;
  User=admin;Password=adminpass");
```

Passing the Connection URL

After registering the driver, the required connection information needs to be passed in the form of a connection URL. The connection URL takes the form:

```
jdbc:datadirect:impala://servername:[port][;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 21050.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon. For more information on connection properties, see "Using Connection Properties."

The *install_dir*/samples directory, where *install_dir* is your product installation directory, contains a sample application that illustrates the steps of connecting to a Cloudera Impala server.

This example shows how to establish a connection to a Cloudera Impala server with user ID/password authentication:

```
Connection conn = DriverManager.getConnection
 ("jdbc:datadirect:impala://Server3:21050;
  DatabaseName=Test;User=admin;Password=adminpass");
```

See also

Using Connection Properties on page 44

Testing the Connection

You can use DataDirect Test™ to verify your connection. The screen shots in this section were taken on a Windows system.

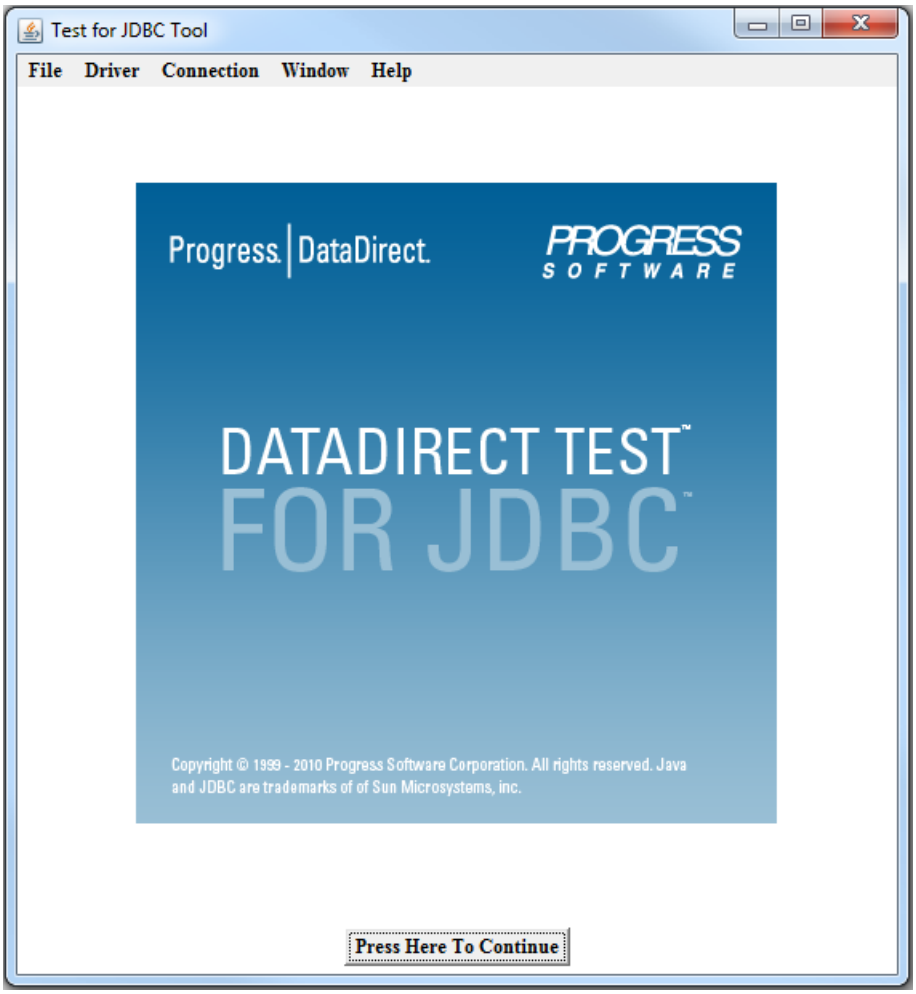
To test the connection from the driver to your data source, follow these steps:

1. Navigate to the installation directory. The default location is:
   - Windows systems: Program Files\Progress\DataDirect\JDBC_51\testforjdbc
   - UNIX and Linux systems: /opt/Progress/DataDirect/JDBC_51/testforjdbc

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

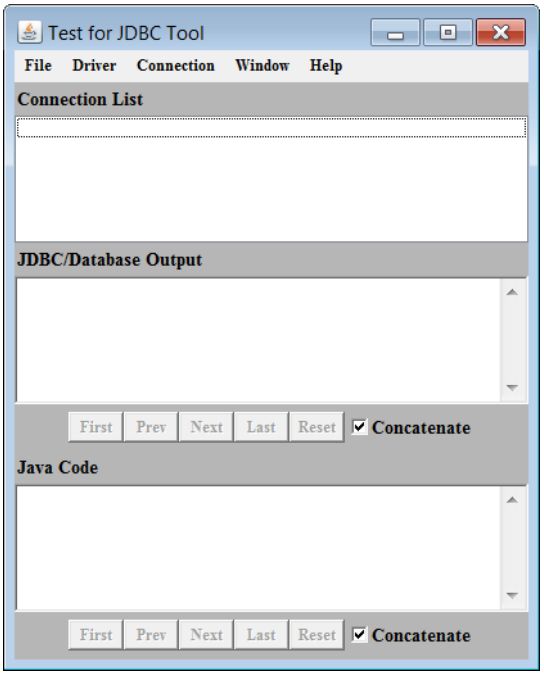
2. From the testforjdbc folder, run the platform-specific tool:
   - testforjdbc.bat (on Windows systems)
   - testforjdbc.sh (on UNIX and Linux systems)

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



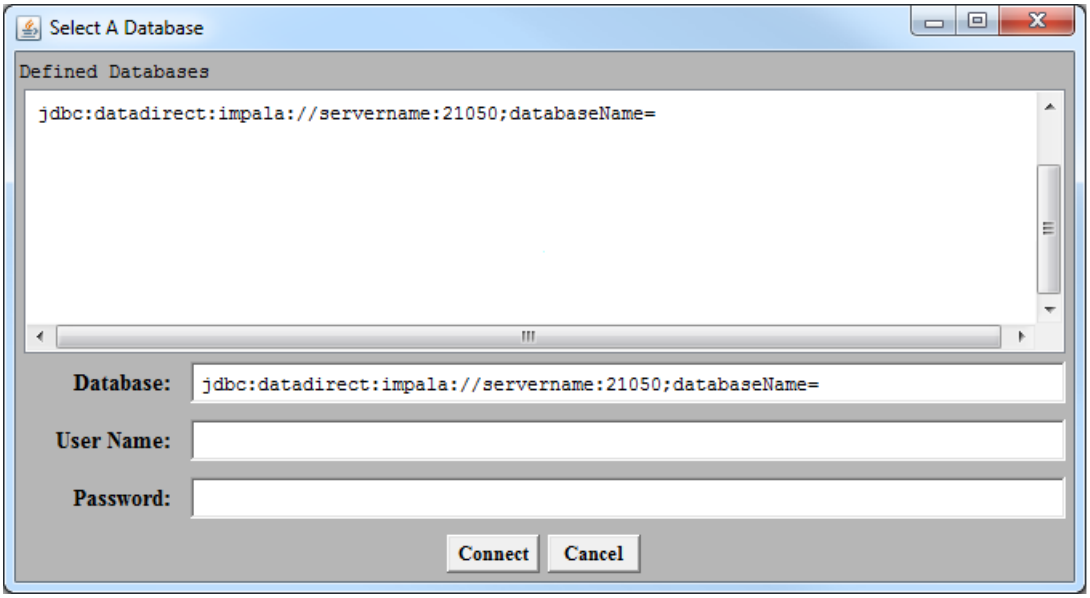
3. Click **Press Here to Continue**.

   The main dialog appears:



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

   The **Select A Database** dialog appears:



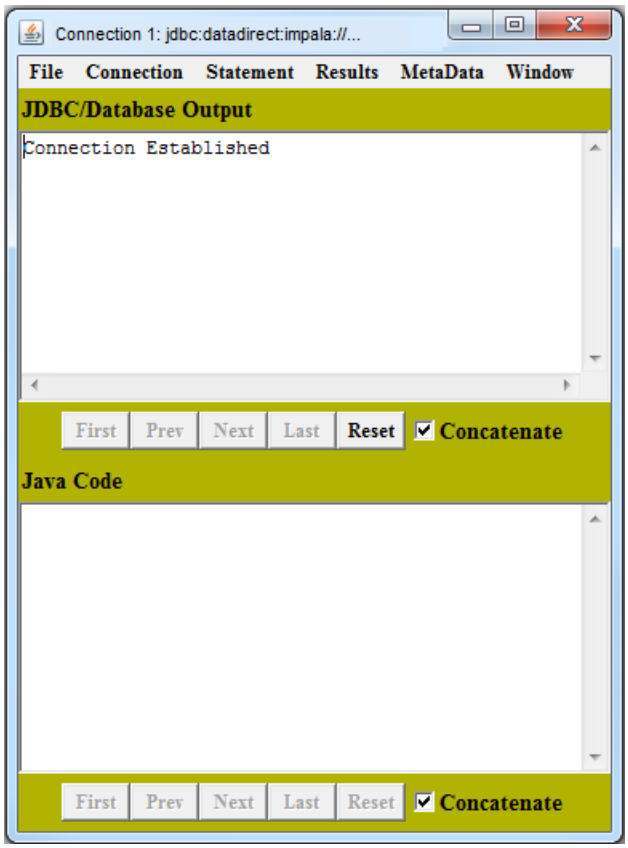
5. Select the appropriate database template from the Defined Databases field.
6. In the Database field, specify the correct ServerName and PortNumber for your Cloudera Impala data source.

   For example:

   ```
   jdbc:datadirect:impala://Server3:21050;databaseName=Test
   ```

7. If required, enter your user name and password in the fields provided.
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.)



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)

See "Data Source and Driver Classes" for data source class information.

See also

Data Source and Driver Classes on page 11

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.

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.

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:

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.impala.ImpalaDataSource;
```

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 connection:

```
ImpalaDataSource mds = new ImpalaDataSource();
mds.setDescription("My Impala Server");
mds.setServerName("MyServer");
mds.setPortNumber(21050);
mds.setDatabaseName("myDB");
```

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 ImpalaDataSource)
{
ImpalaDataSource jmds = (ImpalaDataSource) 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 DataSource Connection

You can use DataDirect Test™ to verify your connection. The screen shots in this section were taken on a Windows system.

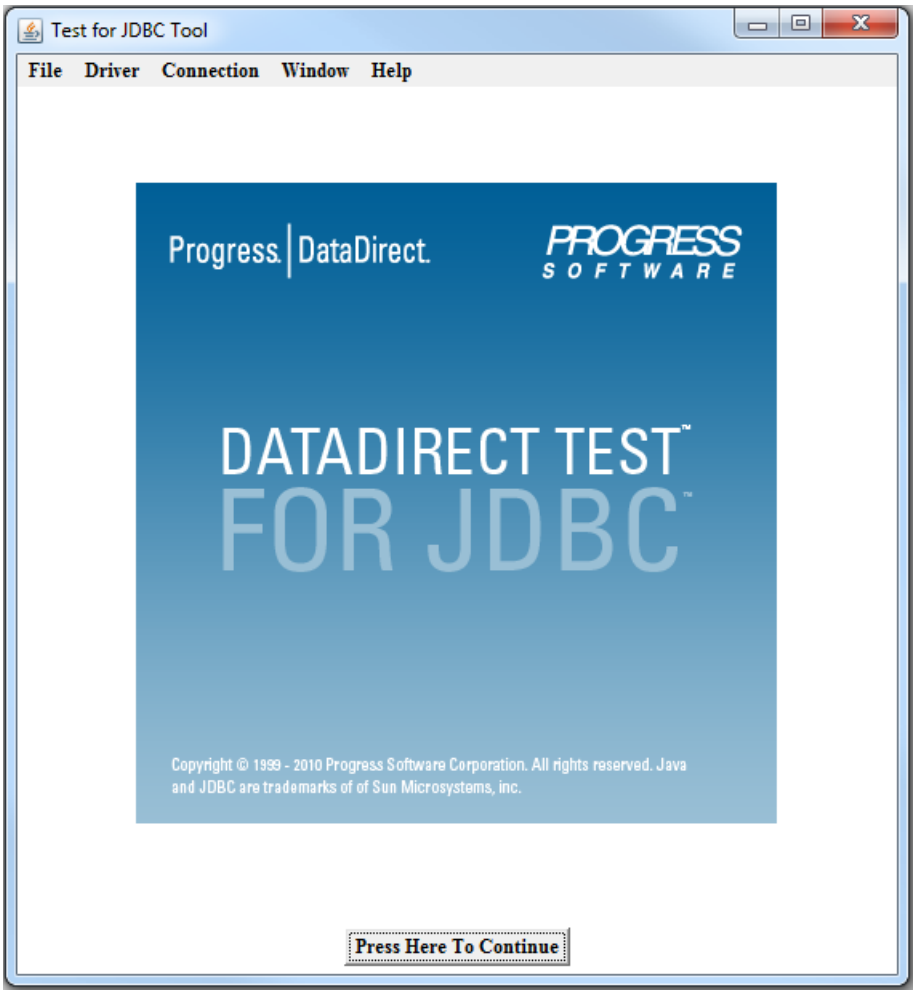
To test the connection from the driver to your data source, follow these steps:

1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC_51\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC_51/testforjdbc`

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

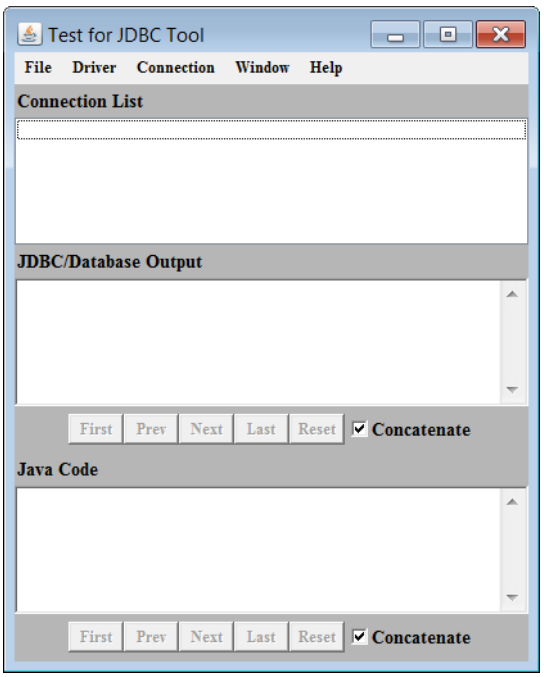
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

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



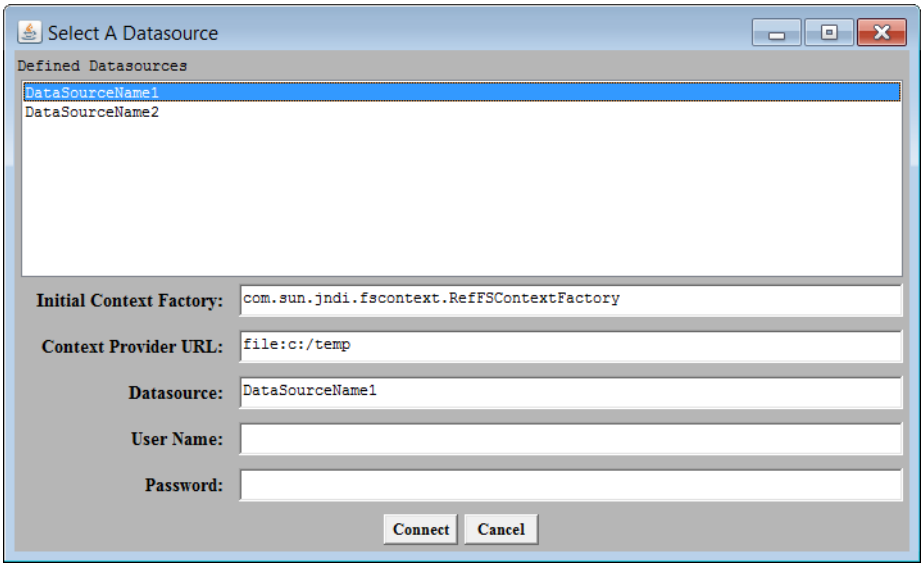
3. Click **Press Here to Continue**.

   The main dialog appears:



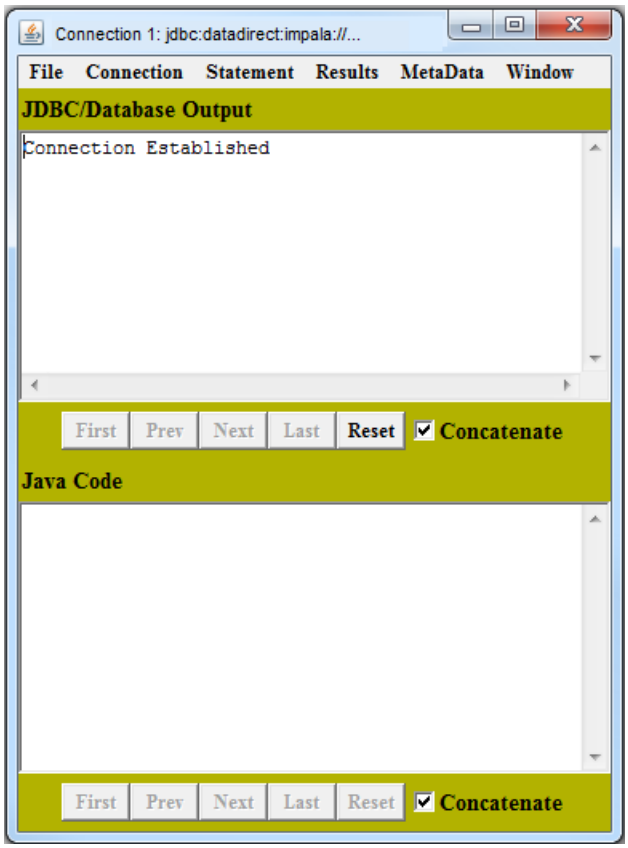
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   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.)



Using Connection Properties

You can use connection properties to customize the driver for your environment. Connection properties can be used to accomplish different tasks, such as implementing driver functionality or optimizing performance.

You can specify connection properties using either of the following methods:

- JDBC Driver Manager (see "Connecting Using the JDBC Driver Manager")
- JDBC data sources (see "Connecting Using Data Sources")

Connection properties take the form `property=value`. All connection property names are case-insensitive. For example, `Password` is the same as `password`.

See "Connection Property Descriptions" for an alphabetical list of connection properties and their descriptions.

See also

Connecting Using the JDBC Driver Manager on page 35
Connecting Using Data Sources on page 26
Connection Property Descriptions on page 69

UserID/Password Authentication Properties

This section describes the connection properties used to configure userID/password authentication.

Table 3: UserID/Password Properties

Property	Characteristic
AuthenticationMethod on page 74	Determines which authentication method the driver uses when establishing a connection. Set this to `userIdPassword` to enable the driver to use userID/password authentication.
DatabaseName on page 79	Specifies the name of the Impala database. The database must exist, or the connection attempt will fail.
Password on page 90	Specifies a password that is used to connect to your database.
PortNumber on page 91	The TCP port of the primary database server that is listening for connections to the database. The default is `21050`.
ServerName on page 94	Specifies either the IP address or the server name (if your network supports named servers) of the primary database server.
User on page 101	Specifies the user name that is used to connect to the database.

See also

Configuring user ID/password authentication on page 54

Connection Property Descriptions on page 69

Kerberos Authentication Properties

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

Table 4: Kerberos Authentication Properties

Property	Characteristic
AuthenticationMethod on page 74	Determines which authentication method the driver uses when establishing a connection. Set this to `kerberos` to enable the driver to use Kerberos authentication.
DatabaseName on page 79	Specifies the name of the Impala database. The database must exist, or the connection attempt will fail.
LoginConfigName on page 88	Specifies the name of the entry in the JAAS login configuration file that contains the authentication technology used by the driver to establish a Kerberos connection.

Property	Characteristic
PortNumber on page 91	The TCP port of the primary database server that is listening for connections to the database. The default is `21050`.
ServerName on page 94	Specifies either the IP address or the server name (if your network supports named servers) of the primary database server.
ServicePrincipalName on page 94	Specifies the service principal name to be used for Kerberos authentication.

See also

Configuring the driver for Kerberos authentication on page 54

Connection Property Descriptions on page 69

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 5: Data Encryption Properties

Property	Characteristic
CryptoProtocolVersion on page 78	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 81	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. **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`.

Property	Characteristic
HostNameInCertificate on page 82	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 86	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 86	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.
KeyStorePassword on page 87	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 99	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 100	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 102	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). If set to `true`, the driver validates the certificate that is sent by the database server. If set to `false`, the driver does not validate the certificate that is sent by the database server. The default is `true`.

See also

Connection Property Descriptions on page 69

Data Encryption on page 59

Data Type Handling Properties

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

Table 6: Data Type Handling Properties

Property	Characteristic
ConvertNull on page 77	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 85	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`.
StringDescribeType on page 98	Specifies whether String columns are described as VARCHAR columns. 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 69

Client Information Properties

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

Table 7: Client Information Properties

Property	Characteristic
AccountingInfo on page 72	Defines accounting information to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
ApplicationName on page 73	Specifies the name of the application to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
ClientHostName on page 75	Specifies the host name of the client machine to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
ClientUser on page 75	Specifies the user ID to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
ProgramID on page 91	Specifies the driver and version information on the client to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.

See also

- Connection Property Descriptions on page 69
- Client Information Properties on page 49

Statement Pooling Properties

The following table summarizes statement pooling connection properties.

Table 8: Statement Pooling Properties

Property	Characteristic
ImportStatementPool on page 83	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 89	The maximum number of pooled prepared statements for this connection. If set to `0` (the default), the driver's internal prepared statement pooling is not enabled. If set to a value greater than zero, the driver's internal prepared statement pooling is enabled. Enabling is useful when the driver is not running from within an application server or another application that provides its own prepared statement pooling. By default, the driver's statement pooling is not enabled.
RegisterStatementPoolMonitorMBean on page 92	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.

See also

- Connection Property Descriptions on page 69
- Performance Considerations on page 53
- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

Additional Properties

The following table summarizes additional connection properties.

Table 9: Additional Properties

Property	Characteristic
ConnectionRetryCount on page 76	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 Cloudera Impala server to which it tried to connect. The default is `5`.

Property	Characteristic
ConnectionRetryDelay on page 77	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` (second).
DefaultOrderByLimit on page 80	Specifies the maximum number of rows returned when a SQL statement containing an ORDER BY clause is executed. Cloudera Impala 1.3 requires statements containing the ORDER BY clause to limit the number of rows returned. This option allows these statements to return a result set without specifying a limit in the application. If set to `-1` (disabled), there is no default limit to the number of rows returned by a statement containing an ORDER BY clause. If set to *x*, the number of rows returned by a SQL statement containing an ORDER BY clause are limited to the specified number of rows for the session. The default is `-1` (disabled).
InitializationString on page 83	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 84	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`.
LoginTimeout on page 89	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`.
Password on page 90	Specifies a password that is used to connect to the database.

Property	Characteristic
RemoveColumnQualifiers on page 93	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` (Disabled).
SpyAttributes on page 95	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 98	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. 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. The default is `noTransactions`.
UseCurrentSchema on page 101	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`.
User on page 101	Specifies the user name that is used to connect to the database.

See also

- Connection Property Descriptions on page 69
- Performance Considerations on page 53

Performance Considerations

You can optimize application performance by setting connection properties as described in this topic.

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 does not provide 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.

Authentication

The driver supports the following authentication methods:

- *User ID/password authentication* authenticates using a database user name and password provided by the application.
- *Kerberos authentication* uses Kerberos, a trusted third-party authentication service, to verify user identities. Kerberos authentication can take advantage of the user name and password maintained by the operating system to authenticate users to the database or use another set of user credentials specified by the application.

  This method requires knowledge of how to configure your Kerberos environment and supports Windows Active Directory Kerberos only.

See also

Configuring user ID/password authentication on page 54
Configuring the driver for Kerberos authentication on page 54
Using Connection Properties on page 44

Configuring user ID/password authentication

Take the following steps to configure user ID/password authentication:

- Set the User property to specify the user ID.
- Set the Password property to specify the password.
- Set the DatabaseName property to specify the database name.
- Specify values for minimum required properties for establishing a connection.
  - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name for your server.
  - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.

For example, the following is a connection string with only the required properties for making a connection using user ID/password authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:impala://Server3:21050;
  DatabaseName=dbname;User=test;Password=secret);
```

See also

User on page 101
Password on page 90
DatabaseName on page 79
ServerName on page 94
PortNumber on page 91

Configuring the driver for Kerberos authentication

Your Kerberos environment should be fully configured before you configure the driver for Kerberos authentication. You should refer to your Impala and Java documentation for instructions on configuring Kerberos. For a Windows Active Directory implementation, you should also consult your Windows documentation. For a non-Active Directory implementation (on a Windows or non-Windows operating system), you should consult MIT Kerberos documentation.

Important: A properly configured Kerberos environment must include a means of obtaining a Kerberos Ticket Granting Ticket (TGT). For a Windows Active Directory implementation, Active Directory automatically obtains the TGT. However, for a non-Active Directory implementation, the means of obtaining the TGT must be automated or handled manually.

Once your Kerberos environment has been configured, take the following steps to configure the driver.

1. Use one of the following methods to integrate the JAAS configuration file into your Kerberos environment. (See "The JAAS Login Configuration File" for details.)

   Note: The *install_dir*/lib/JDBCDriverLogin.conf file is the JAAS login configuration file installed with the driver. You can use this file or another file as your JAAS login configuration file.

Note: Regardless of operating system, forward slashes must be used when designating the path of the JAAS login configuration file.

- 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");
```

- 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. When more than one login configuration file is specified, then the files are read and concatenated into a single configuration.

a) 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.

b) Find the line `# Default login configuration file` in the security properties file.

c) 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
```

2. Ensure your JAAS login configuration file includes an entry with authentication technology that the driver can use to establish a Kerberos connection. (See "The JAAS login configuration file" for details.)

Note: The JAAS login configuration file installed with the driver (*install_dir*`/lib/JDBCDriverLogin.conf`) includes a default entry with the name `JDBC_DRIVER_01`. This entry specifies the Kerberos authentication technology used with an Oracle JVM.

The following examples show that the authentication technology used in a Kerberos environment depends on your JVM.

Oracle JVM

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

IBM JVM

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

Note: The driver uses its default configuration for Kerberos authentication, if:

- A login configuration file is not specified using the `java.security.auth.login.config` system property.
- The login configuration file specified using the `java.security.auth.login.config` system property does not include a `JDBC_DRIVER_01` entry.
- The entry specified using the LoginConfigName connection property does not exist in the login configuration file specified using the `java.security.auth.login.config` system property.

3. Set the driver's AuthenticationMethod connection property to `kerberos`. (See "AuthenticationMethod" for details.)
4. Set the ServicePrincipalName property to specify the case-sensitive service principal name to be used for Kerberos authentication.

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

   The ServicePrincipalName takes the following form.

   `Service_Name/Fully_Qualified_Domain_Name@REALM_NAME`

   The value of the ServicePrincipalName property can include the Kerberos realm name, but it is optional. If you do not specify the realm name, the default realm is used. For example, if the service principal name, including Kerberos realm name, is server/Impala125ase1@XYZ.COM and the default realm is XYZ.COM, valid values for this property are:

   ```
   server/Impala125ase1@XYZ.COM
   ```

   and

   ```
   server/Impala125ase1
   ```

   See "ServicePrincipalName" for details on the composition of the service principal name.
5. Set the LoginConfigName connection property if the name of the JAAS login configuration file entry is different from the driver default `JDBC_DRIVER_01`. (See "The JAAS Login Configuration File" and "LoginConfigName" for details.)

   `JDBC_DRIVER_01` is the default entry name for the JAAS login configuration file (`JDBCDriverLogin.conf`) installed with the driver. When configuring your Kerberos environment, your network or system administrator may have used a different entry name. Check with your administrator to verify the correct entry name.
6. Set the User connection property as appropriate. (See "User" for details.)

   In most circumstances, there is no need to set the User connection property. By default, the driver uses the user principal name in the Kerberos Ticket Granting Ticket (TGT) as the value for the User property.
7. Set the DatabaseName connection property as appropriate. (See "DatabaseName" for details.)

See also

Kerberos authentication requirements on page 56
The JAAS login configuration file on page 57
AuthenticationMethod on page 74
ServicePrincipalName on page 94
LoginConfigName on page 88
User on page 101
DatabaseName on page 79

Kerberos authentication requirements

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

Note: For Windows Active Directory, the domain controller must administer both the database server and the client.

Table 10: Kerberos Configuration Requirements

Component	Requirements
Database server	No restrictions.
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 key distribution center (KDC). Network authentication must be provided by one of the following methods. • Windows Active Directory on Windows Server 2008 or higher. • MIT Kerberos 1.5 or higher
Client	Java SE 8 or higher must be installed.

The JAAS login configuration file

The Java Authentication and Authorization Service (JAAS) login configuration file contains one or more entries that specify authentication technologies to be used by applications. To establish Kerberos connections with the driver, the JAAS login configuration file must include an entry specifically for the driver. In addition, the login configuration file must be referenced either by setting the `java.security.auth.login.config` system property or by setting up a default configuration using the Java security properties file.

Setting up a default configuration

To set up a default configuration, you must 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. When more than one login configuration file is specified, then the files are read and concatenated into a single configuration. The following steps summarize how to modify the security properties file.

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
```

JAAS login configuration file entry for the driver

You can create your own JAAS login configuration file, or you can use the `JDBCDriverLogin.conf` file installed in the `/lib` directory of the product installation directory. In either case, the login configuration file must include an entry that specifies the Kerberos authentication technology to be used by the driver.

JAAS login configuration file entries begin with an entry name followed by one or more LoginModule items. Each LoginModule item contains information that is passed to the LoginModule. A login configuration file entry takes the following form.

```
entry_name {
  login_module flag_value module_options
};
```

where:

entry_name

is the name of the login configuration file entry. The driver's LoginConfigName connection property can be used to specify the name of this entry. `JDBC_DRIVER_01` is the default entry name for the `JDBCDriverLogin.conf` file installed with the driver.

login_module

is the fully qualified class name of the authentication technology used with the driver.

flag_value

specifies whether the success of the module is `required`, `requisite`, `sufficient`, or `optional`.

module_options

specifies available options for the LoginModule. These options vary depending on the LoginModule being used.

The following examples show that the LoginModule used for a Kerberos implementation depends on your JVM.

Oracle JVM

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

IBM JVM

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

Refer to Java Authentication and Authorization Service documentation for information about the JAAS login configuration file and implementing authentication technologies.

See also

LoginConfigName on page 88

Kerberos SASL-QOP

The driver supports Kerberos SASL-QOP data integrity and confidentiality. SASL-QOP is configured on the server side as part of a 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. The driver supports the following SASL-QOP values.

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

- `auth-conf`: authentication with confidentiality protection

See also

AuthenticationMethod on page 74

Data Encryption

The driver supports Secure Sockets Layer (SSL) data encryption. SSL is an industry-standard protocol for sending encrypted data over database connections. 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 TLS/SSL sockets created in the JVM.

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

  ```
  TrustStore=C:\Certficates\MyTruststore
  ```

  and

  ```
  TrustStorePassword=MyTruststorePassword
  ```

  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 TLS/SSL client authentication, the server asks the client to verify its identity after the server has proved its identity. Similar to TLS/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 TLS/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`).

- Specify values for the connection properties KeyStore and KeyStorePassword in the connection URL. For example:

  `KeyStore=C:\Certficates\MyKeyStore`

  and

  `KeyStorePassword=MyKeystorePassword`

Note: If the keystore specified by the KeyStore connection 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`).

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

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:impala://Server3:21050;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. This table lists the connection properties your application can use to store client information. Client information is stored locally. See "Connection Property Descriptions" for a detailed description of each property.
- 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 11: Client Information Properties

Property	Characteristic
AccountingInfo on page 72	Defines accounting information to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
ApplicationName on page 73	Specifies the name of the application to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
ClientHostName on page 75	Specifies the host name of the client machine to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
ClientUser on page 75	Specifies the user ID to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
ProgramID on page 91	Specifies the driver and version information on the client to be stored in the database. 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 69

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:impala://Server3:21050;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:impala://Server3:21050;
   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:impala://123.456.78.90:21050;
   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:

```
ImpalaDataSource mds = new ImpalaDataSource();
mds.setDescription("My ImpalaDataSource");
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 statements:

- `INSERT INTO foo VALUES (?, ?, ?)`
- `INSERT INTO foo (col1, col2, col3) VALUES (?, ?, ?)`

The driver allows the CAST function to be used in specified SQL. However, if a parameter marker is present within a cast operation, the driver will be unable to obtain the parameter metadata and will throw the exception "The requested parameter metadata is not available for the current statement."

Returning parameter metadata for Update and Delete statements is not supported because Cloudera Impala does not have the concept of Update and Delete statements.

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 = ?"
```

SQL Support

The driver provides support for standard SQL (primarily SQL-92). In addition, the product supports a set of SQL extensions. For example, the product supports extensions that allow you to change the default schema or set the maximum number of Web service calls the driver can make when executing a SQL statement.

See also

Supported SQL Functionality on page 103

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.

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][Impala 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][Impala JDBC Driver][Impala]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 Cloudera Impala 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.

Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

Visit https://www.jcp.org/en/jsr/detail?id=114 for more information about JSR 114.

Timeouts

The driver includes the LoginTimeout connection property. This property allows you to specify the amount of time the driver waits for a connection to be established before timing out the connection request. See "Using Connection Properties" and "LoginTimeout" for details.

See also

Using Connection Properties on page 44
LoginTimeout on page 89

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.

Limitations on Cloudera Impala Functionality

Please note the following Cloudera Impala functional limitations:

- No support for transactions
- No support for canceling a running query
- No support for row-level updates or deletes
- No support for stored procedures
- No support for auto-generated keys

For a more complete listing of known issues and limitations for your version of Cloudera Impala, refer to the Cloudera Impala user documentation:

http://www.cloudera.com/content/support/en/documentation.html

Note: Cloudera Impala is not designed for OLTP workloads and does not offer row-level updates or deletes. Instead, Impala is designed for batch type jobs over large data sets with high latency. This means that queries such as "`SELECT * FROM mytable`" return quickly. However, other SELECT statements are much slower.

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: 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.

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

Table 12: Impala Driver Properties

Property	Data Source Method	Default
AccountingInfo on page 72	setAccountingInfo	empty string
ApplicationName on page 73	setApplicationName	empty string
AuthenticationMethod on page 74	setAuthenticationMethod	`userIdPassword`
ClientHostName on page 75	setClientHostName	empty string

Property	Data Source Method	Default
ClientUser on page 75	setClientUser	empty string
ConnectionRetryCount on page 76	setConnectionRetryCount	5
ConnectionRetryDelay on page 77	setConnectionRetryDelay	1 (second)
ConvertNull on page 77	setConvertNull	1
CryptoProtocolVersion on page 78	setCryptoProtocolVersion	Determined by the JRE settings
DatabaseName on page 79	setDatabaseName	The default database
DefaultOrderByLimit on page 80	setDefaultOrderByLimit	-1
EncryptionMethod on page 81	setEncryptionMethod	noEncryption
HostNameInCertificate on page 82	setHostNameInCertificate	empty string
ImportStatementPool on page 83	setImportStatementPool	empty string
InitializationString on page 83	setInitializationString	None
InsensitiveResultSetBufferSize on page 84	setInsensitiveResultSetBufferSize	2048
JavaDoubleToString on page 85	setJavaDoubleToString	false
KeyPassword on page 86	setKeyPassword	None
KeyStore on page 86	setKeyStore	None
KeyStorePassword on page 87	setKeyStorePassword	None
LoginConfigName on page 88	setLoginConfigName	JDBC_DRIVER_01
LoginTimeout on page 89	setLoginTimeout	0
MaxPooledStatements on page 89	setMaxPooledStatements	None
Password on page 90	setPassword	None
PortNumber on page 91	setPortNumber	21050
ProgramID on page 91	setProgramID	empty string
RegisterStatementPoolMonitorMBean on page 92	setRegisterStatementPoolMonitorMBean	false
RemoveColumnQualifiers on page 93	setRemoveColumnQualifiers	false

Property	Data Source Method	Default
ServerName on page 94	setServerName	None
ServicePrincipalName on page 94	setServicePrincipalName	None
SpyAttributes on page 95	setSpyAttributes	None
StringDescribeType on page 98	setStringDescribeType	`varchar`
TransactionMode on page 98	setTransactionMode	`noTransactions`
TrustStore on page 99	setTrustStore	None
TrustStorePassword on page 100	setTrustStorePassword	None
UseCurrentSchema on page 101	setUseCurrentSchema	`false` (results are not restricted to the tables and views in the current schema)
User on page 101	setUser	None
ValidateServerCertificate on page 102	setValidateServerCertificate	`true`

For details, see the following topics:

- AccountingInfo
- ApplicationName
- AuthenticationMethod
- ClientHostName
- ClientUser
- ConnectionRetryCount
- ConnectionRetryDelay
- ConvertNull
- CryptoProtocolVersion
- DatabaseName
- DefaultOrderByLimit
- EncryptionMethod
- HostNameInCertificate
- ImportStatementPool
- InitializationString

- InsensitiveResultSetBufferSize
- JavaDoubleToString
- KeyPassword
- KeyStore
- KeyStorePassword
- LoginConfigName
- LoginTimeout
- MaxPooledStatements
- Password
- PortNumber
- ProgramID
- RegisterStatementPoolMonitorMBean
- RemoveColumnQualifiers
- ServerName
- ServicePrincipalName
- SpyAttributes
- StringDescribeType
- TransactionMode
- TrustStore
- TrustStorePassword
- UseCurrentSchema
- User
- 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

- Client Information on page 61
- Client Information Properties on page 49

ApplicationName

Purpose

Specifies the name of the application to be stored in the database. 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

- Client Information on page 61
- Client Information Properties on page 49

AuthenticationMethod

Purpose

Determines which authentication method the driver uses when establishing a connection. If the specified authentication method is not supported by the database server, the connection fails and the driver throws an exception.

Valid Values

`userIdPassword` | `kerberos`

Behavior

If set to `userIdPassword`, the driver uses user ID/password authentication. If a user ID and password is not specified, the driver throws an exception.

If set to `kerberos`, the driver uses Kerberos authentication. The ServicePrincipalName property must be specified.

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

- Authentication on page 53
- ServicePrincipalName on page 94

ClientHostName

Purpose

Specifies the host name of the client machine to be stored in the database. 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

- Client Information on page 61
- Client Information Properties on page 49

ClientUser

Purpose

Specifies the user ID to be stored in the database. 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

- Client Information on page 61
- Client Information Properties on page 49

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 Cloudera Impala 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

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

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 48

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 driver establishes an SSL connection using the default. If the default is not supported by the server, the connection fails and the driver returns an error.

Data Source Method

setCryptoProtocolVersion

Default

The default is determined by the settings of the JRE.

Data Type

String

See also

- EncryptionMethod on page 81
- Data Encryption on page 59

DatabaseName

Purpose

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

Valid Values

database_name

where:

database_name

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

Data Source Method

setDatabaseName

Default

The default database

Data Type

String

See also

UserID/Password Authentication Properties on page 44

DefaultOrderByLimit

Purpose

Specifies the maximum number of rows returned when a SQL statement containing an ORDER BY clause is executed. Cloudera Impala 1.3 requires statements containing the ORDER BY clause to limit the number of rows returned. This option allows these statements to return a result set without specifying a limit in the application.

Valid Values

`-1` | *x*

where:

x

is a positive integer that represents maximum number of rows returned.

Behavior

If set to `-1` (disabled), there is no default limit to the number of rows returned by a statement containing an ORDER BY clause. The application must limit the number of rows returned by SQL statements that contain an ORDER BY clause, or Cloudera Impala 1.3 databases will return an error.

If set to *x*, the number of rows returned by a SQL statement containing an ORDER BY clause are limited to the specified number of rows for the session. To override this value, specify a new value in a LIMIT clause in the statement that is being executed.

Data Source Method

setDefaultOrderByLimit

Default

`-1` (Disabled)

See also

Using Connection Properties on page 44

EncryptionMethod

Purpose

Determines whether SSL encryption is used to encrypt and decrypt data transmitted over the network between the driver and database server.

Valid Values

`noEncryption` | `SSL`

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.

Notes

- Connection hangs can occur if the driver attempts to connect to a database server that 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.
- 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 59

Performance Considerations on page 53

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

- EncryptionMethod on page 81
- Data Encryption on page 59

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.

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

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.

Data Source Method

setImportStatementPool

Default

empty string

Data Type

String

See also

- Statement Pooling Properties on page 49
- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

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:impala://ImpalaServer:21050;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

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

- EncryptionMethod on page 81
- Data Encryption on page 59

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.

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.

Valid Values

`string`

where:

string

is a valid directory of a keystore file.

Notes

- The keystore and truststore files can be the same file.

Data Source Method

setKeyStore

Default

None

Data Type

String

See also

- EncryptionMethod on page 81
- Data Encryption on page 59

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.

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.

Notes

- The keystore and truststore files can be the same file.

Valid Values

string

where:

string

is a valid password.

Data Source Method

setKeyStorePassword

Default

None

Data Type

String

See also

- EncryptionMethod on page 81
- Data Encryption on page 59

LoginConfigName

Purpose

Specifies the name of the entry in the JAAS login configuration file that contains the authentication technology used by the driver to establish a Kerberos connection. The LoginModule-specific items found in the entry are passed on to the LoginModule.

Valid Values

entry_name

where:

entry_name

is the name of the entry that contains the authentication technology used with the driver.

Example

In the following example, JDBC_DRIVER_01 is the entry name while the authentication technology and related settings are found in the brackets.

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

Default

JDBC_DRIVER_01

Data Type

String

See also

- Authentication on page 53
- The JAAS login configuration file on page 57

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

MaxPooledStatements

Purpose

Specifies the maximum number of pooled prepared statements for this connection. Setting MaxStatements to an integer greater than zero (0) enables the driver's internal prepared statement pooling, which is useful when the driver is not running from within an application server or another application that provides its own prepared statement pooling.

Valid Values

`0` | *x*

where

x

is a positive integer that represents a number of pooled prepared statements.

Behavior

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

If set to *x*, the driver enables the DataDirect Statement Pool and uses the specified value to cache a certain number of 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.

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

Alias

MaxStatements

See also

- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.
- Statement Pooling Properties on page 49

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

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

21050

Data Type

int

ProgramID

Purpose

Specifies the driver and version information on the client to be stored in the database. 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

Data Source Method

setProgramID

Default

empty string

Data Type

String

See also

- Client Information on page 61
- Client Information Properties on page 49

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.

Data Source Method

setRegisterStatementPoolMonitorMBean

Default

`false`

Data Type

boolean

See also

- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.
- Statement Pooling Properties on page 49
- MaxPooledStatements on page 89

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.

Data Source Method

setRemoveColumnQualifiers

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 `MyImpalaServer`

Data Source Method

setServerName

Default

None

Data Type

String

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).

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. It is the same value as the krb_svrname configuration parameter on the server. The default is impala.

Fully_Qualified_Domain_Name

is the fully qualified domain name (FQDN) of the host machine. This value must match the FQDN registered with the KDC. The FQDN consists of a host name and a domain name. For the example myserver.example.com, myserver is the host name and example.com is the domain name.

REALM.COM

is the domain name of the host machine. This value is optional. If no value is specified, the default domain is used. The domain must be specified in upper-case characters. For example, EXAMPLE.COM. For Windows Active Directory, the Kerberos realm name is the Windows domain name.

Example

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

```
impala/myserver.example.com@EXAMPLE.COM
```

Notes

- If AuthenticationMethod is set to userIdPassword, the value of the ServicePrincipalName property is ignored.

Data Source Method

setServicePrincipalName

Default

None

See also

- Authentication on page 53
- AuthenticationMethod on page 74
- Configuring the driver for Kerberos authentication on page 54

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).
load=*classname*	Loads the driver specified by *classname*.
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.

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)
```

Data Source Method

setSpyAttributes

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 columns. 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.

Data Source Method

setStringDescribeType

Default

`varchar`

Data Type

String

See also

Performance Considerations

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

Impala does not support transactions, and by default, the Impala 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.

Data Source Method

setTransactionMode

Default

`noTransactions`

Data Type

String

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.

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`.

Valid Values

`string`

where:

`string`

is the directory of the truststore file.

Data Source Method

setTrustStore

Default

None

Data Type

String

See also

- EncryptionMethod on page 81
- Data Encryption on page 59

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.

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`.

Valid Values

`string`

where:

`string`

is a valid password for the truststore file.

Data Source Method

setTrustStorePassword

Default

None

Data Type

String

See also

- EncryptionMethod on page 81
- Data Encryption on page 59

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

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.

Default

None

Data Type

String

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). 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.

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.

Data Source Method

setValidateServerCertificate

Default

`true`

Data Type

boolean

See also

- EncryptionMethod on page 81
- Data Encryption on page 59

5

Supported SQL Functionality

The Progress DataDirect Connect XE *for* JDBC for Impala driver supports an extended set of SQL-92, in addition to the syntax for Impala SQL, which is a subset of SQL-92 and HiveQL.

Refer to the Impala SQL Language Reference documentation for information about using Impala SQL.

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
- Restrictions

Data Definition Language

The Impala 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 Impala Language Reference documentation for information about using Impala 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)
```

Impala 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 Impala SQL, as shown in the following examples.

ANSI SQL 92 Query	Impala 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.
- A LIMIT clause or a default ORDER BY limit for result sets is required.

Default ORDER BY Limit

Impala will not return result sets for statements containing an order by clause unless a limit clause is included or a default limit is specified for result sets. A default limit can be set in the server; however, this limits the result sets for all queries, not just statements containing the ORDER BY clause.

The driver can work around these limitations by using the DefaultOrderByLimit connection property.

where:

x

is a positive integer that represents maximum number of rows returned.

If set to *x*, the number of rows returned by a SQL statement containing an ORDER BY clause are limited to the specified number of rows for the session. To override this value, specify a new value in a LIMIT clause in the statement that is being executed.

If set to *-1*(disabled), there is no default limit the number of rows returned by a statement containing an ORDER BY clause. The application must limit the number of rows returned by SQL statements that contain an ORDER BY clause, or Impala will return an error.

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 or 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 Impala 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:

Constants

Impala 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 13: Numeric Operators

Entry SQL Level Operator	Impala 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 Impala SQL.

```
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 14: Relational Operators Supported with Impala

Entry SQL Level Operator	Support in Impala SQL
<>	Supported
<	Supported
<=	Supported
=	Supported
<=>	Supported
>	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 15: Logical Operators

Operator	Support in Impala HQL
NOT !	Supported
AND &&	Supported
OR ||	Supported

Functions

The following tables show how SQL-92 functions are supported in the Impala Query Language. Additional methods may be supported with JDBC Escapes.

Table 16: Set Functions Supported

Set Function	Support in Impala SQL
Count	Supported
AVG	Supported
MIN	Supported
MAX	Supported
SUM	Supported
DISTINCT	Supported
ALL	Supported

Table 17: Numeric Functions Supported

Numeric Function	Support in Impala 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 18: String Functions Supported

String Function	Support in Impala SQL
Substring	Supported

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

Table 19: Date/Time Functions Supported

Date/Time Function	Support in Impala SQL
CURRENT_DATE( )	Not supported[11]
CURRENT_TIME( )	Not supported[11]
CURRENT_TIMESTAMP	Not supported. Use UNIX_TIMESTAMP()[11].

Table 20: System Functions Supported

System Function	Support in Impala SQL
CASE . . . END	Supported.
COALESCE	Supported.
NULLIF	Not supported.[11]
CAST	Supported.

Refer to "Scalar functions" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

[11] Supported by JDBC Escapes

Restrictions

Cloudera Impala has the following SQL restrictions:

- Column values and parameters are always nullable.
- No support for row-level updates or deletes
- No support for stored procedures
- No ROWID support
- No support for materialized views
- No support for synonyms
- Primary and foreign keys are not supported.
- Support for indexes is incomplete.
- A single quote within a string literal must be escaped using a \ instead of using a single quote. Because string literals can be expressed with either single or double quotation marks, *Impala's* would be written as *'Impala\'s'* or *"Impala\'s"*.