Progress® DataDirect®

Progress DataDirect for JDBC for Apache Impala

User's Guide

Release 6.0.0

Copyright

Table of Contents

1

Welcome to the Progress DataDirect for JDBC for Apache Impala Driver

The Progress® DataDirect® for JDBC™ for Apache Impala™ driver supports SQL read-write access for JDBC applications to Impala. The driver provides access to data in Impala that can be pulled into a data visualization tool to get important insights into engineering operations. In addition, the driver employs a SQL engine component that provides support to SQL constructs unavailable in Impala. This functionality allows seamless integration with third-party software and provides the most comprehensive SQL support and JDBC standard connectivity to BI (Business Intelligence) and ETL (Extract,Transform, Load) tools.

The Apache Impala server is compatible with data stored in various file formats. In addition, 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-impala.

For details, see the following topics:

- Data Types
- Driver specifications
- DataDirect tools
- Troubleshooting
- Additional information
- Contacting Technical Support

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 6.0.0 GA

- **Enhancements**
  - The driver has been enhanced to comply with FIPS standards for data encryption. As part of this enhancement, the driver was tested with FIPS 140-3 enabled using a Red Hat OpenJDK 21 on a Red Hat Universal Base Image 9 instance. See FIPS (Federal Information Processing Standard) on page 49 for details.
- **Changed Behavior**
  - The connection property `SpyAttributes` has been updated to exclude the attribute `load=classname`, which was previously used to load the driver specified by the given class name. See SpyAttributes on page 90 for details.

Highlights of 6.0.0 Release

- **Enhancements**
  - The driver supports SQL read-write access to Apache Impala data stores. See Supported SQL statements and extensions on page 99 for details.
  - The driver supports JDBC core functions. For details, refer to "JDBC support" in the *Progress DataDirect for JDBC Drivers Reference*.
  - The driver has been enhanced to support the Array data type through data type inference. See Data Types on page 14 and getTypeInfo on page 15 for details.
  - The driver has been enhanced to provide proxy support. See Connection URL examples on page 13 and Connection property descriptions on page 55 for more information.
  - The driver supports Kerberos authentication. See Kerberos authentication on page 41 for details.
  - The driver has been enhanced to support Impersonation with Kerberos authentication using the new ImpersonateUser property. See for ImpersonateUser on page 73 details.

- The driver has been enhanced to support LDAP authentication with user ID and password. See User ID/password authentication on page 39 for details.
- The driver supports No authentication, which allows you to connect without any authentication credentials. See No authentication on page 40 for details.
- Interactive SQL is now installed with the product. Interactive SQL is a command-line interface that supports connecting your driver to a data source, executing SQL statements and retrieving results in a terminal. This tool provides a method to quickly test your drivers in an environment that does not support GUIs. See Interactive SQL on page 26 for details.

- **Changed behavior**
  - The default authentication method used by the driver has been updated to user ID/password via LDAP authentication (`AuthenticationMethod=userIdPassword`). See User ID/password authentication on page 39 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.

Installing and setting up the driver

This section provides you with an overview of the steps required to install and set-up the driver. After completing this procedure, you will be able to begin accessing data with your application.

To begin accessing data with the driver:

1. Install the driver:
   a) After downloading the product, unzip the installer files to a temporary directory.
   b) From the installer directory, run the appropriate installer file to start the installer.
      - **Windows**: `PROGRESS_DATADIRECT_JDBC_INSTALL.exe`
      - **Non-Windows**: `PROGRESS_DATADIRECT_JDBC_INSTALL.jar`
   c) Follow the prompts to complete installation.

   The installer program supports multiple installation methods, including command-line and silent installations. For detailed instructions, refer to the *Progress DataDirect for JDBC Drivers Installation Guide*.
2. Set your system CLASSPATH to include the driver `.jar` file. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. The following examples demonstrate setting the CLASSPATH from a command line using the default installation directory.
   - **Windows Example**

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

- **UNIX/LINUX Example**

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

3. Configure your driver using one of the following methods:

- **Connection URL**: You can begin using the driver immediately by passing a connection URL with your application or tool. The following examples show how to connect using either user ID and password or Kerberos authentication.

  UserID/Password

```
jdbc:datadirect:impala://myserver:21050;DatabaseName=ImpalaDB;
AuthenticationMethod=UserIDPassword;User=JSmith@example.com;Password=secret;
```

  Note: See User ID/password authentication on page 39 for details.

  Kerberos

```
jdbc:datadirect:impala://myserver:21050;DatabaseName=ImpalaDB;
AuthenticationMethod=kerberos;ServicePrincipleName=impala/myserver@EXAMPLE.COM;
```

  Note: See Kerberos authentication on page 41 for details.

- **Data sources**: The driver also supports connecting using JDBC 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. See Connecting using data sources for more information.

Note: For most connections, specifying the minimum required connection properties is sufficient to begin accessing data; however, you can provide values for optional properties to use additional supported features and improve performance.

4. Set the values for any optional properties that you want to configure. For additional information on optional features and functionality, see the following resources:

- Connection URL Examples provides connection string examples that can be used to configure common functionality and features. You can modify and combine these examples to create a string that best suites your environment.
- Connection Property Descriptions provides a complete list of supported properties by functionality.
- Performance Considerations describes connection properties that affect performance, along with recommended settings.

5. Connect to your service and begin accessing data with your applications, BI tools, database tools, and more. To help you get started, the following resources guide you through accessing data with some common tools:

- DataDirect Test: DataDirect Test allows you to test connect, execute SQL statements, and practice using the JDBC API right out of the box.
- Tableau: Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data.
- DbVisualizer: DB Visualizer is a database tool that allows you to connect and execute SQL statements against your data.

- Supported SQL statements and extensions: This section describes the syntax used for SQL statements supported by the driver. You can modify and use the provided examples for your application or tool.

This completes the deployment of the driver.

Driver and DataSource classes

The following are the `Driver` and `DataSource` classes used by the driver:

Driver class:

com.ddtek.jdbc.impala.ImpalaDriver

DataSource class:

com.ddtek.jdbcx.impala.ImpalaDataSource

Connection URL examples

After setting the CLASSPATH, the connection information needs to be passed in the form of a connection URL. This section provides examples of connection strings configured to use common features and functionality. You can modify and/or combine these examples to create a connection string for your environment.

Note:

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

```
jdbc:datadirect:impala://servername:port;DatabaseName=database_name;
[property=value[;...]];
```

- No authentication
- User ID/password authentication
- Kerberos authentication
- Proxy server authentication

No authentication

This string includes the properties used to connect without an authentication method.

```
jdbc:datadirect:impala://servername:port;DatabaseName=database_name;
AuthenticationMethod=none;[property=value[;...]];
```

For more information on these properties and values, see No authentication on page 40.

User ID/password authentication

This string includes the properties used to connect with the user ID/password authentication via LDAP.

```
jdbc:datadirect:impala://servername:port;DatabaseName=database_name;
User=user_name;Password=password;[property=value[;...]];
```

Note: The user credentials are configured in the LDAP server.

For more information on these properties and values, see User ID/password authentication on page 39.

Kerberos authentication

This string includes the properties used to connect with Kerberos authentication.

```
jdbc:datadirect:impala://servername:port;DatabaseName=database_name;
AuthenticationMethod=kerberos;ServicePrincipleName=serviceprincipalname;
[property=value[;...]];
```

For more information on these properties and values, see Kerberos authentication on page 41.

Proxy server authentication

This string includes the properties you may need to connect through a proxy server with the user ID/password authentication via LDAP.

```
jdbc:datadirect:impala://servername:port;DatabaseName=database_name;
ProxyHost=proxy_host;ProxyPassword=proxy_password;
ProxyPort=proxy_port;ProxyUser=proxy_user;
User=user_name;Password=password;
[property=value[;...]];
```

See also

Authentication on page 39

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
Array[1]	ARRAY
Bigint	BIGINT
Boolean	BOOLEAN
Char	CHAR

[1] Supports only DQL with Array data type

Impala	JDBC
Decimal	DECIMAL
Double	DOUBLE
Float	REAL
Int	INTEGER
Smallint	SMALLINT
String[2]	VARCHAR *or* LONGVARCHAR[3]
Timestamp	TIMESTAMP
Tinyint	TINYINT
Varchar	VARCHAR

getTypeInfo

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

Table 2: getTypeInfo

TYPE_NAME = array[4] AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 2003 (ARRAY) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = array 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

[3] 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.

[2] Maximum of 2 GB

[4] Supports only DQL with Array data type

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

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

[5] Maximum of 2 GB

[6] 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 = timestamp AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = {ts' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = timestamp MAXIMUM_SCALE = 9	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 29 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = tinyint AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -6 (TINYINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = Y LOCAL_TYPE_NAME = tinyint MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 3 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = varchar AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = length DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = varchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 65535 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Driver specifications

This section describes the general functionality supported by the driver.

- **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.

- **Isolation levels**: The driver supports the Read Committed isolation level.

- **Scrollable cursor support**: The driver supports scroll-insensitive 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.

- **Error handling**: 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

DataDirect tools

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

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

Troubleshooting

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

Additional information

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

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

Contacting Technical Support

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

https://www.progress.com/support

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

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

- Your number or the serial number that corresponds to the product for which you are seeking support, or a case number if you have been provided one for your issue. If you do not have a SupportLink contract, the SupportLink representative assisting you will connect you with our Sales team.
- Your name, phone number, email address, and organization. For a first-time call, you may be asked for full information, including location.
- The Progress DataDirect product and the version that you are using.
- The type and version of the operating system where you have installed your product.
- Any database, database version, third-party software, or other environment information required to understand the problem.
- A brief description of the problem, including, but not limited to, any error messages you have received, what steps you followed prior to the initial occurrence of the problem, any trace logs capturing the issue, and so

on. Depending on the complexity of the problem, you may be asked to submit an example or reproducible application so that the issue can be re-created.

- A description of what you have attempted to resolve the issue. If you have researched your issue on Web search engines, our Knowledgebase, or have tested additional configurations, applications, or other vendor products, you will want to carefully note everything you have already attempted.
- A simple assessment of how the severity of the issue is impacting your organization.

2

Tutorials

The following sections guide you through using the driver to access your data with some common third-party applications. For information on installing your driver and setting the CLASSPATH, see "Installing and setting-up the driver."

For details, see the following topics:

- Tableau
- DbVisualizer
- Interactive SQL

Tableau

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with Tableau. Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data. By using the driver with Tableau, you can improve performance when retrieving data while leveraging the driver's relational mapping tools.

Note: The driver supports only Windows Active directory Kerberos when using Tableau with Kerberos.

To use the driver to access data with Tableau:

1. Navigate to the `\lib\xx` subdirectory of the Progress DataDirect installation directory; then, locate the `jar` file for your driver:

   `impala.jar`

2. Copy the `.jar` file for your driver into the following directory:

   Windows: `C:\Program Files\Tableau\Drivers`

   Linux: `/opt/tableau/tableau_driver/jdbc`

3. Open Tableau. From the **Connect** menu, select **Other Databases (JDBC)**.
4. In the **Other Databases (JDBC)** dialog, provide values for the following fields; then, click **Sign In**.

   - **URL**: Copy and paste your connection URL into this field. The following examples show how to connect using UserID and Password authentication.

     UserID and Password authentication

     `jdbc:datadirect:impala://myserver:21050;DatabaseName=`*`database_name`*`;`

     Note: See User ID/password authentication on page 39 for details.

   - **Dialect**: Select **SQL92** (the default) from the drop-down box.
   - **User**: If required by the authentication method (UserIdPassword) being used, enter the user name. Alternatively, this value can be specified with the `User` property in the connection string.
   - **Password**: If required by the authentication method (UserIdPassword) being used, enter the password. Alternatively, this value can be specified with the `Password` property in the connection string.

5. The **Data Source** window appears. In the **Schema** field, select the schema for the service you want to use.
6. In the **Table** field, the tables stored in the selected schema are now exposed and available for selection.

You have successfully accessed your data and are now ready to create reports with Tableau. For detailed information, refer to the Tableau product documentation at: https://www.tableau.com/support/help.

DbVisualizer

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with the third-party DbVisualizer tool. The following topics guide you through using DbVisualizer to add your driver, connect, and execute SQL statements.

Adding a driver

To add a driver with DbVisualizer:

1. Open DbVisualizer.
2. From the menu, select **Tools**>**Driver Manager**. The Driver Manager window opens.
3. From the Driver Manager menu, select **Driver**>**Create Driver**.

4. Click the  button to navigate to the location of the driver jar file; then, click **OK**. The following are the default locations for the driver:

   Windows

   ```
   C:\Program Files\Progress\DataDirect\JDBC\lib\60\impala.jar
   ```

   Linux

   ```
   /opt/Progress/DataDirect/JDBC/lib/60/impala.jar
   ```

5. Provide values for the following fields; then, close the Driver Manager window.

   - **Name**: Type an alias for your driver. For example:

     ```
     Impala
     ```

   - **URL Format**: Optionally, specify the format of the connection string for your driver. For example:

     ```
     jdbc:datadirect:impala://myserver:21050
     ```

   - **Driver Class**: From the drop down menu, select the driver class for your driver. For example:

     ```
     com.ddtek.jdbc.impala.ImpalaDriver
     ```

You can now use your driver with DbVisualizer. Proceed to "Connecting and executing SQL statements" for information on connecting and executing SQL statements.

Connecting and executing SQL statements

To use the driver to access data with DbVisualizer:

1. Open DbVisualizer.
2. From the menu, select **Database**>**New Connection**. When prompted to use the Connection Wizard, click **OK**.
3. Provide the following information when prompted; then, click **Next** to proceed:

   - **Connection alias**: Type the name to be used when referring to this connection.
   - **Driver**: Select the alias that you provided for your driver from the drop-down menu.

4. Provide values for the following fields; then, click **Finish**.

   - **Database URL**: Copy and paste your connection URL into this field. The following examples show how to connect using user ID/password authentication.

     UserID/Password

     ```
     jdbc:datadirect:impala://myserver:21050;DatabaseName=ImpalaDB;
     AuthenticationMethod=UserIDPassword;User=JSmith@example.com;Password=secret;
     ```

   - **Database UserId**: If required by the authentication method being used (UserIdPassword), enter the user name. Alternatively, this value can be specified with the `User` property in the connection string.

- **Database Password**: If required by the authentication method being used (UserIdPassword), enter the password. Alternatively, this value can be specified with the `Password` property in the connection string.

5. To execute SQL statements, select **SQL Commander**>**New SQL Commander**. A SQL Commander tab opens.

   Note: Remove the default enclosing double quotes for identifiers. To configure this, go to `Driver>Properties>Generic>Delimited Identifiers`.

6. Select values for the following fields:

   - **Database Connection**: Select connection alias you provided for the connection from the drop-down menu.
   - **Schema**: Select the schema you want to execute queries against from the drop-down menu.

7. In the SQL Commander tab, enter SQL commands you want to execute; then select **SQL Commander**>**Execute**. For example:

   To select all of the rows from the `USERS` table:

   ```
   SELECT * FROM USERS
   ```

   To select the URLs for a specified issue:

   ```
   SELECT * FROM USERS WHERE USERID = <ID>
   ```

   See "Supported SQL statements and extensions" for the supported syntax used to execute SQL statements.

   Note: If you are fetching large sets of data, you may want to limit the results using the Max Rows and Max Chars fields.

You have successfully accessed your data with DbVisualizer.

Interactive SQL

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

To execute commands with Interactive SQL:

1. Start the ISQL tool. From a command line, enter the following:

   ```
   java -jar impala.jar --isql
   ```

2. Enter connection properties one at a time by typing *`property=value`*, then pressing **Enter**. For example, to configure the ServerName property:

   ```
   ServerName=myserver
   ```

3. After specifying values for your properties, type `connect`, then press **Enter**. If successful, the tool will return a confirmation message.

Note: If you are unable to connect, you can review the URL by entering the `SHOW URL` command.

4. At the `ISQL>` prompt, issue a SQL command to query or modify the data source; then, press **Enter**. For example:

   ```
   SELECT * FROM USERS;
   ```

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

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

   The results of the command are displayed in the terminal.

5. After you are finished executing queries and commands, you can disconnect from the data source by typing the following; then, pressing **Enter**:

   ```
   DISCONNECT
   ```

6. To end the session, type `exit`; then, press **ENTER**.

3

Configuring and connecting

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.

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

- Using the JDBC `DriverManager` 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).

For details, see the following topics:

Setting the classpath

The driver must be defined on your CLASSPATH before you can connect. 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 driver jar file as shown, where *`install_dir`* is the path to your product installation directory.

```
install_dir/lib/60/impala.jar
```

Windows Example

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

UNIX Example

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

Connecting using the JDBC Driver Manager

One way to connect to a service is through the JDBC DriverManager using the `DriverManager.getConnection()` method. As the following examples show, this method specifies a string containing a connection URL.

UserID/Password authentication

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:impala://Server3:21050;DatabaseName=database_name;
  UserName=user_name;Password=password");
```

Note: See User ID/password authentication on page 39 for details.

Passing the connection URL

After setting the CLASSPATH, the required connection information needs to be passed in the form of a connection URL. The following example includes the properties required for connecting with UserID/Password authentication.

Connection URL Syntax

The connection URL takes the following form:

```
jdbc:datadirect:impala://myserver:21050;DatabaseName=database_name;
UserName=user_name;Password=password;[property=value[;...]];
```

where:

server_name

specifies the base URL of the Impala instance to which you want to issue requests. For example, `//myserver:21050`.

port_number

specifies the port number of the server listener. The default is `21050`.

database_name

specifies the name of the database to which you are connecting.

user_name

Specifies the user ID of LDAP used to authenticate to Impala with UserID/Password authentication method.

password

Specifies the password of LDAP used to authenticate to Impala with UserID/Password authentication method.

Important: The password is a confidential value used to authenticate to the server. To prevent unauthorized access, this value must be securely maintained.

property=value

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

The following example connection string includes the properties required for connecting with the UserID/Password authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:impala://myserver:21050;DatabaseName=ImpalaDB;
  UserName=JohnSmith@example.com;Password=abc123");
```

See also

Connection property descriptions on page 55
Connection URL examples on page 13

Testing the connection

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

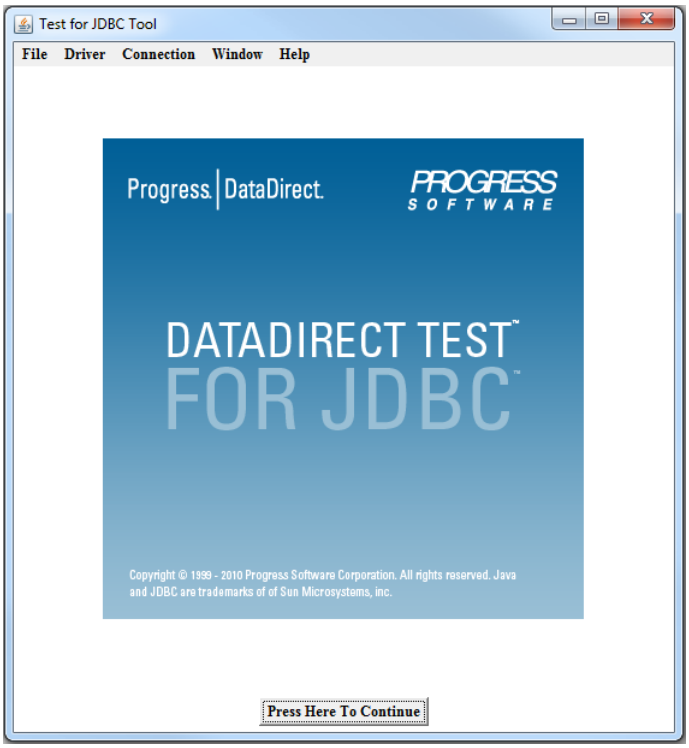
To test the connection from the driver to your data source, follow these steps:

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

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

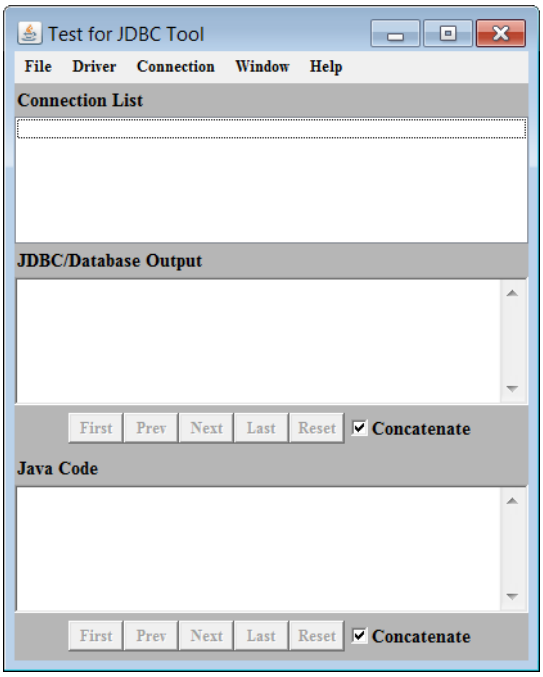
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

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



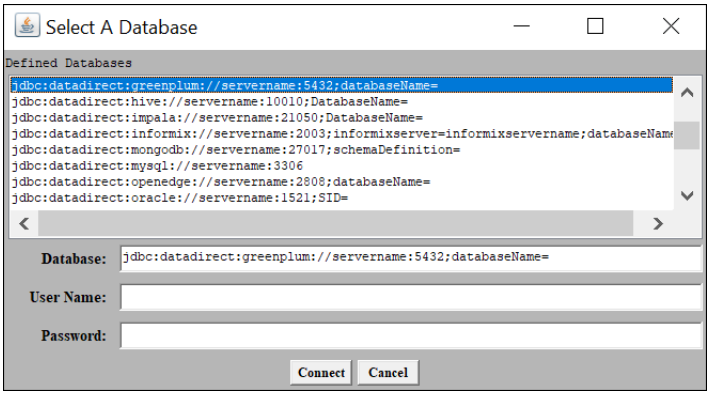
3. Click **Press Here to Continue**.

   The main dialog appears:



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

   The **Select A Database** dialog appears:



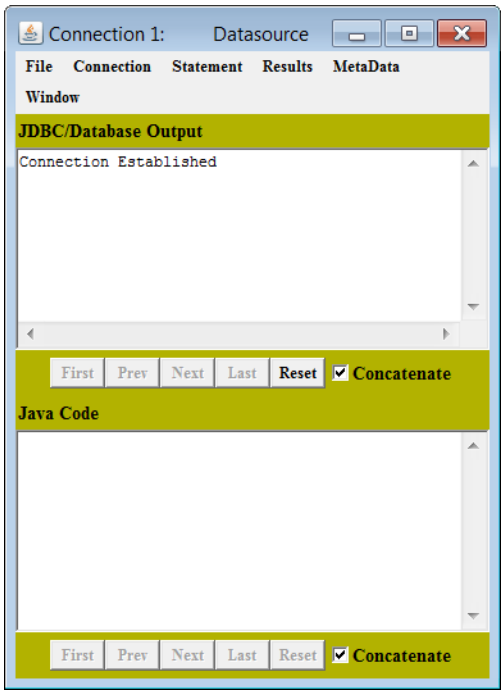
5. Select the appropriate database template from the **Defined Databases** field.
6. In the **Database** field, specify all required connection properties.

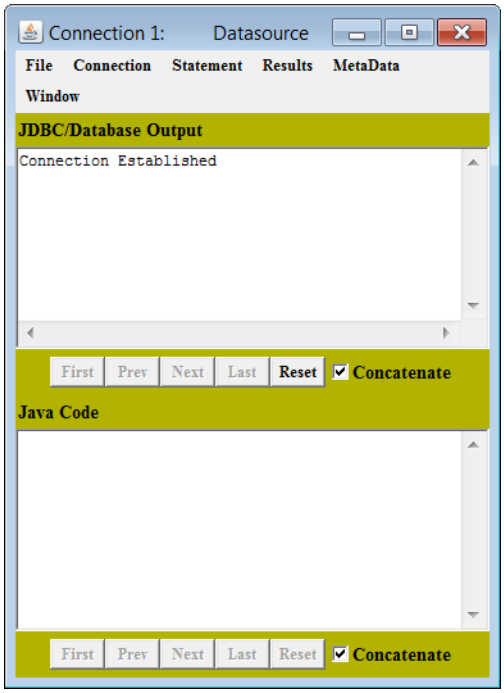
   For example:

   ```
   jdbc:datadirect:impala://myserver:21050;DatabaseName=ImpalaDB;
   User=JSmith@example.com;Password=secret;
   ```

7. 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 `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 `DataSource` class. A data source class implements the following interfaces.

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

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

See also

Driver and DataSource classes on page 13

Creating data sources

The following example files provide details on creating and using Progress DataDirect data sources with the Java Naming Directory Interface (JNDI), where *`install_dir`* is the product installation directory.

- *`install_dir`*`/Examples/JNDI/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.
- *`install_dir`*`/Examples/JNDI/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.

See "Example data source" for an example data source definition for the example files.

To connect using a JNDI data source, the driver needs to access a JNDI data store to persist the data source information. For a JNDI file system implementation, you must download the File System Service Provider from the Oracle Technology Network Java SE Support downloads page, unzip the files to an appropriate location, and add the `fscontext.jar` and `providerutil.jar` files to your CLASSPATH. These steps are not required for LDAP implementations because the LDAP Service Provider is included with supported versions of Java SE.

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 using the UserID/Password authentication.

Note:

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

```
ImpalaDataSource mds = new ImpalaDataSource();
mds.setDescription("My Impala Data Source");
mds.setServerName("MyServer");
mds.setPortNumber(21050);
mds.setDatabaseName("myDB");
mds.setUserName("JSmith@example.com");
mds.setPassword ("Password");
```

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 the minimum properties required to establish a connection:

```
{
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("authenticationmethod=" + jmds.getAuthenticationMethod());
System.out.println("username=" + jmds.getusername());
System.out.println("password=" + jmds.getpassword());
...
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. Then, the `DataSource.getConnection()` method is called to establish a connection.

Testing a data source connection

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

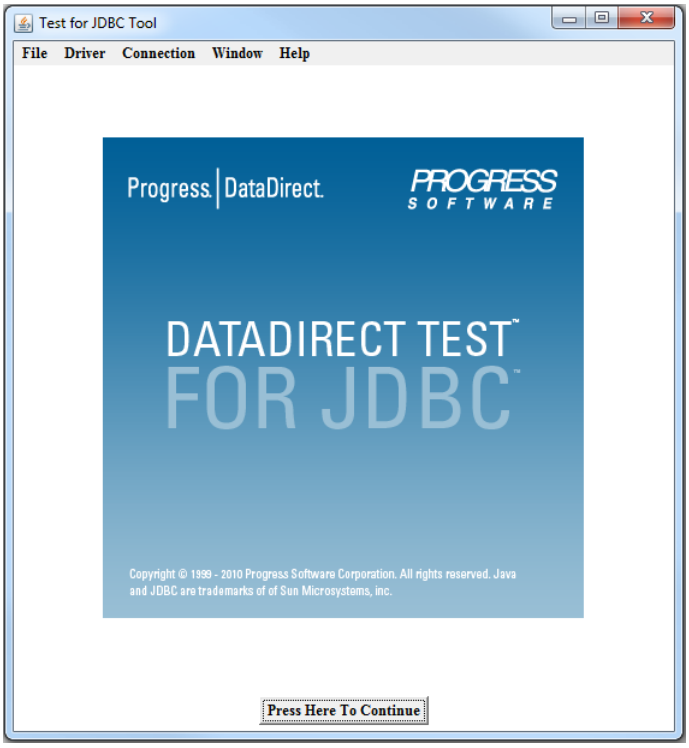
Take the following steps to establish a connection.

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

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

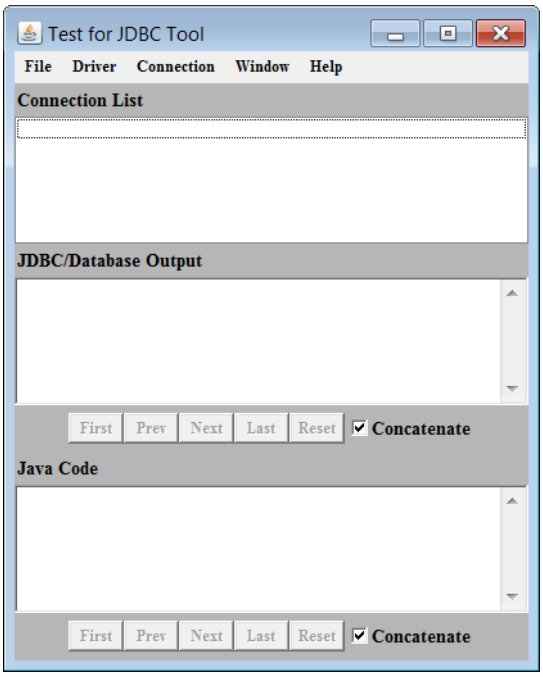
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

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



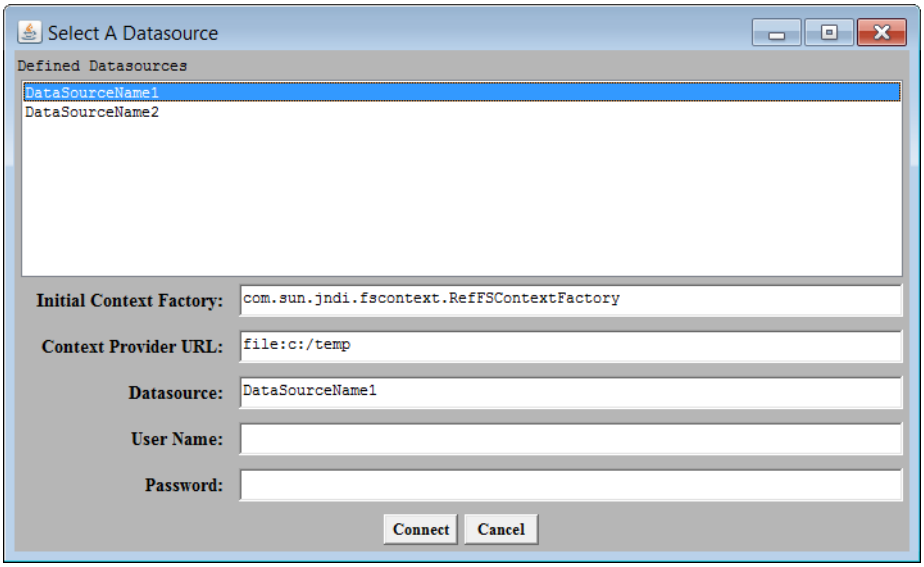
3. Click **Press Here to Continue**.

   The main dialog appears:



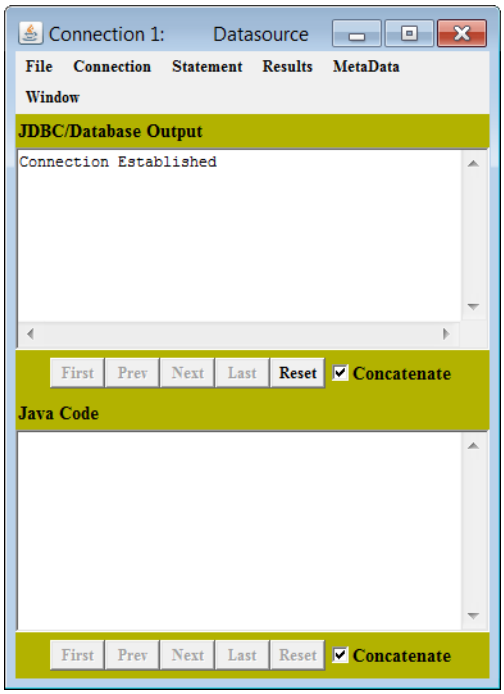
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   The **Select A Database** dialog appears:



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

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



Authentication

The driver supports the following authentication methods:

- *UserID/Password* authenticates using the user name and password provided by the LDAP server.
- *No Authentication* allows the user to connect without any authentication.
- *Kerberos* authenticates the user using 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.

By default, the driver is configured to use UserID/Password authentication via LDAP (`AuthenticationMethod=UserIDPassword`).

User ID/password authentication

Create your user credentials in the LDAP server.

To configure the driver to use user ID/password authentication via LDAP:

- 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.
  - Set the DatabaseName property to specify the database name.

- Set the User property to specify the user ID of the LDAP server.
- Set the Password property to specify the password of the LDAP server.

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

Connection URL:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:impala://myserver:21050;
  DatabaseName=dbname;User=Jsmith@example.com;Password=secret);
```

Note: The User ID and Password properties are not required to be stored in the connection string. They can also be passed separately by the application.

Data Source:

```
ImpalaDataSource mds = new ImpalaDataSource();
mds.setDescription("My Impala Data Source");
mds.setServerName("server_name");
mds.setPort("port");
mds.setDatabaseName("ImpalaDB");
mds.setUser("jsmith@example.com");
mds.setPassword("secret");
```

See also

AuthenticationMethod on page 64
Password on page 82
User on page 97
Connection property descriptions on page 55

No authentication

To configure the driver to use no (None) authentication.

- 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.
  - Set the DatabaseName property to specify the name of the database to which you are connecting.
- Set the AuthenticationMethod property to None.

The following examples demonstrate how to make a connection using the no authentication method.

Connection URL:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:impala://myserver:21050;DatabaseName=dbname);
```

Data Source:

```
ImpalaDataSource mds = new ImpalaDataSource();
mds.setDescription("My Impala Data Source");
mds.setServerName("server_name");
mds.setPort("port");
mds.setAuthenticationMethod("none");
mds.setDatabaseName("ImpalaDB");
```

See also

AuthenticationMethod on page 64
DatabaseName on page 70
ServerName on page 88
PortNumber on page 83

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. Optionally, if impersonation is enabled on the server, set ImpersonateUser property to provide your UserID used for impersonation.
8. Set the DatabaseName connection property as appropriate. (See "DatabaseName" for details.)

For example, the following is a connection URL with the required and optional properties for making a connection using Kerberos authentication.

Connection URL:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:impala://myserver:21050;AuthenticationMethod=kerberos;
  DatabaseName=ImpalaDB;ImpersonateUser=user2;
  ServicePrincipalName=impala/myserver@EXAMPLE.COM;");
```

Data Source:

```
ImpalaDataSource mds = new ImpalaDataSource();
mds.setDescription("My Impala Data Source");
mds.setServerName("server_name");
mds.setPort("port");
mds.setAuthenticationMethod("kerberos");
mds.setDatabaseName("testDB");
mds.setImpersonateUser("user2");
mds.setServicePrincipleName("impala/myserver@EXAMPLE.COM");
```

See also

AuthenticationMethod on page 64
ServicePrincipalName on page 89
ImpersonateUser on page 73
ServicePrincipalName on page 89
Kerberos authentication requirements on page 44
The JAAS login configuration file on page 44
LoginConfigName on page 79
DatabaseName on page 70

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 3: 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.

See also

Kerberos authentication on page 41

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

Kerberos authentication on page 41
LoginConfigName on page 79

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 64

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.

FIPS (Federal Information Processing Standard)

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

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

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

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

Performance considerations

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

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.

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

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

See also

ArrayFetchSize on page 63
InsensitiveResultSetBufferSize on page 75
MaxPooledStatements on page 81
StringDescribeType on page 93
UseCurrentSchema on page 96

4

Additional features and functionality

The following section describes additionally supported features and functionality that are specific to the driver.

For details, see the following topics:

- Parameter Metadata Support
- ResultSet Metadata Support
- Large Object Support
- Rowset Support
- Limitations on Apache Impala Functionality

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 Apache 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 statements and extensions on page 99

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.

Large Object Support

Although Apache 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.

Limitations on Apache Impala Functionality

Please note the following Apache 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 Apache Impala, refer to the Apache Impala user documentation:

https://impala.apache.org/docs/build/plain-html/index.html

Note: Apache 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.

5

Connection property descriptions

You can use connection properties to customize the driver for your environment. This section organizes connection properties according to functionality. You can use connection properties with either the JDBC `DriverManager` or a JDBC data source. For a `DriverManager` connection, a property is expressed as a key value pair and takes the form *`property=value`*. For a data source connection, a property is expressed as a JDBC method and takes the form `set`*`property`*`(`*`value`*`)`.

Note:

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

The following tables describe the connection properties by functionality.

- Data encryption properties
- Statement pooling properties
- Client information properties
- Additional properties

General properties

The following table summarizes the properties that are required for connection.

Property	Data Source Method	Default
AuthenticationMethod on page 64	`getAuthenticationMethod()` `setAuthenticationMethod(String)`	`UserIDPassword`
DatabaseName on page 70	`getDatabaseName()` `setDatabaseName(String)`	No default value
PortNumber on page 83	`getPortNumber()` `setPortNumber(Integer)`	`21050`
ServerName on page 88	`getServerName()` `setServerName(String)`	None

UserID/Password authentication properties

The following table summarizes properties used for UserID/Password authentication method.

Property	Data Source Method	Default
Password on page 82	`getPassword()` `setPassword(String)`	No default value
User on page 97	`getUser()` `setUser(String)`	No default value

Kerberos authentication properties

The following table summarizes properties used for Kerberos authentication method.

Property	Data Source Method	Default
ImpersonateUser on page 73	`getImpersonateUser()` `setImpersonateUser(String)`	No default value

Property	Data Source Method	Default
LoginConfigName on page 79	`getLoginConfigName()` `setLoginConfigName(String)`	`JDBC_DRIVER_01`
ServicePrincipalName on page 89	`getServicePrincipalName()` `setServicePrincipalName(String)`	None

Proxy server properties

The following table summarizes proxy server connection properties.

Property	Data Source Method	Default
ProxyHost on page 86	`getProxyHost()` `setProxyHost(String)`	No default value
ProxyPassword on page 84	`getProxyPassword()` `setProxyPassword(String)`	No default value
ProxyPort on page 85	`getProxyPort()` `setProxyPort(Integer)`	`0` which means the default is determined by the ProxyHost property. For HTTP URLs: `80` For HTTPS URLs: `443`
ProxyUser on page 85	`getProxyUser()` `setProxyUser(String)`	No default value

Data type properties

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

Table 4: Data Type Properties

Property	Data Source Method	Default
ConvertNull on page 68	`getConvertNull()` `setConvertNull(Boolean)`	`1`
JavaDoubleToString on page 76	`getJavaDoubleToString()` `setJavaDoubleToString(Boolean)`	`false`
StringDescribeType on page 93	`getStringDescribeType()` `setStringDescribeType(String)`	`varchar`

Data encryption properties

The following table summarizes data encryption connection properties.

Property	Data Source Method	Default
CryptoProtocolVersion on page 69	`getCryptoProtocolVersion()` `setCryptoProtocolVersion(String)`	The default is determined by the settings of the JRE.
EncryptionMethod on page 71	`getEncryptionMethod()` `setEncryptionMethod(String)`	`noEncryption`
HostNameInCertificate on page 72	`getHostNameInCertificate()` `setHostNameInCertificate(String)`	None
KeyPassword on page 77	`getKeyPassword()` `setKeyPassword(String)`	None
KeyStore on page 78	`getKeyStore()` `setKeyStore(String)`	None
KeyStorePassword on page 79	`getKeyStorePassword()` `setKeyStorePassword(String)`	None
TrustStore on page 94	`getTrustStore()` `setTrustStore(String)`	None
TrustStorePassword on page 95	`getTrustStorePassword()` `setTrustStorePassword(String)`	None
ValidateServerCertificate on page 97	`getValidateServerCertificate()` `setValidateServerCertificate(Boolean)`	`true`

Statement pooling properties

The following table summarizes statement pooling connection properties.

Table 5: Statement Pooling Properties

Property	Data Source Method	Default
ImportStatementPool on page 74	`getImportStatementPool()` `setImportStatementPool(String)`	No default value

Property	Data Source Method	Default
MaxPooledStatements on page 81	`getMaxPooledStatements()` `setMaxPooledStatements(Integer)`	`0`
RegisterStatementPoolMonitorMBean on page 87	`getRegisterStatementPoolMonitorMbean()` `setRegisterStatementPoolMonitorMbean(Boolean)`	`false`

Client information properties

The following table client information properties.

Property	Data Source Method	Default
AccountingInfo on page 62	`getAccountingInfo()` `setAccountingInfo(String)`	None
ApplicationName on page 62	`getApplicationName()` `setApplicationName(String)`	No default value
ClientHostName on page 65	`getClientHostName()` `setClientHostName(String)`	No default value
ClientUser on page 66	`getClientUser()` `setClientUser(String)`	No default value
ProgramID on page 83	`getProgramID()` `setProgramID(String)`	No default value

Additional properties

The following table summarizes additional connection properties.

Property	Data Source Method	Default
ArrayFetchSize on page 63	`getArrayFetchSize()` `setArrayFetchSize(Integer)`	`20000` (fields)
ConnectionRetryCount	`getConnectionRetryCount()` `setConnectionRetryCount(Integer)`	`5`
ConnectionRetryDelay	`getConnectionRetryDelay()` `setConnectionRetryDelay(Integer)`	`1`

Property	Data Source Method	Default
DefaultOrderByLimit on page 70	`setDefaultOrderByLimit` `setDefaultOrderByLimit(Integer)`	`-1`
InitializationString on page 75	`getInitializationString()` `setInitializationString(String)`	None
InsensitiveResultSetBufferSize on page 75	`getInsensitiveResultsetBufferSize()` `setInsensitiveResultsetBufferSize(Integer)`	`2048`
LoginTimeout on page 80	`getLoginTimeout()` `setLoginTimeout(Integer)`	`0`
RemoveColumnQualifiers on page 88	`getRemoveColumnQualifiers()` `setRemoveColumnQualifiers(Boolean)`	`false`
SpyAttributes on page 90	`getSpyAttributes()` `setSpyAttributes(String)`	None
TransactionMode	`getTransactionMode()` `setTransactionMode(String)`	`Full`
UseCurrentSchema on page 96	`getUseCurrentSchema()` `setUseCurrentSchema(Boolean)`	`false`

For details, see the following topics:

- AccountingInfo
- ApplicationName
- ArrayFetchSize
- AuthenticationMethod
- ClientHostName
- ClientUser
- ConnectionRetryCount
- ConnectionRetryDelay
- ConvertNull
- CryptoProtocolVersion
- DatabaseName
- DefaultOrderByLimit

- EncryptionMethod
- HostNameInCertificate
- ImpersonateUser
- ImportStatementPool
- InitializationString
- InsensitiveResultSetBufferSize
- JavaDoubleToString
- KeyPassword
- KeyStore
- KeyStorePassword
- LoginConfigName
- LoginTimeout
- MaxPooledStatements
- Password
- PortNumber
- ProgramID
- ProxyPassword
- ProxyPort
- ProxyUser
- ProxyHost
- 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

```
public String getAccountingInfo()
```

```
public void setAccountingInfo(String)
```

Default

Empty string

Data Type

String

See also

- For more information, refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference* .

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

```
public String getApplicationName()
```

```
public void setApplicationName(String)
```

Default

Empty string

Data Type

String

See also

- For more information, refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference* .

ArrayFetchSize

Purpose

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

Valid Values

-x | *x*

where:

-x

is a negative integer.

x

is a positive integer.

Behavior

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

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

Example

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

Notes

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

Data Source Method

```
public Integer getArrayFetchSize()
```

```
public void setArrayFetchSize(Integer)
```

Default

`20000` (fields)

Data Type

Int

See also

Performance considerations 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` | `none`

Behavior

If set to `userIdPassword`, the driver uses LDAP authentication through user ID and password configured in the LDAP server.

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

If set to `none`, the driver does not use authentication. Only the required properties must be specified for connection.

Notes

- The User property provides the user ID of the LDAP server. The Password property provides the password of the LDAP server.

Data Source Method

```
public String getAuthenticationMethod()
```

```
public void setAuthenticationMethod(String)
```

Default

`userIdPassword`

Data Type

String

See also

Authentication on page 39

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

`public String getClientHostName()`

`public void setClientHostName(String)`

Default

Empty string

Data Type

String

See also

- For more information, refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference* .

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

`public String getClientUser()`

`public void setClientUser(String)`

Default

Empty string

Data Type

String

See also

- For more information, refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference* .

ConnectionRetryCount

Purpose

The number of times the driver retries connection attempts to Impala until a successful connection is established.

Valid Values

`0` | *x*

where:

x

is a positive integer that represents the number of retries.

Behavior

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

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

Example

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

Notes

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

Data Source Methods

```
public Integer getConnectionRetryCount()
```

```
public void setConnectionRetryCount(Integer)
```

Default Value

`5`

Data Type

Integer

ConnectionRetryDelay

Purpose

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 Methods

```
public Integer getConnectionRetryDelay()
```

```
public void setConnectionRetryDelay(Integer)
```

Default Value

1

Data Type

Integer

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

```
public Integer getConvertNull()
```

```
public void setConvertNull(Integer)
```

Default

1

Data Type

Int

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

```
public String getCryptoProtocolVersion()
```

```
public void setCryptoProtocolVersion(String)
```

Default

The default is determined by the settings of the JRE.

Data Type

String

See also

- EncryptionMethod on page 71
- Data Encryption on page 46

DatabaseName

Purpose

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

Valid Values

database_name

where:

database_name

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

Data Source Methods

```
public String getDatabaseName()
```

```
public void setDatabaseName(String)
```

Default

The default database

Data Type

String

DefaultOrderByLimit

Purpose

Specifies the maximum number of rows returned when a SQL statement containing an ORDER BY clause is executed. Apache Impala 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 Apache Impala 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

```
public Integer getDefaultOrderByLimit()
```

```
public void setDefaultOrderByLimit(Integer)
```

Default

`-1` (Disabled)

Data Type

Int

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

```
public String getEncryptionMethod()
public void setEncryptionMethod(String)
```

Default

`noEncryption`

Data Type

String

See also

Data Encryption on page 46

Performance considerations on page 49

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

```
public String getHostNameInCertificate()
```

```
public void setHostNameInCertificate(String)
```

Default

Empty string

Data Type

String

See also

- EncryptionMethod on page 71
- Data Encryption on page 46

ImpersonateUser

Purpose

Specifies the user ID used for Impersonation. When Impersonation is enabled on the server (`impala.doas.user=`*`impersonating_userName`*), this value determines your identity and access rights to Impala resources when executing queries. If Impersonation is disabled, you will execute queries as the Impala user who initiated the process.

Refer to the Impala documentation for more information on using Impersonation with Kerberos.

Valid Values

`string`

where:

`string`

is a valid user ID with permissions to access the database.

Data Source Method

```
public String getImpersonateUser()
```

```
public void setImpersonateUser(String)
```

Default

None

Data Type

String

ImportStatementPool

Purpose

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

Valid Values

String

where:

String

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

Notes

- If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver throws an exception.
- For more information, refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public String getImportStatementPool()
```

```
public void setImportStatementPool(String)
```

Default Value

Empty string

Data Type

String

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

```
public String getInitializationString()
```

```
public void setInitializationString(String)
```

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 Methods

```
public Integer getInsensitiveResultsetBufferSize()
```

```
public void setInsensitiveResultsetBufferSize(Integer)
```

Default Value

`2048`

Data Type

Integer

See also

Performance considerations on page 49

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

```
public Boolean getJavaDoubleToString()
```

```
public void setJavaDoubleToString(Boolean)
```

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

```
public String getKeyPassword()
```

```
public void setKeyPassword(String)
```

Default

None

Data Type

String

See also

- EncryptionMethod on page 71
- Data Encryption on page 46

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

```
public String getKeyStore()
public void setKeyStore(String)
```

Default

None

Data Type

String

See also

- EncryptionMethod on page 71
- Data Encryption on page 46

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

`public String getKeyStorePassword()`

`public void setKeyStorePassword(String)`

Default

None

Data Type

String

See also

- EncryptionMethod on page 71
- Data Encryption on page 46

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;
};
```

Data Source Method

```
public String getLoginConfigName()
```

```
public void setLoginConfigName(String)
```

Default

JDBC_DRIVER_01

Data Type

String

See also

- Authentication on page 39
- The JAAS login configuration file on page 44

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

```
public Integer getLoginTimeout()
```

```
public void setLoginTimeout(Integer)
```

Default

`0`

Data Type

Integer

MaxPooledStatements

Purpose

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

Valid Values

`0` | *x*

where:

x

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

Behavior

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

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

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.

Notes

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

Data Source Methods

```
public Integer getMaxPooledStatements()
```

```
public void setMaxPooledStatements(Integer)
```

Default Value

`0`

Data Type

Integer

See also

Performance considerations on page 49

Password

Purpose

A password configured in the LDAP server that is used to connect to the Impala database.

Behavior

`password`

where:

`password`

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

Data Source Methods

```
public String getPassword()
```

```
public void setPassword(String)
```

Default Value

No default value

Data Type

String

See also

User on page 97

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

`public String getPortNumber()`

`public void setPortNumber(String)`

Default

`21050`

Data Type

Integer

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

```
public String getProgramID()
public void setProgramID(String)
```

Default

Empty string

Data Type

String

See also

- For more information, refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference* .

ProxyPassword

Purpose

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

Valid Values

password

where:

password

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

Data Source Methods

```
public String getProxyPassword()
public void setProxyPassword(String)
```

Default Value

No default value

Data Type

String

See also

ProxyPort on page 85
ProxyUser on page 85
ProxyHost on page 86

Connection URL examples on page 13

ProxyPort

Purpose

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

Valid Values

port

where:

port

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

Data Source Methods

```
public Integer getProxyPort()
```

```
public void setProxyPort(Integer)
```

Default Value

`0` which means that the default value is determined by whether the value specified for the ProxyHost property is an HTTP or HTTPS URL.

For HTTP: `80`

For HTTPS: `443`

Data Type

Integer

See also

ProxyPassword on page 84
ProxyUser on page 85
ProxyHost on page 86
Connection URL examples on page 13

ProxyUser

Purpose

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

Valid Values

user_name

where:

user_name

is a valid user ID for the proxy server.

Data Source Methods

```
public String getProxyUser()
public void setProxyUser(String)
```

Default Value

No default value

Data Type

String

See also

ProxyPassword on page 84
ProxyPort on page 85
ProxyHost on page 86
Connection URL examples on page 13

ProxyHost

Purpose

Identifies a proxy server to use for the first connection.

Valid Values

server_name | *IP_address*

where:

server_name

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

IP_address

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

Data Source Methods

```
public String getProxyHost()
public void setProxyHost(String)
```

Default Value

No default value

Data Type

String

See also

ProxyPassword on page 84
ProxyPort on page 85
ProxyUser on page 85
Connection URL examples on page 13

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.
- For more information, refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public Boolean getRegisterStatementPoolMonitorMbean()
public void setRegisterStatementPoolMonitorMbean(Boolean)
```

Default Value

`false`

Data Type

Boolean

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

```
public Boolean getRemoveColumnQualifiers()
```

```
public void setRemoveColumnQualifiers(Boolean)
```

Default

`false` (Disabled)

Data Type

Boolean

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.

The IP address can be specified in either IPv4 or IPv6 format, or a combination of the two.

Example

`122.23.15.12` or `MyImpalaServer`

Data Source Method

`public String getServerName()`

`public void setServerName(String)`

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

```
public String getServicePrincipalName()
```

```
public void setServicePrincipalName(String)
```

Default

None

Data Type

String

See also

- AuthenticationMethod on page 64
- Kerberos authentication on page 41

SpyAttributes

Purpose

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

Valid Values

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

where:

spy_attribute

is any valid DataDirect Spy attribute.

Behavior

Attribute	Description
`linelimit=`*`numberofchars`*	Sets the maximum number of characters that DataDirect Spy logs on a single line. The default is 0 (no maximum limit).
`log=(file)`*`filename`*	Directs logging to the file specified by *`filename`*. For Windows, if coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: `log=(file)C:\\temp\\spy.log;logIS=yes;logTName=yes`

`log=(filePrefix)`*`file_prefix`*	Directs logging to a file prefixed by *`file_prefix`*. The log file is named *`file_prefixX`*`.log`
	where:
	X is an integer that increments by 1 for each connection on which the prefix is specified.
	For example, if the attribute `log=(filePrefix)C:\\temp\\spy_` is specified on multiple connections, the following logs are created:
	`C:\temp\spy_1.log`
	`C:\temp\spy_2.log`
	`C:\temp\spy_3.log`
	`...`
	If coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash.
	For example:
	`log=(filePrefix)C:\\temp\\spy_;logIS=yes;logTName=yes`

`log=System.out`	Directs logging to the Java output standard, `System.out`.

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

Notes

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

Example

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

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

Data Source Method

```
public String getSpyAttributes()
```

```
public void setSpyAttributes(String)
```

Default

None

Data Type

String

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

```
public String getStringDescribeType()
```

```
public void setStringDescribeType(String)
```

Default

`varchar`

Data Type

String

See also

Performance considerations on page 49

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 Methods

```
public String getTransactionMode()
```

```
public void setTransactionMode(String)
```

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

```
public String getTrustStore()
```

```
public void setTrustStore(String)
```

Default

None

Data Type

String

See also

- EncryptionMethod on page 71
- Data Encryption on page 46

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

```
public String getTrustStorePassword()
```

```
public void setTrustStorePassword(String)
```

Default

None

Data Type

String

See also

- EncryptionMethod on page 71
- Data Encryption on page 46

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.

Data Source Methods

```
public Boolean getUseCurrentSchema()
```

```
public void setUseCurrentSchema(Boolean)
```

Default

`false`

Data Type

Boolean

See also

Performance considerations on page 49

User

Purpose

Specifies the user name configured in the LDAP server that is used to connect to the Impala database.

Valid Values

String

where:

String

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

Data Source Methods

`public String getUser()`

`public void setUser(String)`

Default Value

No default value

Data Type

String

See also

Password on page 82

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

```
public Boolean getValidateServerCertificate()
```

```
public void setValidateServerCertificate(Boolean)
```

Default

`true`

Data Type

Boolean

See also

- EncryptionMethod on page 71
- Data Encryption on page 46

6

Supported SQL statements and extensions

The driver provides support for the SQL statements and the SQL extensions described in this section. SQL extensions are denoted by an (EXT) in the topic title.

For details, see the following topics:

Alter Session (EXT)

Purpose

Changes various attributes of a local or remote session. A local session maintains the state of the overall connection. A remote session maintains the state that pertains to a particular remote data source connection.

Syntax

```
ALTER SESSION SET attribute_name=value
```

where:

attribute_name

specifies the name of the attribute to be changed. Attributes apply to either local or remote sessions.

value

specifies the value for that attribute.

The following table lists the local and remote session attributes, and provides descriptions of each.

Table 6: Alter Session Attributes

Attribute Name	Session Type	Description
Current_Schema	Local	Sets the current schema for the local session. The current schema is the schema used when an identifier in a SQL statement is unqualified. The string value must be the name of a schema visible in the local session. For example: `ALTER SESSION SET CURRENT_SCHEMA=IMPALA`
Stmt_Call_Limit	Local	Sets the maximum number of Web service calls the driver can make in executing a statement. Setting the Stmt_Call_Limit attribute has the same effect as setting the Statement Call Limit connection option. It sets the default Web service call limit used by any statement on the connection. Executing this command on a statement overrides the previously set Statement Call Limit for the connection. The value specified must be a positive integer or `0`. The value `0` means that no call limit exists. For example: `ALTER SESSION SET STMT_CALL_LIMIT=150`

Ws_Call_Count	Remote	Resets the Web service call count of a remote session to the value specified. The value must be `0` or a positive integer. WS_Call_Count represents the total number of Web service calls made to the remote data source instance for the current session. For example:
		`ALTER SESSION SET impala.WS_CALL_COUNT=0`
		The current value of WS_Call_Count can be obtained by referring to the System_Remote_Sessions system table (see SYSTEM_REMOTE_SESSIONS Catalog Table for details). For example:
		`SELECT * from information_schema.system_remote_sessions WHERE session_id = cursessionid()`

Delete

Purpose

The Delete statement is used to delete rows from a table.

Syntax

```
DELETE FROM table_name [WHERE search_condition]
```

where:

table_name

specifies the name of the table from which you want to delete rows.

search_condition

is an expression that identifies which rows to delete from the table.

Notes

- The Where clause determines which rows are to be deleted. Without a Where clause, all rows of the table are deleted, but the table is left intact. See "Where Clause" for information about the syntax of Where clauses. Where clauses can contain subqueries.

Example A

This example shows a Delete statement on the emp table.

```
DELETE FROM emp WHERE emp_id = 'E10001'
```

Each Delete statement removes every record that meets the conditions in the Where clause. In this case, every record having the employee ID `E10001` is deleted. Because employee IDs are unique in the employee table, at most, one record is deleted.

Example B

This example shows using a subquery in a Delete clause.

```
DELETE FROM emp WHERE dept_id = (SELECT dept_id FROM dept WHERE dept_name =
'Marketing')
```

The records of all employees who belong to the department named Marketing are deleted.

Notes

- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.

Explain Plan

Purpose

Retrieves a detailed list of the elements in the execution plan. It generates a result set with a single column named `OPERATION`. The individual elements that comprise the plan are returned as rows in the result set.

Syntax

```
EXPLAIN PLAN FOR {SELECT ...}
```

The returned list of elements includes the indexes used for performing the query and can be used to optimize the query.

Insert

Purpose

The Insert statement is used to add new rows to a local table. You can specify either of the following options:

- List of values to be inserted as a new row
- Select statement that copies data from another table to be inserted as a set of new rows

Syntax

```
INSERT INTO table_name [(column_name[,column_name]...)] {VALUES (expression
[,expression]...) | select_statement}
```

table_name

is the name of the table in which you want to insert rows.

column_name

is optional and specifies an existing column. Multiple column names (a column list) must be separated by commas. A column list provides the name and order of the columns, the values of which are specified in the Values clause. If you omit a *column_name* or a column list, the value expressions must provide values for all columns defined in the table and must be in the same order that the columns are defined for the table. Table columns that do not appear in the column list are populated with the default value, or with NULL if no default value is specified.

expression

is the list of expressions that provides the values for the columns of the new record. Typically, the expressions are constant values for the columns. Character string values must be enclosed in single quotation marks (’). See "Literals" for more information.

select_statement

is a query that returns values for each *column_name* value specified in the column list. Using a Select statement instead of a list of value expressions lets you select a set of rows from one table and insert it into another table using a single Insert statement. The Select statement is evaluated

before any values are inserted. This query cannot be made on the table into which values are inserted. See "Select" for information about Select statements.

Notes

- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.

Specifying an external ID column

Use the following syntax to specify an external ID column to look up the value of a foreign key column.

Syntax

column_name EXT_ID [*schema_name*.[*table_name*.]]*ext_id_column*

where:

EXT_ID

is used to specify that the column specified by *ext_id_column* is used to look up the rowid to be inserted into the column specified by *column_name*.

schema_name

is the name of the schema of the table that contains the foreign key column being specified as the external ID column.

table_name

is the name of the table that contains the foreign key column being specified as the external ID column.

ext_id_column

is the external ID column.

Example A

This example uses a list of expressions to insert records. Each Insert statement adds one record to the table. In this case, one record is added to the table emp. Values are specified for five columns. The remaining columns in the table are assigned the default value or NULL if no default value is specified.

```
INSERT INTO emp (last_name,
                 first_name,
                 emp_id,
                 salary,
                 hire_date)
VALUES ('Smith', 'John', 'E22345', 27500, {1999-04-06})
```

Example B

This example uses a Select statement to insert records. The number of columns in the result of the Select statement must match exactly the number of columns in the table if no column list is specified, or it must match the number of column names specified in the column list. A new entry is created in the table for every row of the Select result.

```
INSERT INTO emp1 (first_name,
                  last_name,
```

```
                  emp_id,
                  dept,
                  salary)
SELECT first_name, last_name, emp_id, dept, salary FROM emp
WHERE dept = 'D050'
```

Example C

This example uses a list of expressions to insert records and specifies an external ID column (a foreign key column) named `accountId` that references a table that has an external ID column named `AccountNum`.

```
INSERT INTO emp (last_name,
                    first_name,
                    emp_id,
                    salary,
                    hire_date,
                    accountId EXT_ID AccountNum)
VALUES ('Smith', 'John', 'E22345', 27500, {1999-04-06}, 0001)
```

Select

Purpose

Use the Select statement to fetch results from one or more tables.

Syntax

```
SELECT select_clause from_clause
[where_clause]
[groupby_clause]
[having_clause]
[{UNION [ALL | DISTINCT] |
 {MINUS [DISTINCT] | EXCEPT [DISTINCT]} |
 INTERSECT [DISTINCT]} select_statement]
[limit_clause]
```

where:

select_clause

specifies the columns from which results are to be returned by the query. See "Select clause" for a complete explanation.

from_clause

specifies one or more tables on which the other clauses in the query operate. See "From clause" for a complete explanation.

where_clause

is optional and restricts the results that are returned by the query. See "Where clause" for a complete explanation.

groupby_clause

is optional and allows query results to be aggregated in terms of groups. See "Group By clause" for a complete explanation.

having_clause

is optional and specifies conditions for groups of rows (for example, display only the departments that have salaries totaling more than $200,000). See "Having clause" for a complete explanation.

UNION

is an optional operator that combines the results of the left and right Select statements into a single result. See "Union operator" for a complete explanation.

INTERSECT

is an optional operator that returns a single result by keeping any distinct values from the results of the left and right Select statements. See "Intersect operator" for a complete explanation.

EXCEPT | MINUS

are synonymous optional operators that returns a single result by taking the results of the left Select statement and removing the results of the right Select statement. See "Except and Minus operators" for a complete explanation.

orderby_clause

is optional and sorts the results that are returned by the query. See "Order By clause" for a complete explanation.

limit_clause

is optional and places an upper bound on the number of rows returned in the result. See "Limit clause" for a complete explanation.

Select clause

Purpose

Use the Select clause to specify with a list of column expressions that identify columns of values that you want to retrieve or an asterisk (*) to retrieve the value of all columns.

Syntax

```
SELECT [{LIMIT offsetnumber | TOP number}] [ALL | DISTINCT] {* | column_expression
[[AS] column_alias] [,column_expression [[AS] column_alias], ...]}
```

where:

LIMIT *offset number*

creates the result set for the Select statement first and then discards the first number of rows specified by *offset* and returns the number of remaining rows specified by *number*. To not discard any of the rows, specify 0 for *offset*, for example, LIMIT 0 *number*. To discard the first *offset* number of rows and return all the remaining rows, specify 0 for *number*, for example, LIMIT *offset*0.

`TOP number`

is equivalent to `LIMIT 0number`.

column_expression

can be simply a column name (for example, `last_name`). More complex expressions may include mathematical operations or string manipulation (for example, `salary * 1.05`). See "SQL expressions" for details. *column_expression* can also include aggregate functions. See "Aggregate functions" for details.

column_alias

can be used to give the column a descriptive name. For example, to assign the alias department to the column dep:

```
SELECT dep AS department FROM emp
```

`DISTINCT`

eliminates duplicate rows from the result of a query. This operator can precede the first column expression. For example:

```
SELECT DISTINCT dep FROM emp
```

Notes

- Separate multiple column expressions with commas (for example, `SELECT last_name, first_name, hire_date`).
- Column names can be prefixed with the table name or table alias. For example, `SELECT emp.last_name` or `e.last_name`, where `e` is the alias for the table `emp`.
- NULL values are not treated as distinct from each other. The default behavior is that all result rows be returned, which can be made explicit with the keyword `ALL`.

See also

SQL expressions on page 117

Aggregate functions

Aggregate functions can also be a part of a Select clause. Aggregate functions return a single value from a set of rows. An aggregate can be used with a column name (for example, `AVG(salary)`) or in combination with a more complex column expression (for example, `AVG(salary * 1.07)`).

The following table lists supported aggregate functions.

Note: Doubly nested aggregates, such as `SUM(COUNT(col1))`, are currently not permitted by the driver.

Table 7: Aggregate Functions

Aggregate	Returns
AVG	The average of the values in a numeric column expression. For example, `AVG(salary)` returns the average of all salary column values.

COUNT	The number of values in any field expression. For example, `COUNT(name)` returns the number of name values. When using `COUNT` with a field name, `COUNT` returns the number of non-NULL column values. A special example is `COUNT(*)`, which returns the number of rows in the set, including rows with NULL values. **Note:** The driver does not support `COUNT(DISTINCT *)`. For example, `SELECT COUNT(DISTINCT *) FROM mytable` results in a syntax error.
MAX	The maximum value in any column expression. For example, `MAX(salary)` returns the maximum salary column value.
MIN	The minimum value in any column expression. For example, `MIN(salary)` returns the minimum salary column value.
SUM	The total of the values in a numeric column expression. For example, `SUM(salary)` returns the sum of all salary column values.

Example

The following example uses the `COUNT`, `MAX`, and `AVG` aggregate functions:

```
SELECT
     COUNT(amount) AS numOpportunities,
     MAX(amount) AS maxAmount,
     AVG(amount) AS avgAmount
FROM opportunity o INNER JOIN user u
     ON o.ownerId = u.id
WHERE o.isClosed = 'false' AND
     u.name = 'MyName'
```

From clause

Purpose

The From clause indicates the tables to be used in the Select statement.

Syntax

`FROM` *`table_name`* `[`*`table_alias`*`] [,...]`

where:

`table_name`

is the name of a table or a subquery. Multiple tables define an implicit inner join among those tables. Multiple table names must be separated by a comma. For example:

```
SELECT * FROM emp, dep
```

Subqueries can be used instead of table names. Subqueries must be enclosed in parentheses. See "Subquery in a From clause" for an example.

table_alias

is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias.

Example

The following example specifies two table aliases, e for emp and d for dep:

```
SELECT e.name, d.deptName
FROM emp e, dep d
WHERE e.deptId = d.id
```

table_alias is a name used to refer to a table in the rest of the Select statement. When you specify an alias for a table, you can prefix all column names of that table with the table alias. For example, given the table specification:

```
FROM emp E
```

you may refer to the last_name field as E.last_name. Table aliases must be used if the Select statement joins a table to itself. For example:

```
SELECT * FROM emp E, emp F WHERE E.mgr_id = F.emp_id
```

The equal sign (=) includes only matching rows in the results.

Join in a From clause

Purpose

You can use a Join as a way to associate multiple tables within a Select statement. Joins may be either explicit or implicit. For example, the following is the example from the previous section restated as an explicit inner join:

```
SELECT * FROM emp INNER JOIN dep ON id=empId
SELECT e.name, d.deptName
FROM emp e INNER JOIN dep d ON e.deptId = d.id;
```

whereas the following is the same statement as an implicit inner join:

```
SELECT * FROM emp, dep WHERE emp.deptID=dep.id
```

Note: The ON clause in a join expression must evaluate to a true or false value.

Syntax

```
FROM table_name {RIGHT OUTER | INNER | LEFT OUTER | CROSS | FULL OUTER} JOIN table.key
 ON search-condition
```

Example

In this example, two tables are joined using LEFT OUTER JOIN. T1, the first table named includes nonmatching rows.

```
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.key = T2.key
```

If you use a CROSS JOIN, no ON expression is allowed for the join.

Subquery in a From clause

Subqueries can be used in the From clause in place of table references (*table_name*).

Example

```
SELECT * FROM (SELECT * FROM emp WHERE sal > 10000) new_emp, dept WHERE
new_emp.deptno = dept.deptno
```

See also

Subqueries on page 115

Where clause

Purpose

Specifies the conditions that rows must meet to be retrieved.

Syntax

```
WHERE expr1 rel_operator expr2
```

where:

expr1

is either a column name, literal, or expression.

expr2

is either a column name, literal, expression, or subquery. Subqueries must be enclosed in parentheses.

rel_operator

is the relational operator that links the two expressions.

Example

The following Select statement retrieves the first and last names of employees that make at least $20,000.

```
SELECT last_name, first_name FROM emp WHERE salary >= 20000
```

See also

SQL expressions on page 117
Subqueries on page 115

Group By clause

Purpose

Specifies the names of one or more columns by which the returned values are grouped. This clause is used to return a set of aggregate values.

Syntax

```
GROUP BY column_expression [,...]
```

where:

column_expression

is either a column name or a SQL expression. Multiple values must be separated by a comma. If *column_expression* is a column name, it must match one of the column names specified in the Select clause. Also, the Group By clause must include all non-aggregate columns specified in the Select list.

Example

The following example totals the salaries in each department:

```
SELECT dept_id, sum(salary) FROM emp GROUP BY dept_id
```

This statement returns one row for each distinct department ID. Each row contains the department ID and the sum of the salaries of the employees in the department.

See also

SQL expressions on page 117
Subqueries on page 115

Having clause

Purpose

Specifies conditions for groups of rows (for example, display only the departments that have salaries totaling more than $200,000). This clause is valid only if you have already defined a Group By clause.

Syntax

```
HAVING expr1 rel_operator expr2
```

where:

expr1 | expr2

can be column names, constant values, or expressions. These expressions do not have to match a column expression in the Select clause. See "SQL expressions" for details regarding SQL expressions.

rel_operator

is the relational operator that links the two expressions.

Example

The following example returns only the departments that have salaries totaling more than $200,000:

```
SELECT dept_id, sum(salary) FROM emp GROUP BY dept_id HAVING sum(salary) > 200000
```

See also

SQL expressions on page 117
Subqueries on page 115

Union operator

Purpose

Combines the results of two Select statements into a single result. The single result is all the returned rows from both Select statements. By default, duplicate rows are not returned. To return duplicate rows, use the All keyword (`UNION ALL`).

Syntax

```
select_statement
UNION [ALL | DISTINCT] | {MINUS [DISTINCT] | EXCEPT [DISTINCT]} | INTERSECT
[DISTINCT]select_statement
```

Notes

- When using the Union operator, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp
UNION
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp
UNION
SELECT salary, last_name FROM raises
```

Intersect operator

Purpose

Intersect operator returns a single result set. The result set contains rows that are returned by both Select statements. Duplicates are returned unless the Distinct operator is added.

Syntax

```
select_statement
INTERSECT [DISTINCT]
select_statement
```

where:

`DISTINCT`

eliminates duplicate rows from the results.

Notes

- When using the Intersect operator, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp
INTERSECT [DISTINCT]
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp
INTERSECT
SELECT salary, last_name FROM raises
```

Except and Minus operators

Purpose

Return the rows from the left Select statement that are not included in the result of the right Select statement.

Syntax

```
select_statement
{EXCEPT [DISTINCT] | MINUS [DISTINCT]}
select_statement
```

where:

`DISTINCT`

eliminates duplicate rows from the results.

Notes

- When using one of these operators, the Select lists for each Select statement must have the same number of column expressions with the same data types and must be specified in the same order.

Example A

The following example has the same number of column expressions, and each column expression, in order, has the same data type.

```
SELECT last_name, salary, hire_date FROM emp
EXCEPT
SELECT name, pay, birth_date FROM person
```

Example B

The following example is *not* valid because the data types of the column expressions are different (`salary FROM emp` has a different data type than `last_name FROM raises`). This example does have the same number of column expressions in each Select statement but the expressions are not in the same order by data type.

```
SELECT last_name, salary FROM emp
EXCEPT
SELECT salary, last_name FROM raises
```

Order By clause

Purpose

The Order By clause specifies how the rows are to be sorted.

Syntax

`ORDER BY` *`sort_expression`* `[DESC | ASC] [,...]`

where:

`sort_expression`

is either the name of a column, a column alias, a SQL expression, or the positioned number of the column or expression in the select list to use.

The default is to perform an ascending (ASC) sort.

Example

To sort by `last_name` and then by `first_name`, you could use either of the following Select statements:

```
SELECT emp_id, last_name, first_name FROM emp
ORDER BY last_name, first_name
```

or

```
SELECT emp_id, last_name, first_name FROM emp
ORDER BY 2,3
```

In the second example, `last_name` is the second item in the Select list, so `ORDER BY 2,3` sorts by `last_name` and then by `first_name`.

See also

SQL expressions on page 117

Limit clause

Purpose

Places an upper bound on the number of rows returned in the result.

Syntax

`LIMIT` *`number_of_rows`* `[OFFSET` *`offset_number`*`]`

where:

`number_of_rows`

specifies a maximum number of rows in the result. A negative number indicates no upper bound.

`OFFSET`

specifies how many rows to skip at the beginning of the result set. *`offset_number`* is the number of rows to skip.

Notes

- In a compound query, the Limit clause can appear only on the final Select statement. The limit is applied to the entire query, not to the individual Select statement to which it is attached.

Example

The following example returns a maximum of 20 rows.

```
SELECT last_name, first_name FROM emp WHERE salary > 20000 ORDER BY dept_idc LIMIT
20
```

Update

Purpose

An Update statement changes the value of columns in the selected rows of a table.

Syntax

`UPDATE` *`table_name`* `SET` *`column_name = expression`*

`[,` *`column_name = expression`*`] [WHERE` *`conditions`*`]`

`table_name`

is the name of the table for which you want to update values.

`column_name`

is the name of a column, the value of which is to be changed. Multiple column values can be changed in a single statement.

expression

is the new value for the column. The expression can be a constant value or a subquery that returns a single value. Subqueries must be enclosed in parentheses.

Example A

The following example changes every record that meets the conditions in the Where clause. In this case, the salary and exempt status are changed for all employees having the employee ID `E10001`. Because employee IDs are unique in the emp table, only one record is updated.

```
UPDATE emp SET salary=32000, exempt=1
WHERE emp_id = 'E10001'
```

Example B

The following example uses a subquery. In this example, the salary is changed to the average salary in the company for the employee having employee ID `E10001`.

```
UPDATE emp SET salary = (SELECT avg(salary) FROM emp)
WHERE emp_id = 'E10001'
```

Notes

- To enable Insert, Update, and Delete, set the ReadOnly connection option to `false`.
- A Where clause can be used to restrict which rows are updated.

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.

A subquery is a query expression that appears in the body of another expression such as a Select, an Update, or a Delete statement. In the following example, the second Select statement is a subquery:

```
SELECT * FROM emp WHERE deptno IN (SELECT deptno FROM dept)
```

IN predicate

Purpose

The In predicate specifies a set of values against which to compare a result set. If the values are being compared against a subquery, only a single column result set is returned.

Syntax

value [NOT] IN (*value1, value2,...*)

OR

value [NOT] IN (*subquery*)

Example

```
SELECT * FROM emp WHERE deptno IN
```

```
(SELECT deptno FROM dept WHERE dname <> 'Sales')
```

EXISTS predicate

Purpose

The Exists predicate is true only if the cardinality of the subquery is greater than 0; otherwise, it is false.

Syntax

```
EXISTS (subquery)
```

Example

```
SELECT empno, ename, deptno FROM emp e WHERE EXISTS
(SELECT deptno FROM dept WHERE e.deptno = dept.deptno)
```

UNIQUE predicate

Purpose

The Unique predicate is used to determine whether duplicate rows exist in a virtual table (one returned from a subquery).

Syntax

```
UNIQUE (subquery)
```

Example

```
SELECT * FROM dept d WHERE UNIQUE
(SELECT deptno FROM emp e WHERE e.deptno = d.deptno)
```

Correlated subqueries

Purpose

A correlated subquery is a subquery that references a column from a table referred to in the parent statement. A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a Select, Update, or Delete statement.

A correlated subquery answers a multiple-part question in which the answer depends on the value in each row processed by the parent statement. For example, you can use a correlated subquery to determine which employees earn more than the average salaries for their departments. In this case, the correlated subquery specifically computes the average salary for each department.

Syntax

```
SELECT select_list
   FROM table1 t_alias1
   WHERE expr rel_operator
```

```
   (SELECT column_list
       FROM table2 t_alias2
       WHERE t_alias1.columnrel_operatort_alias2.column)
UPDATE table1 t_alias1
   SET column =
   (SELECT expr
       FROM table2 t_alias2
        WHERE t_alias1.column = t_alias2.column)
DELETE FROM table1 t_alias1
   WHERE column rel_operator
   (SELECT expr
       FROM table2 t_alias2
       WHERE t_alias1.column = t_alias2.column)
```

Notes

- Correlated column names in correlated subqueries must be explicitly qualified with the table name of the parent.

Example A

The following statement returns data about employees whose salaries exceed their department average. This statement assigns an alias to emp, the table containing the salary information, and then uses the alias in a correlated subquery:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
  ORDER BY deptno
```

Example B

This is an example of a correlated subquery that returns row values:

```
SELECT * FROM dept "outer" WHERE 'manager' IN
  (SELECT managername FROM emp
  WHERE "outer".deptno = emp.deptno)
```

Example C

This is an example of finding the department number (deptno) with multiple employees:

```
SELECT * FROM dept main WHERE 1 <
  (SELECT COUNT(*) FROM emp WHERE deptno = main.deptno)
```

Example D

This is an example of correlating a table with itself:

```
SELECT deptno, ename, sal FROM emp x WHERE sal >
  (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
```

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 of Select statements; and in the Set clauses of Update statements.

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

The driver supports both unquoted and quoted identifiers. An unquoted identifier must start with an ASCII alpha character and can be followed by zero

Quoted identifiers must be enclosed in double quotation marks (""). A quoted identifier can contain any Unicode character including the space character. The driver recognizes the Unicode escape sequence \uxxxx as a Unicode character. You can specify a double quotation mark in a quoted identifier by escaping it with a double quotation mark.

The maximum length of both quoted and unquoted identifiers is 128 characters.

Valid expression elements are:

- Column names
- Literals
- Operators
- Functions

Column names

The most common expression is a simple column name. You can combine a column name with other expression elements.

Literals

Literals are fixed data values. For example, in the expression `PRICE * 1.05`, the value 1.05 is a constant. Literals are classified into types, including the following:

- Binary
- Character string
- Date
- Floating point
- Integer
- Numeric
- Time
- Timestamp

The following table describes the literal format for supported SQL data types.

Table 8: Literal Syntax Examples

SQL Type	Literal Syntax	Example
BIGINT	*n* where *n* is any valid integer value in the range of the INTEGER data type	`12 or -34 or 0`

SQL Type	Literal Syntax	Example
BOOLEAN	Min Value: 0 Max Value: 1	`0` `1`
DATE	DATE'*date*'	`'2010-05-21'`
DATETIME	TIMESTAMP'*ts*'	`'2010-05-21 18:33:05.025'`

DECIMAL	*n.f*
	where:
	n
	is the integral part
	f
	is the fractional part
		`0.25`
		`3.1415`
		`-7.48`

DOUBLE	*n.f*E*x* where: *n* is the integral part *f* is the fractional part *x* is the exponent	`1.2E0` or `2.5E40` or `-3.45E2` or `5.67E-4`
INTEGER	*n* where *n* is a valid integer value in the range of the INTEGER data type	`12` or `-34` or `0`
LONGVARBINARY	'*hex_value*'	`'000482ff'`
LONGVARCHAR	'*value*'	`'This is a string literal'`
TIME	TIME'*time*'	`'2010-05-21 18:33:05.025'`
VARCHAR	'*value*'	`'This is a string literal'`

Character string literals

Text specifies a character string literal. A character string literal must be enclosed in single quotation marks. To represent one single quotation mark within a literal, you must enter two single quotation marks. When the data in the fields is returned to the client, trailing blanks are stripped.

A character string literal can have a maximum length of 32 KB, that is, (32*1024) bytes.

Example

```
'Hello'
'Jim''s friend is Joe'
```

Numeric literals

Unquoted numeric values are treated as numeric literals. If the unquoted numeric value contains a decimal point or exponent, it is treated as a real literal; otherwise, it is treated as an integer literal.

Example

```
+1894.1204
```

Binary literals

Binary literals are represented with single quotation marks. The valid characters in a binary literal are 0-9, a-f, and A-F.

Example

```
'00af123d'
```

Date/Time literals

Date and time literal values are enclosed in single quotion marks ('*value*').

- The format for a Date literal is DATE'*date*'.
- The format for a Time literal is TIME'*time*'.
- The format for a Timestamp literal is TIMESTAMP'*ts*'.

Integer literals

Integer literals are represented by a string of numbers that are not enclosed in quotation marks and do not contain decimal points.

Notes

- Integer constants must be whole numbers; they cannot contain decimals.
- Integer literals can start with sign characters (+/-).

Example

```
1994 or -2
```

Operators

This section describes the operators that can be used in SQL expressions.

Note: Numeric operators are restricted to numeric types. Numeric operators do not support non-numeric types.

Unary operator

A unary operator operates on only one operand.

operator operand

Binary operator

A binary operator operates on two operands.

operand1 operator operand2

If an operator is given a null operand, the result is always null. The only operator that does not follow this rule is concatenation (||).

Arithmetic operators

You can use an arithmetic 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 9: Arithmetic Operators

Operator	Purpose	Example
+ -	Denotes a positive or negative expression. These are unary operators.	`SELECT * FROM emp WHERE comm = -1`
* /	Multiplies, divides. These are binary operators.	`UPDATE emp SET sal = sal + sal * 0.10`
+ -	Adds, subtracts. These are binary operators.	`SELECT sal + comm FROM emp WHERE empno > 100`

Concatenation operator

The concatenation operator manipulates character strings. The following table lists the only supported concatenation operator.

Table 10: Concatenation Operator

Operator	Purpose	Example
||	Concatenates character strings.	`SELECT 'Name is' || ename FROM emp`

The result of concatenating two character strings is the data type VARCHAR.

Comparison operators

Comparison operators compare one expression to another. The result of such a comparison can be TRUE, FALSE, or UNKNOWN (if one of the operands is NULL). The driver considers the UNKNOWN result as FALSE.

The following table lists the supported comparison operators.

Table 11: Comparison Operators

Operator	Purpose	Example
=	Equality test.	`SELECT * FROM emp WHERE sal = 1500`
!=<>	Inequality test.	`SELECT * FROM emp WHERE sal != 1500`
><	"Greater than" and "less than" tests.	`SELECT * FROM emp WHERE sal > 1500 SELECT * FROM emp WHERE sal < 1500`
>=<=	"Greater than or equal to" and "less than or equal to" tests.	`SELECT * FROM emp WHERE sal >= 1500 SELECT * FROM emp WHERE sal <= 1500`
LIKE	% and _ wildcards can be used to search for a pattern in a column. The percent sign denotes zero, one, or multiple characters, while the underscore denotes a single character. The right-hand side of a LIKE expression must evaluate to a string or binary.	`SELECT * FROM emp WHERE ENAME LIKE 'J%'`

ESCAPE clause in LIKE operator
LIKE 'pattern string' ESCAPE 'c'
	The Escape clause is supported in the LIKE predicate to indicate the escape character. Escape characters are used in the pattern string to indicate that any wildcard character that is after the escape character in the pattern string should be treated as a regular character.
	The default escape character is backslash (\).
		`SELECT * FROM emp WHERE ENAME LIKE 'J%\_%' ESCAPE '\'`
		This matches all records with names that start with letter 'J' and have the '_' character in them.
		`SELECT * FROM emp WHERE ENAME LIKE 'JOE\_JOHN' ESCAPE '\'`
		This matches only records with name 'JOE_JOHN'.

[NOT] IN	"Equal to any member of" test.	`SELECT * FROM emp WHERE job IN ('CLERK','ANALYST') SELECT * FROM emp WHERE sal IN (SELECT sal FROM emp WHERE deptno = 30)`
[NOT] BETWEEN x AND y	"Greater than or equal to x" and "less than or equal to y."	`SELECT * FROM emp WHERE sal BETWEEN 2000 AND 3000`

Operator	Purpose	Example
EXISTS	Tests for existence of rows in a subquery.	`SELECT empno, ename, deptno FROM emp e WHERE EXISTS (SELECT deptno FROM dept WHERE e.deptno = dept.deptno)`
IS [NOT] NULL	Tests whether the value of the column or expression is NULL.	`SELECT * FROM emp WHERE ename IS NOT NULL SELECT * FROM emp WHERE ename IS NULL`

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 12: Logical Operators

Operator	Purpose	Example
NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN.	`SELECT * FROM emp WHERE NOT (job IS NULL)` `SELECT * FROM emp WHERE NOT (sal BETWEEN 1000 AND 2000)`
AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE; otherwise, returns UNKNOWN.	`SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10`
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE; otherwise, returns UNKNOWN.	`SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10`

Example

In the Where clause of the following Select statement, the AND logical operator is used to ensure that managers earning more than $1000 a month are returned in the result:

```
SELECT * FROM emp WHERE jobtitle = manager AND sal > 1000
```

Operator precedence

As expressions become more complex, the order in which the expressions are evaluated becomes important. The following table shows the order in which the operators are evaluated. The operators in the first line are evaluated first, then those in the second line, and so on. Operators in the same line are evaluated left to right in the expression.You can change the order of precedence by using parentheses. Enclosing expressions in parentheses forces them to be evaluated together.

Table 13: Operator Precedence

Precedence	Operator
1	+ (Positive), - (Negative)
2	*(Multiply), / (Division)
3	+ (Add), - (Subtract)
4	|| (Concatenate)
5	=, >, <, >=, <=, <>, != (Comparison operators)
6	NOT, IN, LIKE
7	AND
8	OR

Example A

The query in the following example returns employee records for which the department number is 1 or 2 and the salary is greater than $1000:

```
SELECT * FROM emp WHERE (deptno = 1 OR deptno = 2) AND sal > 1000
```

Because parenthetical expressions are forced to be evaluated first, the OR operation takes precedence over AND.

Example B

In the following example, the query returns records for all the employees in department 1, but only employees whose salary is greater than $1000 in department 2.

```
SELECT * FROM emp WHERE deptno = 1 OR deptno = 2 AND sal > 1000
```

The AND operator takes precedence over OR, so that the search condition in the example is equivalent to the expression `deptno = 1 OR (deptno = 2 AND sal > 1000)`.

Functions

The driver supports a number of functions that you can use in expressions, including String, Numeric, Timedate, and System functions.

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

Conditions

A condition specifies a combination of one or more expressions and logical operators that evaluates to either TRUE, FALSE, or UNKNOWN. You can use a condition in the Where clause of the Delete, Select, and Update statements; and in the Having clauses of Select statements. The following describes supported conditions.

Table 14: Conditions

Condition	Description
Simple comparison	Specifies a comparison with expressions or subquery results. `= , !=, <>, < , >, <=, <=`
Group comparison	Specifies a comparison with any or all members in a list or subquery. `[= , !=, <>, < , >, <=, <=] [ANY, ALL, SOME]`
Membership	Tests for membership in a list or subquery. `[NOT] IN`
Range	Tests for inclusion in a range. `[NOT] BETWEEN`
NULL	Tests for nulls. `IS NULL, IS NOT NULL`
EXISTS	Tests for existence of rows in a subquery. `[NOT] EXISTS`
LIKE	Specifies a test involving pattern matching. `[NOT] LIKE`
Compound	Specifies a combination of other conditions. `CONDITION [AND/OR] CONDITION`