Progress DataDirect for JDBC for Snowflake

User's Guide

Release 6.0.1

Copyright

Table of Contents

1

Welcome to the Progress DataDirect for JDBC for Snowflake Driver

The Progress® DataDirect® for JDBC™ for Snowflake™ driver supports SQL read-write access for JDBC applications to Snowflake. The driver provides access to data in Snowflake 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 Snowflake. 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 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-snowflake.

For details, see the following topics:

- 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 for 6.0.1

- **Driver Enhancements**
  - Corrections have been made to the contents of the getTypeInfo result set. For details, see Data types on page 23 and getTypeInfo() on page 24.
  - 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 62 for details.
  - The driver has been enhanced to support key-pair authentication. When key-pair authentication is enabled (`AuthenticationMethod=KeyPair`), you can authenticate to Snowflake using a pair of private and public keys. To configure the driver to use key-pair authentication, you can use the new PrivateKeyContent on page 92, PrivateKeyFile on page 93, and PrivateKeyPassphrase on page 94 connection properties. For details, see Key-pair authentication on page 61.
  - The driver now allows you to specify whether the driver fetches metadata for only tables in the database to which you are connected when a database name is not specified in metadata calls. When enabled, this behavior can provide better performance for metadata calls by reducing the number of tables queried. You can use the new UseSessionDatabaseForMetadata on page 107 connection property to configure the driver's behavior.
  - The driver has been modified to map Snowflake fixed-point number types where precision and scale cannot be modified to the Numeric data type by default. In addition, the IntegerFieldMapping connection property has been added to the driver. This connection property allows you to map these fixed-point number types to BigInt. For details, see IntegerFieldMapping on page 87 and Data types on page 23.
  - The driver has been enhanced to support the refresh token grant for OAuth 2.0 authentication. To support it, a new connection property, RefreshToken on page 101, has been added. See Refresh token grant on page 59 for details.
  - The driver has been enhanced to fall back to JSON query format when the arrow format is not properly initialized. The driver uses a high-speed arrow transfer that requires the restricted APIs from java.nio package. When the JVM is not correctly configured for Java SE 16 and higher, an exception is returned when the driver executes a query. To allow you to continue executing queries in this scenario, you can configure the new ArrowFallbackToJson property to switch to the JSON query format. See ArrowFallbackToJson on page 76 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 104 for details.
  - The TransactionMode property has been removed from the driver as of version 6.0.1. It was determined that TransactionMode provides no value with respect to Snowflake connections. If you previously used TransactionMode with version 6.0.0, you must remove this setting after upgrading to 6.0.1. Continuing to use it may result in an error and prevent the driver from establishing a connection. In version 6.0.1, all transaction handling is delegated to the server.
  - The installer program now requires you to install a JRE that is Java SE 11 or higher before running the installer. In earlier versions, the JRE used by the installer program was included in the product. However, to avoid potential security vulnerabilities, the installer program no longer includes a JRE. Instead, the installer program uses the JRE in your environment to allow for the most secure version of a JRE to be used.

    Note: This change does not affect the JVM requirements for the driver. For the latest driver requirements, refer to the Product Compatibility Guide:

    https://docs.progress.com/bundle/datadirect-product-compatibility/resource/datadirect-product-compatibility.pdf

Highlights of 6.0.0 Release

- The driver supports SQL read-write access to Snowflake. See Supported SQL statements and extensions on page 111 for details.
- The driver supports JDBC core functions. For details, refer to "JDBC support" in the *Progress DataDirect for JDBC Drivers Reference*.
- The driver supports Snowflake data types through data type inference. See Data types on page 23 and getTypeInfo() on page 24 for details.
- The driver provides proxy support. See Proxy server on page 22 and Connection property descriptions on page 67 for more information.
- The driver supports OAuth 2.0 authentication. See OAuth 2.0 authentication on page 49 for details.
- The driver supports user ID and password authentication. See User ID/password authentication on page 48 for details.
- The driver supports browser-based SSO authentication for Microsoft Windows.
- The driver supports using the COPY command for loading and unloading data from local file systems and cloud platforms, such as Amazon S3, Google Cloud, and Microsoft Azure. See Using COPY command on page 65 for further details.
- The driver supports the handling of large result sets with paging, and the FetchSize on page 84 connection property.
- 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 37 for details.
- The driver includes the Progress DataDirect Snowflake Configuration Manager for quick configuration and testing of your driver in a web browser. The tool allows you to:
  - Generate and edit connection URLs

- Test connect your connection URLs
- Execute SQL commands for testing
- Fetch access tokens and configure OAuth 2.0

For details, see Generating connection URLs with the Configuration Manager on page 41 and Testing connections and queries on page 42.

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\snowflake.jar
     ```
   - **UNIX/LINUX Example**

     ```
     CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/snowflake.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 OAuth 2.0 or user ID and password authentication.

OAuth 2.0 access token flow

```
jdbc:datadirect:snowflake://snowflakecomputing.com;
AccountName=account_name.us-east-1;AuthenticationMethod=OAuth2;
AccessToken=access_token;
```

Note: See OAuth 2.0 access token flow on page 15 for details.

UserID/Password

```
jdbc:datadirect:snowflake://snowflakecomputing.com;
AccountName=account_name.us-east-1;User=user_name;Password=password;
```

Note: See User ID/password authentication on page 14 for details.

You can also generate a connection string using the Progress DataDirect Snowflake Configuration Manager. For details, see Generating connection URLs with the Configuration Manager on page 41.

- **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:
   - Progress DataDirect Snowflake Configuration Manager: The Snowflake Configuration Manager is a browser-based tool that allows you to quickly generate connection URLs, test connections, and execute test queries.
   - 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.snowflake.SnowflakeDriver

DataSource class:

com.ddtek.jdbcx.snowflake.SnowflakeDataSource

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:

- You can also use the DataDirect Configuration Manager tool to generate and test connection URLs. For more information, see "Generating connection URLs with the Configuration Manager."
- 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}}`.

User ID/password authentication

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

Note: Since `AuthenticationMethod=userIdPassword` is the default, this setting does not have to be specified in a connection URL used for user ID/password implementations.

```
jdbc:datadirect:snowflake:AccountName=account_name;
DatabaseName=database_name;Schema=schema_name;Warehouse=warehouse_name;
User=user_name;Password=password;[property=value[;...]];
```

where:

account_name

specifies the full name of your account with region and cloud platform.

database_name

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

schema_name

specifies the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.

warehouse_name

specifies the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.

user_name

specifies the user ID that is used to connect to the Snowflake database.

password

specifies a password that is used to connect to your Snowflake database.

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 user ID/password authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake:AccountName=account_name.us-east-1;
  DatabaseName=payroll;Schema=xyz;Warehouse=accounting;
  User=JSmith@example.com;Password=secret;");
```

See also

User ID/password authentication on page 48
Connection property descriptions on page 67

OAuth 2.0 access token flow

Note: For OAuth 2.0 authentication, using a refresh token is more user-friendly and secure than using an access token. The access token expires every 10 minutes and may need to be generated multiple times a day, which can prolong the process of establishing a connection and cause a security risk.

The access token authentication flow passes the token directly from the client to the Snowflake instance for authentication. The token is obtained from sources external to the flow and specified using the AccessToken property. The following string includes the properties used to connect with the access token flow.

```
jdbc:datadirect:snowflake:AccountName=account_name;AuthenticationMethod=OAuth2;
AccessToken=access_token;DatabaseName=database_name;Schema=schema_name;
Warehouse=warehouse_name;[property=value[;...]];
```

where:

account_name

specifies the full name of your account with region and cloud platform.

access_token

specifies the access token required to authenticate to Snowflake. This property allows you to set the token manually.

Important:

- The access token is a confidential value used to authenticate to the server. To prevent unauthorized access, this value must be securely maintained.
- Access tokens expire ten minutes after generation. Once connected, the access token remains valid till the session is disconnected.

database_name

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

schema_name

specifies the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.

warehouse_name

specifies the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.

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 OAuth 2.0 access token flow.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake:AccountName=account_name.us-east-1;
  AuthenticationMethod=OAuth2;AccessToken=abc12cd34efg5678h9ij87klm6543no32pqr10;
  DatabaseName=payroll;Schema=xyz;Warehouse=accounting;");
```

See also

OAuth 2.0 authentication on page 49
Creating an OAuth security integration and obtaining client information on page 49
Obtaining access and refresh tokens using the Configuration Manager on page 54
Connection property descriptions on page 67
Access token flow on page 56

OAuth 2.0 authorization code grant

The authentication flow for the authorization code grant exchanges the authorization code and client credentials for the access token at the location specified by the RedirectURI. The following string includes the properties used to connect with the authorization code grant.

```
jdbc:datadirect:snowflake:AccountName=account_name;AuthenticationMethod=OAuth2;
AuthURI=auth_uri;DatabaseName=database_name;Schema=schema_name;
Warehouse=warehouse_name;ClientID=client_id;ClientSecret=client_secret;
RedirectURI=redirect_uri;[property=value[;...]];
```

where:

account_name

specifies the the full name of your account with region and cloud platform.

database_name

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

schema_name

specifies the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.

warehouse_name

specifies the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.

auth_uri

specifies the endpoint for obtaining an authorization code from the authorization service.

client_id

specifies the client ID key for your application.

client_secret

specifies the client secret for your application.

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

redirect_uri

specifies the endpoint that the client is returned to after authenticating with the service. This value must match the redirect URI specified in the Snowflake OAuth security integration.

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 OAuth 2.0 authorization code grant.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:snowflake:AccountName=account_name.us-east-1;
    AuthenticationMethod=OAuth2;DatabaseName=payroll;Schema=xyz;Warehouse=accounting;
    AuthURI=https://account_name.us-east-1.snowflakecomputing.com/oauth/authorize;
    ClientId=cd34efg5678h9ij87klm6543no32pqr10st987;
    ClientSecret=098zyx765wvu432tsr123qpo456;
    RedirectURI=http://localhost;");
```

See also

OAuth 2.0 authentication on page 49

Creating an OAuth security integration and obtaining client information on page 49
Connection property descriptions on page 67
Authorization code grant on page 57

OAuth 2.0 client credentials grant

The authentication flow for the client credentials grant exchanges client credentials for the access token at the location specified by the TokenURI. Web-based login and consent are not required. The following string includes the properties used to connect with the client credentials grant.

```
jdbc:datadirect:snowflake:AccountName=account_name;AuthenticationMethod=OAuth2;
DatabaseName=database_name;Schema=schema_name;Warehouse=warehouse_name;
ClientID=client_id;ClientSecret=client_secret;
TokenURI=token_uri;[property=value[;...]];
```

where:

account_name

specifies the full name of your account with region and cloud platform.

database_name

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

schema_name

specifies the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.

warehouse_name

specifies the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.

client_id

specifies the client ID for your application.

client_secret

specifies the client secret for your application.

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

token_uri

specifies the endpoint from which the driver fetches access tokens.

property=value

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

The following example connection string includes the properties for connecting with the OAuth 2.0 client credentials grant.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:snowflake:AccountName=account_name.us-east-1;
    AuthenticationMethod=OAuth2;DatabaseName=payroll;Schema=xyz;Warehouse=accounting;

ClientID=cd34efg5678h9ij87klm6543no32pqr10st987;ClientSecret=098zyx765wvu432tsr123qpo456;


TokenURI=https://account_name.us-east-1.snowflakecomputing.com/oauth/token-request;");
```

See also

OAuth 2.0 authentication on page 49
Creating an OAuth security integration and obtaining client information on page 49
Connection property descriptions on page 67
Client credentials grant on page 58

OAuth 2.0 refresh token grant

The refresh token grant is used to request a new access token or renew an expired one by exchanging the refresh token at the endpoint specified by the TokenURI property. The following string includes the properties used to connect with the refresh token grant.

```
jdbc:datadirect:snowflake:AccountName=account_name;AuthenticationMethod=OAuth2;
DatabaseName=database_name;Schema=schema_name;Warehouse=warehouse_name;
ClientID=client_id;ClientSecret=client_secret;TokenURI=token_uri;
RefreshToken=refresh_token;[property=value[;...]];
```

where:

account_name

specifies the full name of your account with region and cloud platform.

database_name

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

schema_name

specifies the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.

warehouse_name

specifies the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.

client_id

specifies the client ID for your application.

client_secret

specifies the client secret for your application.

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

token_uri

specifies the endpoint from which the driver fetches access tokens.

refresh_token

specifies the refresh token used to either request a new access token or renew an expired access token.

Important: The refresh token 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 OAuth 2.0 access token flow.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake:AccountName=account_name.us-east-1;
  AuthenticationMethod=OAuth2;DatabaseName=payroll;Schema=xyz;
  Warehouse=accounting;ClientID=123456789;ClientSecret=FaZBFRsGXTaR;
  TokenURI=https://account_name.us-east-1.snowflakecomputing.com/oauth/token-request;
  RefreshToken=abc12cd34efg5678h9ij87klm6543no32pqr10;");
```

See also

OAuth 2.0 authentication on page 49
Creating an OAuth security integration and obtaining client information on page 49
Obtaining access and refresh tokens using the Configuration Manager on page 54
Connection property descriptions on page 67
Refresh token grant on page 59

Single sign-on authentication

This string includes the properties used to connect with single sign-on (SSO) authentication.

```
jdbc:datadirect:snowflake:AccountName=account_name;
AuthenticationMethod=BrowserBasedSSO;DatabaseName=database_name;
Schema=schema_name;Warehouse=warehouse_name;
User=user_name;[property=value[;...]];
```

where:

account_name

specifies the the full name of your account with region and cloud platform.

database_name

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

schema_name

specifies the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.

warehouse_name

specifies the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.

user_name

specifies the user ID that is used to connect to the Snowflake database.

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 browser-based single sign-on authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake:AccountName=account_name.us-east-1;
  AuthenticationMethod=BrowserBasedSSO;DatabaseName=payroll;
  Schema=xyz;Warehouse=accounting;User=jSmith@example.com;");
```

Key-pair authentication

This string includes the properties used to connect to Snowflake using key-pair authentication.

```
jdbc:datadirect:snowflake:AccountName=account_name;AuthenticationMethod=KeyPair;
DatabaseName=database_name;Schema=schema_name;Warehouse=warehouse_name;
PrivateKeyFile=privatekeyfile_location;PrivateKeyPassphrase=password;[property=value[;...]];
```

where:

account_name

specifies the full name of your account with region and cloud platform.

database_name

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

schema_name

specifies the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.

warehouse_name

specifies the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.

privatekeyfile_location

specifies the absolute path to the private key file you want to use for authentication.

Note: Alternatively, you can specify the content of a private key in the connection string. To specify the private key content, use the PrivateKeyContent connection property.

password

specifies a password that is used to descrypt either a private key or the content of a private key.

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 key-pair authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake:AccountName=account_name.us-east-1;
  AuthenticationMethod=KeyPair;DatabaseName=payroll;Schema=xyz;Warehouse=accounting;
  PrivateKeyFile=C:\Program Files\privatekey.p8;PrivateKeyPassphrase=abc123;");
```

See also

Key-pair authentication on page 61

Proxy server

This string includes the properties you may need to connect through a proxy server with UserID/Password authentication.

```
jdbc:datadirect:snowflake:AccountName=account_name;DatabaseName=database_name;
Schema=schema_name;Warehouse=warehouse_name;ProxyHost=proxy_host;
ProxyPassword=proxy_password;ProxyPort=proxy_port;ProxyUser=proxy_user;
User=user_name;Password=password;[property=value[;...]];
```

where:

account_name

specifies the full name of your account with region and cloud platform.

database_name

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

schema_name

specifies the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.

warehouse_name

specifies the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.

proxy_host

specifies the proxy server to use for the first connection.

proxy_password

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

proxy_port

specifies the port number where the proxy server is listening for requests for the first connection. The default is `0`.

proxy_user

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

user_name

specifies the user ID that is used to connect to the Snowflake instance. For example, `jsmith@example.com`.

password

specifies the password used to connect to your Snowflake instance.

property=value

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

The following example connection string includes the properties required for using a proxy server with UserID/Password authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake:AccountName=account_name.us-east-1;
  DatabaseName=payroll;Schema=xyz;Warehouse=accounting;
  ProxyHost=pserver;ProxyPassword=secret;ProxyPort=808;
  ProxyUser=PUser;User=jsmith@example.com;Password=secret;");
```

See also

Connection property descriptions on page 67

Data types

The following table lists data types supported by the driver and how they are mapped to JDBC data types.

See "getTypeInfo()" for getTypeInfo() results of data types supported by the driver.

Table 1: Snowflake Data Types

Snowflake Data Type	JDBC Data Type
ARRAY	VARCHAR
BIGINT[1]	NUMERIC

[1] This data type is a fixed-point number type for which precision and scale cannot be specified. By default, this data type maps to NUMERIC. However, you may use the IntegerFieldMapping on page 87 property to map this type and other such types to BIGINT.

Snowflake Data Type	JDBC Data Type
BINARY	BINARY
BOOLEAN	BOOLEAN
CHAR	VARCHAR
DATE	DATE
DECIMAL	DECIMAL
DOUBLE	DOUBLE
FLOAT	DOUBLE
INTEGER[1]	NUMERIC
NUMBER	DECIMAL
OBJECT	VARCHAR
REAL	DOUBLE
TIME	TIME
TIMESTAMP	TIMESTAMP
TIMESTAMPLTZ	TIMESTAMP
TIMESTAMPNTZ	TIMESTAMP
TIMESTAMPTZ	TIMESTAMP
VARBINARY	BINARY
VARCHAR	VARCHAR
VARIANT	VARCHAR

getTypeInfo()

The DatabaseMetaData.getTypeInfo() method returns information about data types. The following table provides getTypeInfo() results for supported data types.

Table 2: getTypeInfo() Results

TYPE_NAME = ARRAY AUTO_INCREMENT = NULL CASE_SENSITIVE = TRUE CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) 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 = 16777216 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = ARRAY UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = BIGINT[2] AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 2 (NUMERIC) 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 = 38 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = BIGINT UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = BINARY AUTO_INCREMENT = NULL CASE_SENSITIVE = FALSE CREATE_PARAMS = MAX LENGTH DATA_TYPE = -2 (BINARY) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = BINARY MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 8388608 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = BINARY UNSIGNED_ATTRIBUTE = NULL

[2] This data type is a fixed-point number type for which precision and scale cannot be specified. By default, this data type maps to NUMERIC. However, you may use the IntegerFieldMapping on page 87 property to map this type and other such types to BIGINT.

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 = NULL PRECISION = 1 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = BOOLEAN UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = CHAR AUTO_INCREMENT = NULL CASE_SENSITIVE = TRUE CREATE_PARAMS = MAX LENGTH DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = CHAR MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16777216 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = CHAR UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = DATE AUTO_INCREMENT = NULL CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 91 (DATE) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = DATE MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = DATE UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = DECIMAL AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = PRECISION,SCALE DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = DECIMAL MAXIMUM_SCALE = 37	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = DECIMAL 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 = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = DOUBLE UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = FLOAT 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 = FLOAT MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 15 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = FLOAT UNSIGNED_ATTRIBUTE = FALSE

TYPE_NAME = INTEGER[2] AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 2 (NUMERIC) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = INTEGER MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = INTEGER UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = NUMBER AUTO_INCREMENT = FALSE CASE_SENSITIVE = FALSE CREATE_PARAMS = PRECISION,SCALE DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = NUMBER MAXIMUM_SCALE = 37	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = NUMBER UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = OBJECT AUTO_INCREMENT = NULL CASE_SENSITIVE = TRUE CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = OBJECT MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16777216 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = OBJECT UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = REAL 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 = REAL MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 15 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = REAL UNSIGNED_ATTRIBUTE = FALSE
TYPE_NAME = TIME AUTO_INCREMENT = NULL CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 92 (TIME) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = TIME MAXIMUM_SCALE = 9	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 18 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = TIME UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = TIMESTAMP AUTO_INCREMENT = NULL CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = TIMESTAMP MAXIMUM_SCALE = 9	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 35 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = TIMESTAMP UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = TIMESTAMPLTZ AUTO_INCREMENT = NULL CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = TIMESTAMPLTZ MAXIMUM_SCALE = 9	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 35 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = TIMESTAMPLTZ UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = TIMESTAMPNTZ AUTO_INCREMENT = NULL CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = TIMESTAMPNTZ MAXIMUM_SCALE = 9	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 35 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = TIMESTAMPNTZ UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = TIMESTAMPTZ AUTO_INCREMENT = NULL CASE_SENSITIVE = FALSE CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = TIMESTAMPTZ MAXIMUM_SCALE = 9	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 35 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = TIMESTAMPTZ UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = VARBINARY AUTO_INCREMENT = NULL CASE_SENSITIVE = FALSE CREATE_PARAMS = MAX LENGTH DATA_TYPE = -2 (BINARY) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = VARBINARY MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 8388608 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = VARBINARY UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = VARCHAR AUTO_INCREMENT = NULL CASE_SENSITIVE = TRUE CREATE_PARAMS = MAX 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 = 16777216 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = VARCHAR UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = VARIANT AUTO_INCREMENT = NULL CASE_SENSITIVE = TRUE CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = FALSE LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = VARIANT MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16777216 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL TYPE_NAME = VARIANT UNSIGNED_ATTRIBUTE = NULL

DataDirect tools

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

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

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:

- DbVisualizer
- Interactive SQL

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\snowflake.jar
   ```

   Linux

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

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

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

     ```
     Snowflake
     ```

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

     ```
     jdbc:datadirect:snowflake://snowflakecomputing.com
     ```

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

     ```
     com.ddtek.jdbc.snowflake.SnowflakeDriver
     ```

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 OAuth 2.0 authentication.

     Note: You can also generate connection strings using Snowflake Configuration Manager. For more information, see Generating connection URLs with the Configuration Manager on page 41.

     OAuth 2.0 Access token flow

     ```
     jdbc:datadirect:snowflake://snowflakecomputing.com;AccountName=account_name;
     AuthenticationMethod=OAuth2;AccessToken=access_token;
     ```

- **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.
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 AGENTS WHERE AGENTID = <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 snowflake.jar --isql
   ```

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

   ```
   AccountName=myorg-account.us-east-1
   ```

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 INFORMATION_SCHEMA.SYSTEM_TABLES;
```

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/snowflake.jar
```

Windows Example

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

UNIX Example

```
CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/snowflake.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:snowflake://snowflakecomputing.com;AccountName=account_name;
  UserName=user_name;Password=password");
```

Note: See User ID/password authentication on page 14 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:snowflake://snowflakecomputing.com;AccountName=account_name;
UserName=user_name;Password=password;[property=value[;...]];
```

where:

server_name

specifies the base URL of the Snowflake instance to which you want to issue requests. For example, `//snowflakecomputing.com`.

account_name

specifies the the full name of your account with region and cloud platform.

user_name

Specifies the user ID used to authenticate to Snowflake with UserID/Password authentication method.

password

Specifies the password used to authenticate to Snowflake 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:snowflake://snowflakecomputing.com;AccountName=accountname.us-east-1;

  UserName=JohnSmith;Password=abc123");
```

See also

Connection URL examples on page 14
Connection property descriptions on page 67

Generating connection URLs with the Configuration Manager

The driver includes a browser-based tool, Progress DataDirect Snowflake Configuration Manager, that allows you to generate connection URLs, test connections, and execute test queries. This section will guide you through generating and testing a connection URL that can be used by your application.

To generate a connection URL:

1. Open the Snowflake Configuration Manager by double-clicking the driver jar file. Or, in a command line, navigate to the directory containing your driver jar file; then, execute the following command:

   ```
   java -jar snowflake.jar
   ```

   The Snowflake Configuration Manager opens in your default web browser.

2. From the browser window, provide values of the connection properties you want to configure in the corresponding fields. A connection URL will generate in the Connection String field as you provide settings. To view more properties, select the tabs at the top of the page. See "Connection URL examples" for a list of required properties and properties used for different configurations.

Note: If you do not specify a value for an optional property, the property will be omitted from the string and the default value will be used.

3. Optionally, you can manually edit your string by clicking the Edit button ( ).
4. At any point during the process, you can click **Test Connect** to attempt to connect to the service using the string generated in the Connection String field. In the **Test Connection** window:
   a) Provide values for any fields required by your service.
   b) Optionally, in the Test Query field, enter any SQL queries you want to execute during the test. For example:

      ```
      SELECT * FROM INFORMATION_SCHEMA.TABLES
      ```

   c) Click **Execute**.

      If successful, the window displays a confirmation message and, if a query was specified, the results of the query.

Note: The information you enter in the logon dialog box during a test connect is not saved.

5. To use your string, click the Copy button ( ) and paste the string to a location that can be used by your application.

See also

Connection URL examples on page 14

Testing connections and queries

You can quickly test a connection string and queries using Progress DataDirect Snowflake Configuration Manager.

To test your connection and query:

1. Open the Snowflake Configuration Manager by double-clicking on the driver jar file. Or, in a command line, navigate to the directory containing your driver jar file; then, execute the following command:

   ```
   java -jar snowflake.jar
   ```

   The Snowflake Configuration Manager opens in your default web browser.

2. Provide a connection string to test using one of the following methods:

   - Entering a string: Click the Edit button ( ); then, paste your string into the Connection String field. If you prefer, you can also type a string directly into this field.
   - Generating a string: Provide values of the connection properties you want to configure into the corresponding fields. The Snowflake Configuration Manager will generate a string in the Connection String field based on the values you specify.

3. Click **Test Connect** to attempt to connect to the service using the string specified in the Connection String field. The **Test Connection** window appears.

4. Provide connection property values for any fields required by your service.
5. To execute a test query with the test connection, enter a SQL query into the Test Query field. For example:

   ```
   SELECT * FROM INFORMATION_SCHEMA.TABLES
   ```

6. Click **Execute**.

   If successful, the window displays a confirmation message and, if a query was specified, the results of the query.

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 14

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.snowflake.SnowflakeDataSource;
```

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")`.

```
SnowflakeDataSource mds = new SnowflakeDataSource();
mds.setDescription("My Snowflake Data Source");
mds.setUserName("JSmith");
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:

```
if (ds instanceof SnowflakeDataSource)
{
SnowflakeDataSource jmds = (SnowflakeDataSource) ds;
System.out.println("description=" + jmds.getDescription());
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.

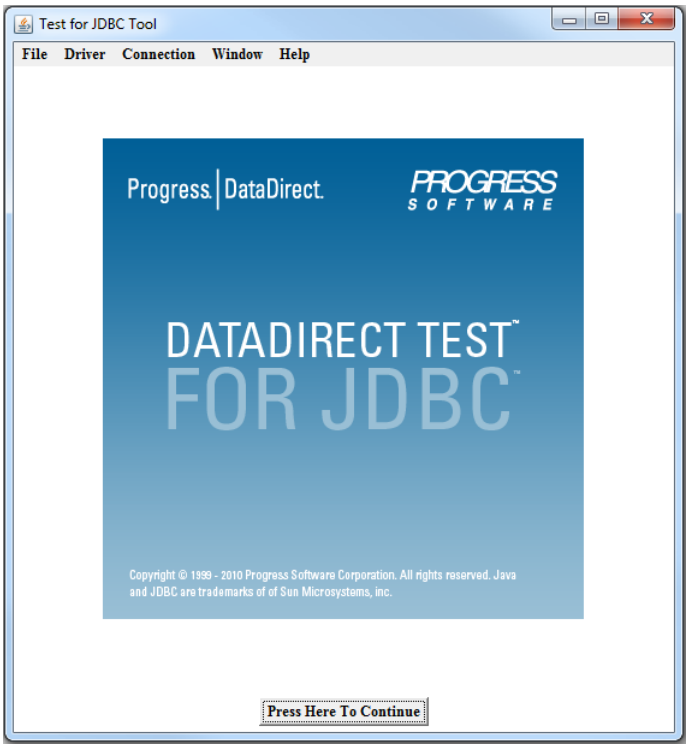
Take the following steps to establish a connection.

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.

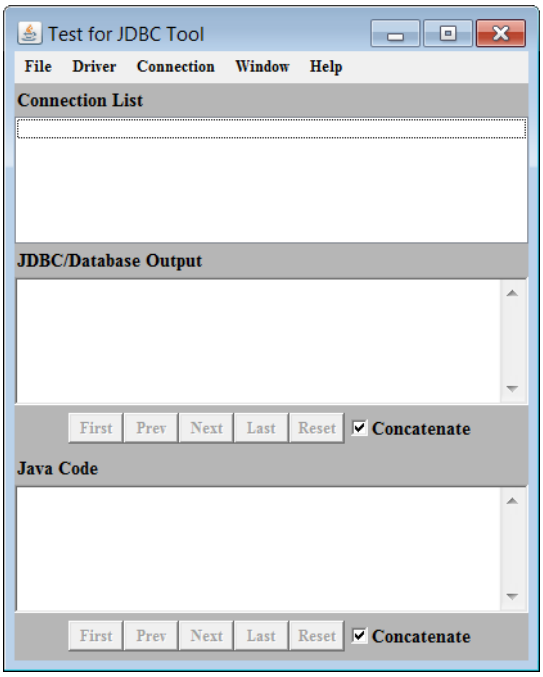
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

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



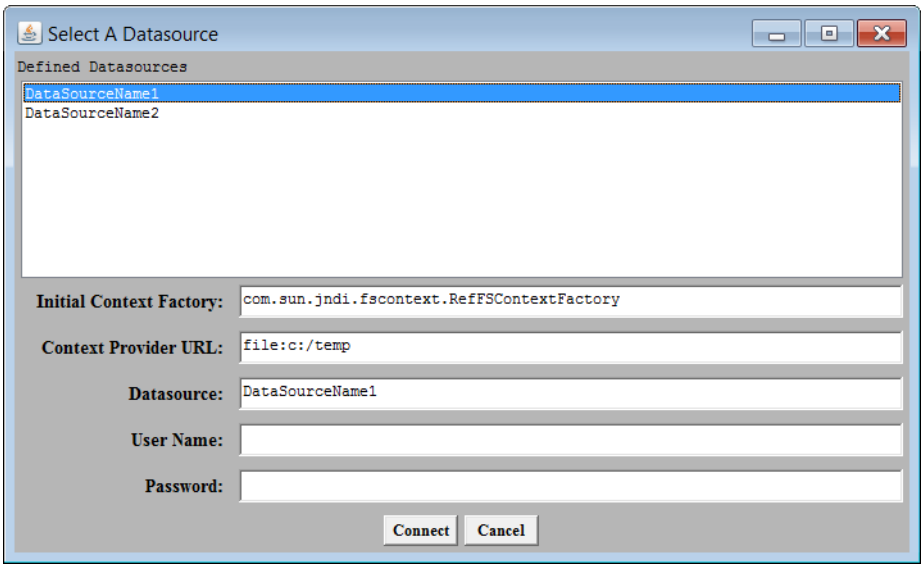
3. Click **Press Here to Continue**.

   The main dialog appears:



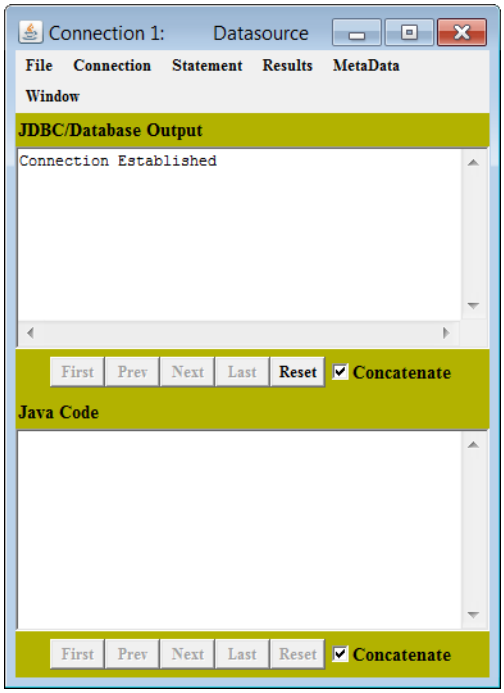
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   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 specified user IDs and passwords.
- *OAuth 2.0* allows the user to authenticate to a Snowflake instance without having to specify user ID and password.
- *Browser-based SSO* authenticates the user using a web browser in Microsoft Windows.
- *Key-pair* authenticates the user using a pair of private and public keys.

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

See also

AuthenticationMethod on page 77

User ID/password authentication

To configure the driver to use user ID/Password authentication.

- Set the AuthenticationMethod property to `userIdPassword`. Since `userIdPassword` is the default, it does not have to be specified in a connection URL used for user ID/password implementations.
- Set the AccountName property to specify the full name of your account and the region where it is hosted. For example, *`account_name`*.us-east-1.
- Set the DatabaseName property to specify the name of the database to which you are connecting.
- Set the Schema property to specify the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.
- Set the Warehouse property to specify the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.
- Set the User property to provide the user ID.
- Set the Password property to provide the password.

The following examples demonstrate how to make a connection using the user ID/password authentication.

Connection URL:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake:AccountName=account_name.us-east-1;
  DatabaseName=payroll;Schema=xyz;Warehouse=accounting;
  User=jsmith;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:

```
SnowflakeDataSource mds = new SnowflakeDataSource();
mds.setDescription("My Snowflake Data Source");
mds.setAccountName("account_name.us-east-1");
mds.setDatabaseName("payroll");
mds.setSchema("xyz");
mds.setWarehouse("accounting");
mds.setUser("jsmith");
mds.setPassword("secret");
```

See also

AuthenticationMethod on page 77
Password on page 92
User on page 108

OAuth 2.0 authentication

The driver supports OAuth 2.0 access to Snowflake resources. The driver may be configured to use either Snowflake OAuth or External OAuth. Snowflake OAuth is Snowflake's built-in OAuth service, while External OAuth allows you to use an external authorization service.

The following workflow describes the process for setting up OAuth 2.0 access.

1. Create an OAuth security integration and obtain client information. See this topic for guidance on creating the security integration and obtaining client information, such as client ID, client secret, authorization URI, redirect URI, and token URI.
2. Obtain access and refresh tokens using the Configuration Manager. If you want to use the access token flow or refresh token grant, see this topic for directions on obtaining access and refresh tokens using the Configuration Manager.
3. Configure the driver to use OAuth 2.0. You can configure the driver to access Snowflake resources using any supported OAuth 2.0 grant types. The driver supports the following grant types:
   - Access token flow
   - Authorization code grant
   - Client credentials grant
   - Refresh token grant

Creating an OAuth security integration and obtaining client information

An OAuth security integration is a Snowflake object that enables client application access to Snowflake resources using OAuth 2.0. There are two types of OAuth security integrations in Snowflake: Snowflake OAuth and External OAuth. Snowflake OAuth is Snowflake's built-in OAuth service, while External OAuth allows you to use an external authorization service. It is during the process of creating an OAuth integration that you obtain the client information the driver requires to connect with Snowflake.

See the following topics for details.

- Creating a Snowflake OAuth security integration
- Creating an External OAuth security integration

Note: For additional details, refer to OAuth in the Snowflake documentation.

Creating a Snowflake OAuth security integration

Snowflake OAuth is Snowflake's built-in OAuth service. When you create a Snowflake OAuth security integration, you effectively register your client application with Snowflake. After you create the security integration, you can obtain the client information (client ID, client secret, authorization endpoint, and token endpoint) required for configuring the driver.

Take the following steps to create a Snowflake security integration and obtain required client information.

Note: The Snowflake user must have either the ACCOUNTADMIN role or the global CREATE INTEGRATION privilege to execute the `create security integration` command.

1. Log in to Snowflake.
2. Open **Worksheets** or select the scheme you are working with.

   Note: Administrators may check to see whether the SYSADMIN role has the required privileges for the warehouse by navigating to **Admin > Warehouses > *warehouse_name* > Edit > Privileges**.

3. Run the following command to create the security integration.

   Note: For details on parameters, refer to CREATE SECURITY INTEGRATION (Snowflake OAuth) in the Snowflake documentation.

   ```
   create security integration integration_name
    type = oauth
    enabled = true
    oauth_client = custom
    oauth_client_type = confidential
    oauth_redirect_uri =redirect_uri
    oauth_issue_refresh_tokens = true
    oauth_refresh_token_validity=7776000;
   ```

   where:

   integration_name

   is the name of the security integration.

   redirect_uri

   is the client URI. The web browser is redirected to this URI after authorization. For example, to test, you might use `http://localhost`.

4. Run the following `describe` command to obtain the client ID, authorization URI, and token URI.

   ```
   describe security integration integration_name;
   ```

5. Run the following `select` command to obtain the client secret.

   ```
   select SYSTEM$SHOW_OAUTH_CLIENT_SECRETS(integration_name);
   ```

Note: Two client secrets will be returned. Either may be used to configure the driver.

Results

You have created a Snowflake OAuth security integration.

What's next

If you are using the client credentials grant or the authorization code grant, proceed to the corresponding topic for guidance on configuring the driver.

- Authorization code grant
- Client credentials grant

If you are using the access token flow or refresh token grant, see Obtaining access and refresh tokens using the Configuration Manager.

Creating an External OAuth security integration

External OAuth allows you to use an external authorization service to access Snowflake. To implement OAuth for a client application, you must configure the external authorization service as well as Snowflake. During this process, you obtain the client information (client ID, client secret, authorization endpoint, and token endpoint) required to configure the driver.

Note: The following instructions provide guidance on developing an External OAuth integration with Okta. However, Snowflake supports a number of other external authorization services. Refer to External OAuth in the Snowflake documentation for details.

The following workflow describes the process for creating an External OAuth security integration with Okta.

1. Okta: Create an OAuth client on page 51
2. Okta: Create the OAuth authorization server on page 52
3. Snowflake: Create External OAuth security integration on page 53

Okta: Create an OAuth client

1. Log in to Okta, and navigate to the Admin Console.
2. Navigate to **Applications > Applications**. Then, click **Create App Integration**.
3. For **Sign-in method**, select **OIDC**.
4. For **Application type**, select **Native Application**.

Note: If you are using Hybrid Data Pipeline, select **Web Application**.

5. Click **Next**.
6. On the **App Integration** screen, enter the **App integration name**, and select the grant types you plan to use.
7. For **Sign-in redirect URIs**, enter your Snowflake account URL.

Note: For testing, you could also enter `http://localhost` or the Postman callback URL.

Note:

8. Click **Save**.
9. From the **General** tab of your integration, click **Edit** next to **Client Credentials**.
10. Select **Use Client Authentication**.
11. For **Client authentication**, select **Client secret**.
12. Click **Save**.
13. From the **General** tab of your integration, save the **Client ID** and **Client Secret** values.
14. Select the **Assignments** tab, and assign the current user to the app.

Results: An Okta OAuth client has been created. In addition, you have saved the client ID and client secret which will be needed to configure the driver.

Okta: Create the OAuth authorization server

1. From the Okta Admin Console, navigate to **Security > API** and click **Add Authorization Server**.
2. For **Audience**, enter the Snowflake account URL. Then, click **Save**.
3. From the **Settings** tab, save the **Issuer** value.
4. From the **Settings** tab, open the **Metadata URI** and save the following values.
   - **JWS keys URL:** the value of the `jwks_uri` parameter
   - **Authorization URL:** the value of the `authorization_endpoint` parameter
   - **Token URL:** the value of the `token_endpoint` parameter
5. From the **Scopes** tab, click **Add Scope**.
6. Enter the Snowflake scope and scope information.

   Note: Snowflake scopes are typically tied to Snowflake roles. For example: `session:role:SYSADMIN`.

7. Click **Save**.
8. From the **Access Policies** tab, click **Add New Access Policy**.
9. Select **The following clients**, and add the OAuth client application created in "Okta: Create an OAuth client".
10. Click **Create Policy**.
11. Click **Add Rule**, and make selections based on your requirements.

    Note:

    - **Client Credentials** and **Resource Owner Password** are required for Snowflake.
    - **Authorization Code** should be selected if you plan to use the authorization code grant or use the driver Configuration Manager to generate access and refresh tokens. This is also needed to support the authorization code grant in Hybrid Data Pipeline.

12. Click **Create rule**.

Results: An Okta authorization server has been created to use with an External OAuth security integration in Snowflake. In addition, you have recorded the token URI and the authorization URI which are required to configure the driver for some OAuth 2.0 grant types.

Snowflake: Create External OAuth security integration

Note: The Snowflake user must have either the ACCOUNTADMIN role or the global CREATE INTEGRATION privilege to execute the `create security integration` command.

1. Log in to Snowflake.
2. Open **Worksheets** or select the scheme you are working with.

   Note: Administrators may check to see whether the SYSADMIN role has the required privileges for the warehouse by navigating to **Admin > Warehouses > *warehouse_name* > Edit > Privileges**.

3. Run the following command to create the security integration.

   Note: For more details on parameters, refer to CREATE SECURITY INTEGRATION (External OAuth) in the Snowflake documentation.

```
create security integration integration_name
    type = external_oauth
    enabled = true
    external_oauth_any_role_mode = 'ENABLE'
    external_oauth_type = okta
    external_oauth_issuer = 'okta_issuer'
    external_oauth_jws_keys_url = 'okta_jws_keys_url'
    external_oauth_audience_list = ('audience_list')
    external_oauth_token_user_mapping_claim = 'sub'
    external_oauth_snowflake_user_mapping_attribute = 'login_name';
```

   where:

   integration_name

   is the name of the security integration.

   okta_issuer

   is the URL that defines the Okta OAuth 2.0 authorization server. Obtained in Step 3 of "Okta: Create the OAuth authorization server". For example: `https://dev-123456.okta.com/oauth2/abcdefg`.

   okta_jws_keys_url

   is the Okta URL where public keys may be downloaded to validate an External OAuth access token. Obtained in Step 4 of "Okta: Create the OAuth authorization server". For example: `https://dev-123456.okta.com/oauth2/abcdefg/v1/keys`.

   audience_list

   is your Snowflake Account URL. For example: `https://myorg-account.us-east-1.snowflakecomputing.com`.

`external_oauth_token_user_mapping_claim`

specifies a key that indicates the user associated with the access token. Typically, in Okta, the `sub` key passes the Okta user associated with a given access token.

`external_oauth_snowflake_user_mapping_attribute`

specifies the Snowflake user attribute that should be used to map the access token to a Snowflake user record. In the example, the Snowflake user `login_name` attribute is being used for this purpose. Therefore, the value of the `sub` key must match the value of `login_name` to complete the OAuth flow and grant the client application access to Snowflake resources.

Results: You have created an External OAuth security integration.

What's next

If you are using the client credentials grant or the authorization code grant, proceed to the corresponding topic for guidance on configuring the driver.

- Authorization code grant
- Client credentials grant

If you are using the access token flow or refresh token grant, see Obtaining access and refresh tokens using the Configuration Manager.

Obtaining access and refresh tokens using the Configuration Manager

You need the following information before you begin. See "Creating an OAuth security integration and obtaining client information" for details.

- Authorization URI: The endpoint for obtaining an authorization code from a third-party authorization service
- Token URI: The endpoint used to exchange authentication credentials for access tokens
- Redirect URI: The endpoint that the client is returned to after authenticating with the service
- Client ID: The client ID for your application
- Client Secret: The client secret for your application

The following steps describe how you can use the Progress DataDirect Snowflake Configuration Manager to obtain access and refresh tokens. In addition, the Configuration Manager produces a connection URL that you can use in your application.

Note: You must allow popups in your browser to obtain access and refresh tokens with the Configuration Manager.

---

1. Open the Snowflake Configuration Manager by double-clicking the driver jar file. Or, in a command line, navigate to the directory containing your driver jar file; then, execute the following command:

   ```
   java -jar snowflake.jar
   ```

   The Snowflake Configuration Manager opens in your default web browser.

2. Provide the following information in the fields provided. These values are required in the connection string. Without them, even basic SQL queries cannot be run.

   - **Account Name**

- **Database Name**
- **Schema**
- **Warehouse**

3. Set **Authentication Method** to `OAuth2`.
4. Provide the following information in the fields provided.
   - **Authorization URI**
   - **Token URI**
   - **Redirect URI**
   - **Client ID**
   - **Client Secret**
5. Obtain access and refresh tokens.
   a. Click **Fetch OAuth Token**.
   b. If the logon popup appears, enter your credentials. (This popup may not appear if you previously logged on.)
   c. If consent popup appears, provide consent, allowing the Configuration Manager to retrieve the tokens. (This popup may not appear if you previously provided consent to the Configuration Manager.)
   d. The **Access Token** and **Refresh Token** fields populate with values retrieved from the OAuth authorization server.
6. Click **Test Connect** to verify connectivity and run SQL queries against the service.

Results

The **Access Token** and **Refresh Token** fields include access and refresh tokens that you can use to implement OAuth 2.0.

The connection string in the **Connection String** field may be copied and used in your JDBC application to connect with your service.

Note:

Not all the values in the resulting connection string are required. However, the connection string can be copied directly into your JDBC application. The driver ignores any values that do not apply to your OAuth implementation.

For example, the refresh token grant connection string, derived from the Configuration Manager, might include the following properties.

```
jdbc:datadirect:snowflake:AuthenticationMethod=OAuth2;
AuthURI=auth_uri;TokenURI=token_uri;RedirectURI=redirect_uri
ClientID=client_id;ClientSecret=client_secret;
OAuthCode=oauth_code;AcessToken=access_token;RefreshToken=refresh_token;
AccountName=account_name;DatabaseName=database_name;Schema=schema_name;
Warehouse=warehouse_name;
```

However, only the following properties are required for a refresh token grant connection string.

```
jdbc:datadirect:snowflake:AuthenticationMethod=OAuth2;
TokenURI=token_uri;RedirectURI=redirect_uri;ClientID=client_id;
ClientSecret=client_secret;RefreshToken=refresh_token;
AccountName=account_name;DatabaseName=database_name;
Schema=schema_name;Warehouse=warehouse_name;
```

What's next

You may now configure the driver to use either the access token flow or the refresh token grant. See the following topics for details.

- Access token flow on page 56
- Refresh token grant on page 59

See also

Creating an OAuth security integration and obtaining client information on page 49

Configuring the driver to use OAuth 2.0

The driver supports the following OAuth 2.0 flow and grant types.

- Access token flow on page 56
- Authorization code grant on page 57
- Client credentials grant
- Refresh token grant on page 59

Access token flow

Note: For OAuth 2.0 authentication, using a refresh token is more user-friendly and secure than using an access token. The access token expires every 10 minutes and may need to be generated multiple times a day, which can prolong the process of establishing a connection and cause a security risk.

The access token authentication flow passes the token directly from the client to the Snowflake instance for authentication. The token is obtained from sources external to the flow and specified using the AccessToken property.

To configure the driver to use the access token flow for OAuth 2.0 authentication:

- Set the AuthenticationMethod property to `OAuth2`.
- Set the AccountName property to specify the full name of your account and the region where it is hosted. For example, *`account_name`*`.us-east-1`.
- Set the DatabaseName property to specify the name of the database to which you are connecting.
- Set the Schema property to specify the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.
- Set the Warehouse property to specify the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.
- Set the AccessToken property to the value of the token obtained from external sources.

  Important:

  - The access token is a confidential value used to authenticate to the server. To prevent unauthorized access, this value must be securely maintained.
  - Access tokens expire ten minutes after generation. Once connected, the access token remains valid till the session is disconnected.

The following examples show the connection information required to establish a session using the access token flow.

Connection URL

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:snowflake:AuthenticationMethod=OAuth2;
    AccountName=account_name.us-east-1;DatabaseName=payroll;Schema=xyz;
    Warehouse=accounting;AccessToken=abc12cd34efg5678h9ij87klm6543no;");
```

Data Source

```
SnowflakeDataSource mds = new SnowflakeDataSource();
mds.setDescription("My Snowflake Data Source");
mds.setAuthenticationMethod("OAuth2");
mds.setAccountName("account_name.us-east-1");
mds.setDatabaseName("payroll");
mds.setSchema("xyz");
mds.setWarehouse("accounting");
mds.setAccessToken("abc12cd34efg5678h9ij87klm6543no32pqr10");
```

See also

Creating an OAuth security integration and obtaining client information on page 49
Obtaining access and refresh tokens using the Configuration Manager on page 54
Connection property descriptions on page 67

Authorization code grant

The authorization code grant is a commonly used authentication flow for web and native applications. It provides secure connections by requiring multiple points of authentication before permitting access to data. When using the authorization code flow, the application is first redirected to the location hosting the temporary authorization code and retrieves it. Next, after being redirected to the location specified by the RedirectURI property, the application exchanges the authorization code, client ID, and client secret for the access token.

To use an authorization code grant:

- Set the AuthenticationMethod property to `OAuth2`.
- Set the AccountName property to specify the full name of your account and the region where it is hosted. For example, *`account_name`*`.us-east-1`.
- Set the DatabaseName property to specify the name of the database to which you are connecting.
- Set the Schema property to specify the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.
- Set the Warehouse property to specify the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.
- Set the AuthURI property to the endpoint used to obtain the authorization code from the authorization service.
- Set the ClientID property to specify the client ID key for your application.
- Set the ClientSecret property to specify the client secret for your application.

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

- Set the RedirectURI to specify the endpoint that the client is returned to after authenticating with the service. This value must match the redirect URI specified in the Snowflake OAuth security integration.

The following example demonstrates using an authorization code grant:

Connection URL:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake:AuthenticationMethod=OAuth2;
  AccountName=account_name.us-east-1;
  DatabaseName=payroll;Schema=xyz;Warehouse=accounting;
  AuthURI=https://account_name.us-east-1.snowflakecomputing.com/oauth/authorize;
  ClientId=cd34efg5678h9ij87klm6543no32pqr10st987;
  ClientSecret=098zyx765wvu432tsr123qpo456;
  RedirectUri=https://lvn.me/app_callback.html;");
```

Data Source:

```
SnowflakeDataSource mds = new SnowflakeDataSource();
mds.setDescription("My Snowflake Data Source");
mds.setAuthenticationMethod("OAuth2");
mds.setAccountName("account_name.us-east-1");
mds.setDatabaseName("payroll");
mds.setSchema("xyz");
mds.setWarehouse("accounting");
mds.setAuthURI("https://account_name.us-east-1.snowflakecomputing.com/oauth/authorize");
mds.setClientID("cd34efg5678h9ij87klm6543no32pqr10st987");
mds.setClientSecret("098zyx765wvu432tsr123qpo456");
mds.setRedirectURI("https://lvh.me/app_callback.html");
```

See also

Creating an OAuth security integration and obtaining client information on page 49
Connection property descriptions on page 67

Client credentials grant

The authentication flow for the client credentials grant exchanges client credentials for the access token at the location specified by the TokenURI. Web-based login and consent are not required.

To configure the driver to use a client credentials grant:

- Set the AuthenticationMethod property to `OAuth2`.
- Set the AccountName property to specify the full name of your account and the region where it is hosted. For example, *`account_name`*`.us-east-1`.
- Set the DatabaseName property to specify the name of the database to which you are connecting.
- Set the Schema property to specify the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.
- Set the Warehouse property to specify the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.
- Set the ClientID property to specify the client ID for your application.
- Set the ClientSecret property to specify client secret for your application.

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

- Set the TokenURI property to specify the endpoint from which the driver fetches access tokens.

The following examples show the connection information required to establish a session using the client credentials grant.

Connection URL

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake:AuthenticationMethod=OAuth2;
 AccountName=account_name.us-east-1;DatabaseName=payroll;Schema=xyz;
 Warehouse=accounting;ClientID=123456789;ClientSecret=FaZBFRsGXTaR;
 TokenURI=https://account_name.us-east-1.snowflakecomputing.com/oauth/token-request;");
```

Data Source

```
SnowflakeDataSource mds = new SnowflakeDataSource();
mds.setDescription("My Snowflake Data Source");
mds.setAuthenticationMethod("OAuth2");
mds.setAccountName("account_name.us-east-1");
mds.setDatabaseName("payroll");
mds.setSchema("xyz");
mds.setWarehouse("accounting");
mds.setClientID("123456789");
mds.setClientSecret("FaZBFRsGXTaR");
mds.setTokenURI("https://account_name.us-east-1.snowflakecomputing.com/oauth/token-request");
```

See also

Creating an OAuth security integration and obtaining client information on page 49
Connection property descriptions on page 67

Refresh token grant

The refresh token grant is used to request a new access token or renew an expired one by exchanging the refresh token at the endpoint specified by the TokenURI property.

To configure the driver to use the refresh token grant for OAuth 2.0 authentication:

- Set the AuthenticationMethod property to `OAuth2`.
- Set the AccountName property to specify the full name of your account and the region where it is hosted. For example, *`account_name`*`.us-east-1`.
- Set the DatabaseName property to specify the name of the database to which you are connecting.
- Set the Schema property to specify the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.
- Set the Warehouse property to specify the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.
- Set the ClientID property to specify the client ID for your application.
- Set the ClientSecret property to specify the client secret for your application.

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

- Set the TokenURI property to specify the endpoint from which the driver fetches access tokens.
- Set the RefreshToken property to specify the refresh token used to request a new access token or renew an expired one.

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

The following examples show the connection information required to establish a session using the refresh token grant.

Connection URL

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake:AuthenticationMethod=OAuth2;
 AccountName=account_name.us-east-1;DatabaseName=payroll;Schema=xyz;
 Warehouse=accounting;ClientID=123456789;ClientSecret=FaZBFRsGXTaR;
 TokenURI=https://account_name.us-east-1.snowflakecomputing.com/oauth/token-request;
 RefreshToken=abc12cd34efg5678h9ij87klm6543no32pqr10;");
```

Data Source

```
SnowflakeDataSource mds = new SnowflakeDataSource();
mds.setDescription("My Snowflake Data Source");
mds.setAuthenticationMethod("OAuth2");
mds.setAccountName("account_name.us-east-1");
mds.setDatabaseName("payroll");
mds.setSchema("xyz");
mds.setWarehouse("accounting");
mds.setClientID("123456789");
mds.setClientSecret("FaZBFRsGXTaR");
mds.setTokenURI("https://account_name.us-east-1.snowflakecomputing.com/oauth/token-request");
mds.setRefreshToken("abc12cd34efg5678h9ij87klm6543no32pqr10");
```

See also

Creating an OAuth security integration and obtaining client information on page 49
Obtaining access and refresh tokens using the Configuration Manager on page 54
Connection property descriptions on page 67

Single sign-on authentication

To configure the driver to use single sign-on (SSO) authentication.

- Set the AuthenticationMethod property to `BrowserBasedSSO`.
- Set the AccountName property to the account and region where it is hosted.
- Set the DatabaseName property to specify the name of the database to which you are connecting.
- Set the Schema property to specify the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.
- Set the Warehouse property to specify the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.
- Set the User property to provide the user ID.
- Set the credentials user name and password in the Active Directory page, if you are not logged in to Snowflake.

The following examples demonstrate how to make a connection using the single sign-on (SSO) authentication.

Connection URL:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake://snowflakecomputing.com;
  AuthenticationMethod=BrowserBasedSSO;AccountName=accountname.us-east-1;
  DatabaseName:payroll;Schema=xyz;Warehouse=accounting;User=jSmith@example.com;);
```

Data Source:

```
SnowflakeDataSource mds = new SnowflakeDataSource();
mds.setDescription("My Snowflake Data Source");
mds.setAuthenticationMethod("BrowserBasedSSO");
mds.setAccountName("accountname.us-east-1");
mds.setDatabaseName("payroll");
mds.setSchema("xyz");
mds.setWarehouse("accounting");
mds.setUser("jSmith@example.com");
```

Key-pair authentication

The driver supports key-pair authentication. Key-pair authentication allows you to authenticate to Snowflake using a pair of private and public keys. The keys used for authentication must be RSA keys that are at least 2048 bits long.

You can generate the private and public key files in Privacy Enhanced Mail (PEM) format using OpenSSL. While the public key must be assigned to your account on Snowflake, the private key can be specified using the driver. Note that only the users with the necessary privileges can assign public keys, such as users with the SECURITYADMIN role.

To learn how to generate private and public key files, refer to the Snowflake documentation.

To configure the driver to use key-pair authentication:

- Configure the basic connection properties used to establish a connection:
  - Set the AccountName property to specify the full name of your account and the region where it is hosted. For example, *account_name*.us-east-1.
  - Set the DatabaseName property to specify the name of the database to which you are connecting.
  - Set the Schema property to specify the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.
  - Set the Warehouse property to specify the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.
- Set the AuthenticationMethod property to `KeyPair`.
- Configure one of the following connection properties to specify the private key:
  - Set the PrivateKeyFile property to specify the absolute path to the private key file you want to use for authentication.
  - Set the PrivateKeyContent property to specify the content of the private key you want to use for authentication.
- If you are using an encrypted private key or private key content, set the PrivateKeyPassphrase property to specify the password for decrypting the private key or private key content you are using.

Note:

If the encryption schema you are using to generate the encrypted private keys is not compatible with the native encryption libraries of your JRE, the JRE will return an error. To resolve this issue, either generate encrypted private keys using an encryption schema that is compatible with your JRE or add a third-party Java cryptography library to your application (for example, Bouncy Castle) that supports the encryption schema you are using. For example, to add Bouncy Castle to your application, add the following line to the `java.security` file: `security.provider.`*`n`*`=org.bouncycastle.jce.provider.BouncyCastleProvider`, and then add the Bouncy Castle jars to the classpath of your JRE.

The following examples demonstrate how to make a connection using key-pair authentication.

Connection URL:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:snowflake:AccountName=account_name.us-east-1;
  DatabaseName=payroll;Schema=xyz;Warehouse=accounting;AuthenticationMethod=KeyPair;
  PrivateKeyFile=C:\Program Files\privatekey.p8;PrivateKeyPassphrase=abc123);
```

Data Source:

```
SnowflakeDataSource mds = new SnowflakeDataSource();
mds.setDescription("My Snowflake Data Source");
mds.setAccountName("account_name.us-east-1");
mds.setDatabaseName("payroll");
mds.setSchema("xyz");
mds.setWarehouse("accounting");
mds.setAuthenticationMethod("KeyPair");
mds.setPrivateKeyFile("C:\Program Files\privatekey.p8");
mds.setPrivateKeyPassphrase("abc123");
```

Data Encryption

All communication between the driver and Snowflake is encrypted using TLS/SSL.

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

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

ArrowFallbackToJson: ArrowFallbackToJson can be configured to allow the driver to fall the JSON query format when the arrow format is not properly initialized (`ArrowFallbackToJson= Enable | Warn`). By falling back to JSON query format, the driver can continue to execute queries that would normally return an exception; however, there is a significant performance tradeoff from not using the high-speed arrow transfer. For ideal performance, configure your JVM properly by specifying the `--add-opens=java.base/java.nio=ALL-UNNAMED` option value and disable this connection property (`ArrowFallbackToJson= Disable`).

FetchSize: FetchSize can be used to adjust the trade-off between throughput and response time. In general, setting larger values for FetchSize will improve throughput, but can reduce response time. You should set FetchSize to suit your environment. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes improve overall fetch times at the cost of additional memory.

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.

ReadAhead: The ReadAhead property allows you to issue multiple fetch requests in parallel. By increasing this number, you can improve throughput and performance, but it does so with the following restrictions:

- Larger values can increase the load on the server, which may adversely affect performance of other users. If you encounter issues, decrease the value specified for this option.
- Larger values may result in unnecessary requests if your application only requires the first few rows of results. This may be an issue if your service places limits on the number of web requests.

Caution: Due to potential impacts to other users, we strongly recommend specifying only smaller values for the ReadAhead property. For example, in fully optimized environments, which include exceptionally fast connections and low latency, we recommend a setting of no higher than `5`. For typical environments, this value is sometimes lower.

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:

- Using COPY command

Using COPY command

The driver supports using the COPY command for loading and unloading data from local file systems and cloud platforms, such as Amazon S3, Google Cloud, and Microsoft Azure. The data is moved from staged files to an existing table and vice versa.

Snowflake supports the following stages for loading and unloading of files.

- **User**: This stage is used for files that are accessed by a single user and copied into multiple tables.
- **Table**: This stage is used for files that are accessed by multiple users and copied into a single table.
- **Internal Named**: This stage is used for named database objects that provide flexibility and security during data loading.

The driver supports all the copy options and format type options supported by Snowflake. The process of data loading and unloading depends on your environment. Refer to the following topics in your Snowflake documentation for further details.

- Data Loading

  https://docs.snowflake.com/en/user-guide-data-load.html

- Data Unloading

  https://docs.snowflake.com/en/user-guide-data-unload.html

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.

General properties

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

Property	Data Source Method	Default
AccountName on page 75	`getAccountName()` `setAccountName(String)`	No default value
AuthenticationMethod on page 77	`getAuthenticationMethod()` `setAuthenticationMethod(String)`	`UserIDPassword`
DatabaseName on page 83	`getDatabaseName()` `setDatabaseName(String)`	No default value
Schema on page 103	`getSchema()` `setSchema(String)`	No default value
Warehouse on page 109	`getWarehouse()` `setWarehouse(String)`	No default value

UserID/Password authentication properties

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

Property	Data Source Method	Default
Password on page 92	`getPassword()` `setPassword(String)`	No default value
User on page 108	`getUser()` `setUser(String)`	No default value

OAuth 2.0 authentication properties

The following table summarizes properties used for OAuth 2.0 authentication method.

Property	Data Source Method	Default
AccessToken on page 74	`getAccessToken()` `setAccessToken(String)`	No default value
AuthURI on page 78	`getAuthUri()` `setAuthUri(String)`	No default value
ClientID on page 79	`getClientId()` `setClientId(String)`	No default value
ClientSecret on page 79	`getClientSecret()` `setClientSecret(String)`	No default value
OAuthCode on page 90	`getCode()` `setCode(String)`	No default value
RedirectURI on page 100	`getRedirUri()` `setRedirUri(String)`	No default value
RefreshToken on page 101	`getRefreshToken()` `setRefreshToken(String)`	No default value
Scope on page 104	`getScope()` `setScope(String)`	No default value
TokenURI on page 107	`getTokenUri()` `setTokenUri(String)`	No default value

Key-pair authentication properties

The following table summarizes the properties used for key-pair authentication method.

Property	Data Source Method	Default
AuthenticationMethod on page 77	`getAuthenticationMethod()` `setAuthenticationMethod(String)`	`UserIDPassword`
PrivateKeyContent on page 92	`getPrivateKeyContent()` `setPrivateKeyContent(String)`	No default value

Property	Data Source Method	Default
PrivateKeyFile on page 93	`getPrivateKeyFile()` `setPrivateKeyFile(String)`	No default value
PrivateKeyPassphrase on page 94	`getPrivateKeyPassphrase()` `setPrivateKeyPassphrase(String)`	No default value

Proxy server properties

The following table summarizes proxy server connection properties.

Property	Data Source Method	Default
ProxyHost on page 95	`getProxyHost()` `setProxyHost(String)`	No default value
ProxyPassword on page 96	`getProxyPassword()` `setProxyPassword(String)`	No default value
ProxyPort on page 96	`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 97	`getProxyUser()` `setProxyUser(String)`	No default value

Data type properties

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

Table 3: Data Type Properties

Property	Data Source Method	Default
ConvertNull on page 82	`getConvertNull()` `setConvertNull(Boolean)`	`true`

Timeout properties

The following table summarizes timeout connection properties.

Property	Data Source Method	Default
LoginTimeout	`getLoginTimeout()` `setLoginTimeout(Integer)`	`60`
QueryTimeout	`getQueryTimeout()` `setQueryTimeout(Integer)`	`0`

Statement pooling properties

The following table summarizes statement pooling connection properties.

Table 4: Statement Pooling Properties

Property	Data Source Method	Default
ImportStatementPool on page 85	`getImportStatementPool()` `setImportStatementPool(String)`	No default value
MaxPooledStatements on page 89	`getMaxPooledStatements()` `setMaxPooledStatements(Integer)`	`0`
RegisterStatementPoolMonitorMBean on page 101	`getRegisterStatementPoolMonitorMbean()` `setRegisterStatementPoolMonitorMbean(Boolean)`	`false`

Mapping properties

The following table summarizes mapping properties.

Property	Data Source Method	Default
IntegerFieldMapping	`getIntegerFieldMapping()` `setIntegerFieldMapping(String)`	`Numeric`
KeywordConflictSuffix	`getKeywordConflictSuffix()` `setKeywordConflictSuffix(String)`	No default value
Schema	`getSchema()` `setSchema(String)`	No default value

Additional properties

The following table summarizes additional connection properties.

Property	Data Source Method	Default
ArrowFallbackToJson	`public String getArrowFallbackToJson()` `public void setArrowFallbackToJson(String)`	`Enable`
ClientSessionKeepAlive	`getClientSessionKeepAlive()` `setClientSessionKeepAlive(Boolean)`	`false`
ConnectionRetryCount	`getConnectionRetryCount()` `setConnectionRetryCount(Integer)`	`5`
ConnectionRetryDelay	`getConnectionRetryDelay()` `setConnectionRetryDelay(Integer)`	`1`
DisableSOCKSProxy (DSP)	`getDisableSocksProxy()` `setDisableSocksProxy(String)`	`false`
FetchSize on page 84	`getFetchSize()` `setFetchSize(Integer)`	`100` (rows)
InsensitiveResultSetBufferSize on page 86	`getInsensitiveResultsetBufferSize()` `setInsensitiveResultsetBufferSize(Integer)`	`2048`
LogConfigFile on page 88	`getLogConfigFile()` `setLogConfigFile(String)`	`ddlogging.properties`
PartnerApplicationName on page 91	`getPartnerApplicationName()` `setPartnerApplicationName(String)`	No default value
ReadAhead on page 98	`getReadAheadThreads()` `setReadAheadThreads(Integer)`	`5`
ReadOnly on page 99	`getReadOnly()` `setReadOnly(Boolean)`	`false`
Schema	`getSchema()` `setSchema(String)`	None
SpyAttributes	`getSpyAttributes()` `setSpyAttributes(String)`	No default value

Property	Data Source Method	Default
UseSessionDatabaseForMetadata	getUseSessionDatabaseForMetadata() setUseSessionDatabaseForMetadata(boolean)	`false`
Warehouse	`getWarehouse()` `setWarehouse(String)`	No default value

For details, see the following topics:

- AccessToken
- AccountName
- ArrowFallbackToJson
- AuthenticationMethod
- AuthURI
- ClientID
- ClientSecret
- ClientSessionKeepAlive
- ConnectionRetryCount
- ConnectionRetryDelay
- ConvertNull
- DatabaseName
- DisableSocksProxy
- FetchSize
- ImportStatementPool
- InsensitiveResultSetBufferSize
- IntegerFieldMapping
- KeywordConflictSuffix
- LogConfigFile
- LoginTimeout
- MaxPooledStatements
- OAuthCode
- PartnerApplicationName
- Password
- PrivateKeyContent
- PrivateKeyFile

AccessToken

Purpose

Specifies the access token used to authenticate to Snowflake with OAuth 2.0 enabled. Typically, this property is configured by the application; however, in some scenarios, you may need to secure a token using external processes. In those instances, you can also use this property to set the access token manually.

Valid Values

`String`

where:

`String`

is an access token you have obtained from the authentication service.

Notes

- Access tokens expire ten minutes after generation. Once connected, the access token remains valid till the session is disconnected.

- See "OAuth 2.0 authentication" for examples and more information.

Data Source Methods

`public String getAccessToken()`

`public void setAccessToken(String)`

Default Value

No default value

Data Type

String

AccountName

Purpose

Specifies the full name of your account and the region where it is hosted. The full account name may include additional segments that denote region and cloud platform.

Valid Values

`string`

where:

`string`

is the full name of an account with region and cloud platform.

Example

Table 5: AccountName by region:

Cloud Platform/Region	Full Account Name
AWS	
US East (N.Virginia)	*`account_name.us-east-1`*
US East (Ohio)	*`account_name.us-east-2.aws`*
US West (Oregon)	*`account_name`*
Canada (Central)	*`account_name.ca-central-1.aws`*
EU (Ireland)	*`account_name.eu-west-1`*
EU (Frankfurt)	*`account_name.eu-central-1`*
Asia Pacific (Singapore)	*`account_name.ap-southeast-1`*

Cloud Platform/Region	Full Account Name
Asia Pacific (Sydney)	*account_name.ap-southeast-2*
Asia Pacific (Tokyo)	*account_name.ap-northeast-1.aws*
Azure	
East US 2	*account_name.east-us-2.azure*
West US 2	*account_name.west-us-2.azure*
US Gov Virginia	*account_name.us-gov-virginia.azure*
Canada Central	*account_name.canada-central.azure*
West Europe	*account_name.west-europe.azure*
Australia East	*account_name.australia-east.azure*
Southeast Asia	*account_name.southeast-asia.azure*

Data Source Methods

```
public String getAccountName()
```

```
public void setAccountName(String)
```

Default Value

No default value

Data Type

String

ArrowFallbackToJson

Purpose

Specifies whether the driver uses the JSON query format when the arrow format is not properly initialized. The driver uses a high-speed arrow transfer that requires the restricted APIs from java.nio package. For JVMs that are Java SE 16 (or equivalent) and higher, the following JVM value must be set to use arrow transfer:

```
--add-opens=java.base/java.nio=ALL-UNNAMED
```

If this value is not specified, the API throws an `InaccessibleObjectException` during query execution. To allow you to continue executing queries in this scenario, you can set this property to configure the driver to fall back to the JSON query format when the JVM is not properly configured.

Valid Values

`Enable` | `Warn` | `Disable`

Behavior

If set to `Enable`, the driver falls back to JSON query format if the arrow APIs can not be initialized using Java SE 16 and higher, and a CONFIG-level message is logged.

If set to `Warn`, the driver falls back to JSON query format if the arrow APIs can not be initialized using Java SE 16 and higher, and a WARN-level message is logged.

If set to `Disable`, the driver does not fall back to JSON query format if arrow APIs can not be initialized, and the driver throws a `SQLException` when executing a `SELECT` query that uses arrow transfer.

Notes

- The recommended practice is to configure the JVM with the `--add-opens=java.base/java.nio=ALL-UNNAMED` option value and disable this connection property. This configuration provides the best performance.
- Alternatively, your application can set JSON query format for the session by executing the `ALTER SESSION SET JDBC_QUERY_RESULT_FORMAT='JSON'` statement.
- This connection property can affect performance.

Data Source Methods

```
public String getArrowFallbackToJson()
```

```
public void setArrowFallbackToJson(String)
```

Default Value

`Enable`

Data Type

String

AuthenticationMethod

Purpose

Specifies the authentication method to use for verifying user login credentials.

Valid Values

`UserIDPassword` | `OAuth2` | `BrowserBasedSSO` | `KeyPair`

Behavior

If set to `UserIDPassword`, the driver uses your user name and password for authentication.

If set to `OAuth2`, the driver authenticates using OAuth. When OAuth2 is specified as the authentication method, you must also set the accessToken parameter to specify the OAuth access token.

If set to `BrowserBasedSSO`, the driver uses your web browser to authenticate with Okta, ADFS, or any other SAML 2.0-compliant identity provider (IdP) that has been defined for your account.

If set to `KeyPair`, the driver uses a pair of private and public keys for authentication.

Data Source Methods

```
public String getAuthenticationMethod()
```

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

Default Value

`UserIDPassword`

Data Type

String

See also

Authentication on page 48

AuthURI

Purpose

Specifies the endpoint for obtaining an authorization code from a third-party authorization service for OAuth 2.0 implementations.

Valid Values

`String`

where:

`String`

is the endpoint for retrieving the OAuth 2.0 authorization code from the third party authorization service.

Notes

- When this endpoint is queried, the authorization service presents an interface prompting the user to approve or deny access to backend data.
- See "OAuth 2.0 authentication" for examples and more information.

Data Source Methods

```
public String getAuthUri()
```

```
public void setAuthUri(String)
```

Default Value

No default value

Data Type

String

ClientID

Purpose

Specifies the client ID key for your application when authenticating to Snowflake with OAuth 2.0 enabled.

Valid Values

String

where:

String

is the client ID key for your application.

Notes

See "OAuth 2.0 authentication" for more information.

Data Source Methods

```
public String getClientId()
```

```
public void setClientId(String)
```

Default Value

No default value

Data Type

String

ClientSecret

Purpose

Specifies the client secret for your application when authenticating to Snowflake with OAuth 2.0 enabled.

Important: The client secret is a confidential value used to authenticate the application to the service. To prevent unauthorized access, this value must be securely maintained.

Valid Values

String

where:

String

is the client secret for your application.

Notes

See "OAuth 2.0 authentication" for more information.

Data Source Methods

```
public String getClientSecret()
```

```
public void setClientSecret(String)
```

Default Value

No default value

Data Type

String

ClientSessionKeepAlive

Purpose

Determines whether the session is closed after four hours of inactivity.

Valid Values

`true` | `false`

Behavior

If set to `true`, the session remains active indefinitely, even if there is no activity from the user.

If set to `false`, the session is closed after four hours of inactivity. The user must re-authenticate to start a new session.

Notes

Closing the session during periods of inactivity causes the warehouse to consume less credits.

Data Source Methods

```
public Boolean getClientSessionKeepAlive()
```

```
public void setClientSessionKeepAlive(Boolean)
```

Default Value

`false`

Data Type

Boolean

ConnectionRetryCount

Purpose

The number of times the driver retries connection attempts to Snowflake 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

`true` | `false`

Behavior

If set to `true`, 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`.

If set to `false`, 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.

Data Source Methods

```
public Boolean getConvertNull()
```

```
public void setConvertNull(Boolean)
```

Default Value

`true`

Data Type

Boolean

DatabaseName

Purpose

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

Valid Values

`database_name`

where:

`database_name`

is the name of a valid database.

Important: The value is case-insensistive if you have access privileges to query the list of databases on the server. If you do not have access, the value is case-sensitive.

Data Source Methods

```
public String getDatabaseName()
```

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

Default Value

No default value

Data Type

String

DisableSocksProxy

Purpose

Specifies whether the driver should ignore the SOCKS proxy configuration specified in the Java system options (if any). Setting this connection property alters the behavior of all the connections on the same JVM.

Valid Values

`true` | `false`

Behavior

If set to `false`, the driver uses the SOCKS proxy.

If set to `true`, the driver ignores the SOCKS proxy.

Notes

This property is used when you want the driver to ignore the existing Java proxy configuration on your client for Snowflake connections.

Data Source Methods

```
public Boolean getDisableSocksProxy()
```

```
public void setDisableSocksProxy(Boolean)
```

Default Value

`false`

Data Type

Boolean

FetchSize

Purpose

Specifies the maximum number of rows that the driver processes before returning data to the application when executing a Select. This value provides a suggestion to the driver as to the number of rows it should internally process before returning control to the application. The driver may fetch fewer rows to conserve memory when processing exceptionally wide rows.

Valid Values

`0` | *x*

where:

x

is a positive integer indicating the number of rows that should be processed.

Behavior

If set to `0`, the driver processes all the rows of the result before returning control to the application. When large data sets are being processed, setting FetchSize to `0` can diminish performance and increase the likelihood of out-of-memory errors.

If set to *x*, the driver limits the number of rows that may be processed for each fetch request before returning control to the application.

Notes

- To optimize throughput and conserve memory, the driver uses an internal algorithm to determine how many rows should be processed based on the width of rows in the result set. Therefore, the driver may process fewer rows than specified by FetchSize when the result set contains exceptionally wide rows. Alternatively, the driver processes the number of rows specified by FetchSize when the result set contains rows of unexceptional width.
- FetchSize can be used to adjust the trade-off between throughput and response time. Smaller fetch sizes can improve the initial response time of the query. Larger fetch sizes can improve overall response times at the cost of additional memory.
- You can use FetchSize to reduce demands on memory and decrease the likelihood of out-of-memory errors. Simply, decrease FetchSize to reduce the number of rows the driver is required to process before returning data to the application.

Data Source Methods

```
public Integer getFetchSize()
```

```
public void setFetchSize(Integer)
```

Default Value

`100`

Data Type

Integer

See also

Performance considerations on page 63

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

No default value

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

IntegerFieldMapping

Purpose

Specifies the data type mapping for fixed-point numbers where precision and scale cannot be specified. These include BIGINT, INTEGER, and synonymous types. Refer to your Snowflake documentation for details.

Valid Values

BigInt | Numeric

Behavior

If set to BigInt, the driver maps fixed-point numbers where precision and scale cannot be specified to BigInt.

If set to Numeric, the driver maps fixed-point numbers where precision and scale cannot be specified to Numeric.

Data Source Methods

```
public String getIntegerFieldMapping()
```

```
public void setIntegerFieldMapping(String)
```

Default Value

Numeric

Data Type

String

KeywordConflictSuffix

Purpose

Specifies a string of up to 5 alphanumeric characters that the driver appends to any object or field name that conflicts with a SQL engine keyword.

Valid Values

String

where:

String

is a string of up to 5 alphanumeric characters.

Example

A field called CASE exists in the data schema. To avoid a naming conflict with the SQL engine keyword CASE, you could set KeywordConflictSuffix=TAB. In this scenario, the driver maps the CASE field to the CASETAB column.

Data Source Methods

```
public String getKeywordConflictSuffix()
```

```
public void setKeywordConflictSuffix(String)
```

Default Value

No default value

Data Type

String

LogConfigFile

Purpose

Specifies the file name, and optionally, the path of the properties file used to initialize driver logging.

Valid Values

String

where:

String

is the relative or fully qualified path of the properties file to load to initialize driver logging. If you do not specify a path, the driver looks for this file in the current working directory. If the specified file does not exist, the driver continues searching for an appropriate properties file as described in "Using Java Logging" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public String getLogConfigFile()
```

```
public void setLogConfigFile(String)
```

Default Value

ddlogging.properties

Data Type

String

LoginTimeout

Purpose

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

Valid Values

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

where:

x

is a positive integer that specifies a number of seconds.

Behavior

If set to `-1` | `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 Methods

```
public Integer getLoginTimeout()
```

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

Default Value

`60`

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

OAuthCode

Purpose

Specifies the temporary authorization code that is exchanged for access tokens when OAuth 2.0 authentication is enabled.

Valid Values

`String`

where:

`String`

is an OAuth 2.0 authorization code.

Notes

- This property is configured by the application.
- See "OAuth 2.0 authentication" for more information.

Data Source Methods

```
public String getCode()
```

```
public void setCode(String)
```

Default Value

No default value

Data Type

String

PartnerApplicationName

Purpose

Snowflake partner use only.

Specifies the name of a partner application to which you are trying to connect. This property is useful for users who have an existing partner contract with Snowflake and are using the driver with a Snowflake partner application or plug-in.

Valid Values

String

where:

String

is a valid partner application name string.

Data Source Methods

```
public String getPartnerApplicationName()
```

```
public void setPartnerApplicationName(String)
```

Default Value

No default value

Data Type

String

Password

Purpose

A password that is used to connect to the service.

Important: Setting the password using a data source is not recommended. The data source persists all properties, including password, in clear text.

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 108

PrivateKeyContent

Purpose

Specifies the content of a private key. It is used to authenticate to Snowflake when key-pair authentication is enabled (`AuthenticationMethod=KeyPair`).

Valid Values

String

where:

String

is the content of a private key.

Notes

- When specifying private key content, use the following syntax:
  - For encrypted private key content: `-----BEGIN ENCRYPTED PRIVATE KEY-----`*`private key content`*`-----END ENCRYPTED PRIVATE KEY-----`
  - For unencrypted private key content: `-----BEGIN PRIVATE KEY-----`*`private key content`*`-----END PRIVATE KEY-----`
- If the encryption schema you are using to generate the encrypted private keys is not compatible with the native encryption libraries of your JRE, the JRE will return an error. To resolve this issue, either generate encrypted private keys using an encryption schema that is compatible with your JRE or add a third-party Java cryptography library to your application (for example, Bouncy Castle) that supports the encryption schema you are using. For example, to add Bouncy Castle to your application, add the following line to the `java.security` file: `security.provider.`*`n`*`=org.bouncycastle.jce.provider.BouncyCastleProvider`, and then add the Bouncy Castle jars to the classpath of your JRE.
- The private key is a confidential value. To prevent unauthorized access, this value must be securely maintained.
- The private key content you specify must correspond to the content of the public key assigned to your Snowflake account.

Data Source Methods

```
public String getPrivateKeyContent()
```

```
public void setPrivateKeyContent(String)
```

Default Value

No default value

Data Type

String

PrivateKeyFile

Purpose

Specifies the absolute path to the private key file used to authenticate to Snowflake when key-pair authentication is enabled (`AuthenticationMethod=KeyPair`).

Valid Values

`String`

where:

`String`

is the the absolute path to the private key file.

Notes

- If the encryption schema you are using to generate the encrypted private keys is not compatible with the native encryption libraries of your JRE, the JRE will return an error. To resolve this issue, either generate encrypted private keys using an encryption schema that is compatible with your JRE or add a third-party Java cryptography library to your application (for example, Bouncy Castle) that supports the encryption schema you are using. For example, to add Bouncy Castle to your application, add the following line to the `java.security` file: `security.provider.`*`n`*`=org.bouncycastle.jce.provider.BouncyCastleProvider`, and then add the Bouncy Castle jars to the classpath of your JRE.
- The private key is a confidential value. To prevent unauthorized access, this value must be securely maintained.
- The private key contained in the private key file you specify must correspond to the public key assigned to your Snowflake account.

Data Source Methods

```
public String getPrivateKeyFile()
```

```
public void setPrivateKeyFile(String)
```

Default Value

No default value

Data Type

String

PrivateKeyPassphrase

Purpose

Specifies the password used to decrypt the encrypted private key. The private key is used to authenticate to Snowflake when key-pair authentication is enabled (`AuthenticationMethod=KeyPair`).

Valid Values

`String`

where:

`String`

is the password used to decrypt the encrypted private key.

Notes

- This property can be used with both the PrivateKeyFile and PrivateKeyContent connection properties, to decrypt either a private key or the content of a private key.
- Both private key and private key password are confidential values. To prevent unauthorized access, these values must be securely maintained.

Data Source Methods

`public String getPrivateKeyPassphrase()`

`public void setPrivateKeyPassphrase(String)`

Default Value

No default value

Data Type

String

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

Proxy server on page 22
ProxyPassword on page 96
ProxyPort on page 96
ProxyUser on page 97

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

ProxyHost on page 95
ProxyPort on page 96
ProxyUser on page 97
Proxy server on page 22

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

Proxy server on page 22
ProxyHost on page 95
ProxyPassword on page 96
ProxyUser on page 97

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

ProxyHost on page 95

ProxyPassword on page 96
ProxyPort on page 96
Proxy server on page 22

QueryTimeout

Purpose

Sets the default query timeout (in seconds) for all statements created by a connection.

Valid Values

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

where:

x

is a number of seconds.

Behavior

If set to `-1`, the query timeout functionality is disabled. The driver silently ignores calls to the Statement.setQueryTimeout() method.

If set to `0`, the default query timeout is infinite (the query does not time out).

If set to *x*, the driver uses the value as the default timeout for any statement that is created by the connection. To override the default timeout value that is set by this property, call the Statement.setQueryTimeout() method to set a timeout value for a particular statement.

Data Source Methods

```
public Integer getQueryTimeout()
```

```
public void setQueryTimeout(Integer)
```

Default Value

`0`

Data Type

Integer

ReadAhead

Purpose

Specifies the maximum number of fetch requests the driver issues in parallel. By default, the driver queues the next page when processing the current page. This property allows you to fetch multiple requests simultaneously, thereby improving throughput and performance.

Caution: Due to potential impacts to other users on the network, we strongly recommend specifying only smaller values for this property. For example, in fully optimized environments, which include exceptionally fast connections and low latency, we recommend a setting of no higher than `10`. For typical environments, this value should be considerably lower.

Valid Values

`0` | *x*

where:

x

is the maximum number of fetch requests the driver issues in parallel up to 100.

Behavior

If set to `0`, the driver queues the next page while processing the current page.

If set to *x*, the driver executes fetch requests as they are issued until the number of active parallel-requests equals the specified value. When that threshold is met, the driver waits until the results of a request are processed before requesting the next page of data.

Notes

- Specifying larger values for this property generally improves performance; however, with the following warnings:
  - Larger values can increase the load on the server, which may adversely affect performance of other users. If you encounter issues, decrease the value specified for this property.
  - Larger values may result in unnecessary requests if your application only requires the first few rows of results. This may be an issue if your service places limits on the number of web requests.

Data Source Methods

```
public Integer getReadAheadThreads()
```

```
public void setReadAheadThreads(Integer)
```

Default Value

`0`

Data Type

Integer

ReadOnly

Purpose

Specifies whether the connection has read-only access to the data source.

Valid Values

`true` | `false`

Behavior

If set to `true`, the connection has read-only access.

If set to `false`, the connection is opened for read/write access, and you can use all commands supported by the product.

Notes

You can use use the JDBC connection method setReadOnly to set a read-only state for a connection.

Data Source Methods

```
public Boolean getReadOnly()
```

```
public void setReadOnly(Boolean)
```

Default Value

`false`

Data Type

Boolean

RedirectURI

Purpose

Specifies the endpoint to which the client is returned after third-party authorization for OAuth 2.0 implementations.

Valid Values

`String`

where:

`String`

is the endpoint to which the client is returned after third-party authorization. For example, `http://localhost`.

Notes

- The redirect URI is often registered with the authentication service to provide improved security. Registering the endpoint prevents your valid authentication credentials being redirected to a malicious site; therefore, reducing the risk of sharing your access token and other sensitive information with unauthorized parties.
- See "OAuth 2.0 authentication" for examples and more information.

Data Source Methods

```
public String getRedirUri()
```

```
public void setRedirUri(String)
```

Default Value

No default value

Data Type

String

RefreshToken

Purpose

Specifies the refresh token used to either request a new access token or renew an expired access token for OAuth 2.0 implementations.

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

Valid Values

String

where:

String

is the refresh token you have obtained from the Snowflake service.

Notes

- See "OAuth 2.0 authentication" for more information.

Data Source Methods

```
public String getRefreshToken()
```

```
public void setRefreshToken(String)
```

Default Value

No default value

Data Type

String

See also

Refresh token grant on page 59

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

RoleName

Purpose

Specifies the default role to use for access control in the Snowflake session initiated by the driver. The specified role should be an existing role that has already been assigned to the specified user. If the specified role has not already been assigned to the user, the role is not used when the session is initiated by the driver.

Valid Values

`string`

where:

`string`

is name of the default role to use for access control.

Notes

- The USE ROLE command can be executed to set a different role for the session.
- The Role property is an alias for the RoleName property.

Data Source Methods

```
public String getRoleName()
```

```
public void setRoleName(String)
```

Default Value

No default value

Data Type

String

Schema

Purpose

Specifies the default schema to use for the specified database once connected. The specified schema should be an existing schema for which the specified default role has privileges.

After connecting, the USE SCHEMA command can be executed to set a different schema for the session.

Valid Values

String

where:

String

is a valid schema name.

Data Source Methods

```
public String getSchema()
```

```
public void setSchema(String)
```

Default Value

No default value

Data Type

String

See also

RoleName on page 102

Scope

Purpose

Specifies a space-separated list of OAuth scopes that limit the permissions granted by an access token.

Valid Values

String

where:

String

is a space-separated list of security scopes.

Data Source Methods

```
public String getScope()
```

```
public void setScope(String)
```

Default Value

No default value

Data Type

String

SpyAttributes

Purpose

Enables DataDirect Spy to log detailed information about calls that are 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 no.

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)
```

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.
- Refer to "Tracking JDBC Calls with DataDirect Spy" in the *Progress DataDirect for JDBC Drivers Reference* for a list of supported attributes.

Data Source Methods

```
public String getSpyAttributes()
```

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

Default

No default value

Data Type

String

TokenURI

Purpose

Specifies the endpoint for retrieving access tokens when OAuth 2.0 authentication is enabled.

Valid Values

String

where:

String

is the endpoint used to retrieve access tokens.

Notes

See "OAuth 2.0 authentication" for more information.

Data Source Methods

`public String getTokenUri()`

`public void setTokenUri(String)`

Default Value

No default value

Data Type

String

UseSessionDatabaseForMetadata

Purpose

Specifies whether the driver fetches metadata for only tables in the database to which you are connected when a database name is not specified in metadata calls. When enabled, this property can provide better performance for metadata calls by reducing the number of tables queried.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver fetches metadata for only tables in the database to which you are connected.

If set to `false`, the driver fetches metadata for all tables in all databases.

Notes

- When the database name is specified in the metadata calls, the driver ignores the value specified for this connection property and returns metadata only for the database specified in the metadata calls.

Data Source Methods

```
public Boolean getUseSessionDatabaseForMetadata()
```

```
public void setUseSessionDatabaseForMetadata(Boolean)
```

Default Value

`false`

Data Type

Boolean

User

Purpose

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

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 92

Warehouse

Purpose

Specifies the virtual warehouse to use once connected. The specified warehouse should be an existing warehouse for which the specified default role has privileges.

After connecting, the USE WAREHOUSE command can be executed to set a different warehouse for the session.

Valid Values

String

where:

String

is a valid warehouse string.

Data Source Methods

```
public String getWarehouse()
```

```
public void setWarehouse(String)
```

Default Value

No default value

Data Type

String

See also

RoleName on page 102

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=SNOWFLAKE`
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 snowflake.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 0`*`number`*.

`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 129

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 127

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 129
Subqueries on page 127

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 129
Subqueries on page 127

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 129
Subqueries on page 127

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 129

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`