Progress DataDirect for JDBC for PostgreSQL

User's Guide

Release 6.0.0

Copyright

Table of Contents

Welcome to the Progress DataDirect for JDBC for PostgreSQL Driver...9

What's new in this release?....10
Requirements....12
Installing and setting up the driver....12
Driver and DataSource classes....14
Connection URL examples....14
Data types....16
getTypeInfo()....18
SQL escape sequences....33
Supported scalar functions....34
DataDirect tools....35
Troubleshooting....35
Additional information....35
Contacting Technical Support....36

Tutorials....37

Tableau....37
DbVisualizer....38
Adding a driver....38
Connecting and executing SQL statements....39

Configuring and connecting....41

Setting the classpath....42
Connecting using the JDBC Driver Manager....42
Passing the connection URL....42
Testing the connection....43
Connecting using data sources....46
How data sources are implemented....46
Creating data sources....47
Calling a data source in an application....48
Testing a data source connection....48
Authentication....51
User ID and password authentication....51
Kerberos authentication....52
Microsoft Entra ID authentication....56
AWS IAM authentication....57
Data encryption....59
TLS/SSL encryption....59

Using client information....62
Bulk load....62
Performance considerations....63

Additional features and functionality 65

Statement pooling....66
Connection pooling....66
Failover....66
Configuring failover....67
Refcursors....69
Returning and inserting/updating XML data....69
Returning XML data....69
Inserting/updating XML data....69
Isolation levels....71
Using scrollable cursors....71
JTA support....71
Batch inserts....71
Large object (LOB) support....71
Using COPY command....72
Parameter metadata support....73
User-defined function results....73
ResultSet metadata support....75
Rowset support....75
Auto-generated keys support....76

Connection property descriptions....77

AccessKey....87
AccountingInfo....88
AlternateServers....88
ApplicationName....89
AuthenticationMethod....90
AzureTenantID....91
BatchMechanism....91
BulkLoadBatchSize....93
CallEscapeBehavior....94
CatalogOptions....94
ClientHostName....95
ClientUser....96
CodePageOverride....96
ConnectionRetryCount....97
ConnectionRetryDelay....98
ConvertNull....99
CryptoProtocolVersion....99

DatabaseName....101
EnableCancelTimeout....101
EnablePrepareThreshold....102
EncryptionMethod....103
ExtendedColumnMetadata....104
HostNameInCertificate....104
ImportStatementPool....105
InitializationString....106
InsensitiveResultSetBufferSize....107
JavaDoubleToString....108
KeyPassword....108
KeyStore....109
KeyStorePassword....110
LoadBalancing....111
LoginConfigName....111
LoginTimeout....112
MaxLongVarcharSize....113
MaxNumericPrecision....113
MaxNumericScale....114
MaxPooledStatements....115
MaxStatements....116
MaxVarcharSize....116
Password....117
PortNumber....117
PrepareThreshold....118
ProgramID....119
ProxyPassword....120
ProxyPort....120
ProxyUser....121
ProxyHost....122
QueryTimeout....123
Region....123
RegisterStatementPoolMonitorMBean....124
ResultSetMetaDataOptions....125
SecretKey....126
ServerName....126
ServicePrincipalName....127
SpyAttributes....128
SupportsCatalogs....130
TransactionErrorBehavior....131
TrustStore....132
TrustStorePassword....133
User....134
ValidateServerCertificate....134
VarcharClobThreshold....135

1

Welcome to the Progress DataDirect for JDBC for PostgreSQL Driver

The Progress® DataDirect® for JDBC™ for PostgreSQL™ driver (PostgreSQL driver) supports the JDBC API for SQL read-write access to:

- Google AlloyDB for PostgreSQL
- Amazon Aurora PostgreSQL
- EnterpriseDB (EDB)
- PostgreSQL Database
- Yellowbrick

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-postgresql.

For details, see the following topics:

- What's new in this release?
- Requirements
- Installing and setting up the driver
- Driver and DataSource classes

- Connection URL examples
- Data types
- SQL escape sequences
- DataDirect tools
- Troubleshooting
- Additional information
- Contacting Technical Support

What's new in this release?

Support and certification

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

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

Changes Since 6.0.0

- **Enhancements**
  - The driver has been enhanced to support AWS IAM authentication. When AWS IAM authentication is enabled (`AuthenticationMethod=AWSIAM`), you can connect with AWS IAM using the new AccessKey, Region, and SecretKey properties. See AWS IAM authentication on page 57 for details.
  - The driver supports the Geography and Geometry data types with the `getColumns()` method.

    Feature details:

    - Available: Driver version 6.0.0.001767 (F002919.U001917)

  - 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 the SCRAM-SHA-256-PLUS password hashing mechanism for user ID/password authentication. The SCRAM-SHA-256-PLUS mechanism uses channel binding to establish a secure connection with PostgreSQL (v11.0 and higher).
  - The driver has been enhanced to support Microsoft Entra ID authentication (formerly Azure Active Directory Authentication). Entra ID authentication is an alternate authentication type that allows administrators to centrally manage user permissions to Azure SQL Database data stores. The driver supports the following methods of Entra ID authentication:

    - User and password authentication
    - Service principal authentication

    See Microsoft Entra ID authentication on page 56 for details.

- The driver has been enhanced to support the TLSv1.3 cryptographic protocol. See CryptoProtocolVersion on page 99 for details.
- The driver has been enhanced to support MERGE against PostgreSQL 15 and later. MERGE is a SQL command that modifies rows in the target table using the data from the source table. It can conditionally INSERT, UPDATE or DELETE rows in a single statement, eliminating the need to write multiple procedural statements.

  Feature details:

  - Available: Driver version 6.0.0.001277 (F002097.U001111)

- The driver has been enhanced with the new EnablePrepareThreshold and PrepareThreshold connection properties, which allow you to configure the behavior of server-side prepared statements. See EnablePrepareThreshold on page 102 and PrepareThreshold on page 118 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 128 for details.

Highlights of 6.0.0 Release

- The driver has been enhanced to support stored procedures for PostgreSQL 11 and later.
- The CallEscapeBehavior connection property has been added to the driver. It determines whether the driver calls a user-defined function or a stored procedure when JDBC Call escape syntax is used in a SQL statement. See CallEscapeBehavior on page 94 for details.
- The driver has been enhanced to support the following types of data types:
  - Network address types
  - Monetary types
  - Date/time types
  - Range types

  Note: For more information, see Data types on page 16.

- The driver has been enhanced to support the TXID_SNAPSHOT data type.
- The driver has been enhanced to support the SCRAM-SHA-256 authentication method, which uses a hashing mechanism for encrypting passwords to establish a secure connection with PostgreSQL (v10.0 and higher). This method requires a Java Virtual Machine (JVM) that is Java SE 8 or higher. During connection, the driver detects and uses the most secure method supported by the server. You must also provide a value for the User and Password connection properties.
- The driver has been enhanced to include timestamp in the Spy and JDBC packet logs by default. If required, you can disable the timestamp logging by specifying the following at connection: For Spy logs, set `spyAttributes=(log=(file)Spy.log;timestamp=no)` and for JDBC packet logs, set `ddtdbg.ProtocolTraceShowTime=false`.
- Interactive SQL for JDBC (JDBCISQL) is now installed with the product. JDBCISQL is a command-line interface that supports connecting your driver to a data source, executing SQL statements and retrieving results in a terminal. This tool provides a method to quickly test your drivers in an environment that does not support GUIs.

- The driver has been enhanced to support a customized version of the PostgreSQL COPY command. It provides an additional keyword, LOCALFILE, to allow you to copy data from or to standard file-system files that are stored anywhere on your network, not just on the database server. See Using COPY command on page 72 for details.
- The driver has been enhanced to support the PostgreSQL COPY command for batch inserts. When BatchMechanism is set to copy, the driver leverages the PostgreSQL COPY command for substantial performance gains.
- The driver has been enhanced to support retrieving the values of auto-generated keys. See Auto-generated keys support on page 76 for details.
- The driver has been enhanced to support the following data types: Citext, JSON, JSONB, and UUID.
- The driver has been enhanced to support the Kerberos authentication protocol with the following connection properties:
  - AuthenticationMethod
  - ServicePrincipalName

  See Kerberos authentication on page 52 for details.
- The ExtendedColumnMetadata connection property has been added to the driver. This property determines how the driver returns column metadata when retrieving results with ResultSetMetaData methods. See ExtendedColumnMetadata on page 104 for details.
- The RegisterStatementPoolMonitorMBean connection property has been added. Note that the driver no longer registers the Statement Pool Monitor as a JMX MBean by default. You must set RegisterStatementPoolMonitorMBean to `true` to register the Statement Pool Monitor and manage statement pooling with standard JMX API calls. See RegisterStatementPoolMonitorMBean on page 124 for details.

Changed behavior

- The default value for the SupportsCatalogs connection property has been changed to `true`. Now, for catalog calls, such as getTables and getColumns, the driver returns the database as catalog by default. See SupportsCatalogs on page 130 for details.
- Java SE 7 has reached the end of its product life cycle and will no longer receive generally available security updates. As a result, the driver will no longer support JVMs that are version Java SE 7 or earlier. Support for distributed versions of Java SE 7 and earlier will also end, including IBM SDK (Java Edition).

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\postgresql.jar
     ```

   - **UNIX/LINUX Example**

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

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

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

     User ID/password authentication

     ```
     jdbc:datadirect:postgresql://server1:5432;
     User=test;Password=secret;
     ```

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

     Note: See User ID and password authentication on page 51 for details.

     Kerberos authentication

     ```
     jdbc:datadirect:postgresql://server1:5432;
     DatabaseName=postgresqlDB;AuthenticationMethod=kerberos;
     ServicePrincipalName=postgres/myserver.example.com@EXAMPLE.COM;
     ```

     Note: See Kerberos authentication on page 52 for details.

   - **Data sources**: The driver also supports connecting using JDBC data sources. A JDBC data source is a Java object, specifically a DataSource object, that defines connection information required for a JDBC driver to connect to the database. See Connecting using data sources for more information.

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

4. Set the values for any optional properties that you want to configure. For additional information on optional features and functionality, see the following resources:
   - Connection URL examples provides connection string examples that can be used to configure common functionality and features. You can modify and combine these examples to create a string that best suites your environment.
   - Connection property descriptions provides a complete list of supported properties by functionality.
   - Performance considerations describes connection properties that affect performance, along with recommended settings.
5. Connect to your service and begin accessing data with your applications, BI tools, database tools, and more. To help you get started, the following resources guide you through accessing data with some common tools:
   - Tableau: Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data.
   - DbVisualizer: DB Visualizer is a database tool that allows you to connect and execute SQL statements against your data.
   - DataDirect Test: DataDirect Test allows you to test connect, execute SQL statements, and practice using the JDBC API right out of the box.

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.postgresql.PostgreSQLDriver

DataSource class:

com.ddtek.jdbcx.postgresql.PostgreSQLDataSource

Connection URL examples

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

Note:

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

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

- User ID/Password authentication
- Microsoft Entra ID user ID/password authentication
- Microsoft Entra ID service principal authentication
- Kerberos authentication
- Proxy server authentication
- AWS IAM authentication

User ID/password authentication

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

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:postgresql://server1:5432;User=test;Password=secret;");
```

For more information on these properties and values, see User ID and password authentication on page 51.

Microsoft Entra ID user ID/password authentication

This string includes the properties used to connect with the Microsoft Entra ID (Azure Active Directory) user ID and password authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:postgresql://myserver.postgres.database.azure.com:5432;
AuthenticationMethod=EntraIDPassword;User=test@mydomain.com;
Password=secret;AzureTenantID=xyz012");
```

For more information on these properties and values, see Microsoft Entra ID authentication on page 56.

Microsoft Entra ID service principal authentication

This string includes the properties used to connect with the Microsoft Entra ID (Azure Active Directory) service principal authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:postgresql://myserver.postgres.database.azure.com:5432;
AuthenticationMethod=EntraIDServicePrincipal;User=1234abcd-1234-abcd-1234-abcd1234abcd;
Password=ABcdEFg/hiJkLmNOPqR01stUvWxyzYx2wvUTsrQpO;AzureTenantID=xyz012");
```

For more information on these properties and values, see Microsoft Entra ID authentication on page 56.

Kerberos authentication

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

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:postgresql://server1:5432;DatabaseName=postgresqlDB;AuthenticationMethod=kerberos;
ServicePrincipalName=postgres/myserver.example.com@EXAMPLE.COM;");
```

For more information on these properties and values, see Kerberos authentication on page 52.

Proxy server authentication

This string includes the properties used to connect with proxy server authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:postgresql://server1:5432;ProxyHost=pserver;ProxyPassword=secret;
ProxyPort=808;ProxyUser=PUser;User=jsmith@example.com;Password=secret;");
```

For more information on these properties and values, see Connection property descriptions on page 77.

AWS IAM authentication

This string includes the properties used to connect with AWS IAM authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:postgresql://server1:5432;AuthenticationMethod=AWSIAM;
  AccessKey=ABCDEFGHIJKL1EXAMPLE;Region=us-east-2;
  SecretKey=aBcdeFGhiJKLM/N1OPQRS/tUvWxyzAEXAMPLEKEY");
```

For more information on these properties and values, see AWS IAM authentication on page 57.

See also

Authentication on page 51

Data types

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

Table 1: PostgreSQL Data Types

PostgreSQL Data Type	JDBC Data Type
Bigint	BIGINT
Bigserial	BIGINT
Bit	BIT or BINARY[1]
Bit varying	VARBINARY
Boolean	BOOLEAN
Box	VARCHAR

[1] Bit maps to -7 (BIT) when the length for the bit is 1. If the length is greater than 1, the driver maps the column to BINARY.

PostgreSQL Data Type	JDBC Data Type
Bytea	LONGVARBINARY
Character	CHAR
Character varying	VARCHAR or LONGVARCHAR[2]
Citext[3, 4]	LONGVARCHAR
CIDR	VARCHAR
Circle	VARCHAR
Date	DATE
DATERANGE[5]	VARCHAR
Double precision	DOUBLE
INET	VARCHAR
INT4RANGE[5]	VARCHAR
INT8RANGE[5]	VARCHAR
Interval	VARCHAR
Integer	INTEGER
Json[6]	VARCHAR
Jsonb[4]	VARCHAR
Line[7]	VARCHAR
LSEG	VARCHAR
MACADDR	VARCHAR
Money	VARCHAR
Numeric	NUMERIC
NUMRANGE[5]	VARCHAR

[2] You may determine how these columns are described by setting the VarcharClobThreshold on page 135 connection property.
[3] The Citext data type behaves the same as the Text data type, except that it is case-insensitive. The select operations performed on Citext columns return case-insensitive results.
[4] Supported for PostgreSQL versions 9.4 and higher.
[5] Supported for PostgreSQL versions 9.2 and higher.
[6] Supported for PostgreSQL versions 9.2 and higher.
[7] Supported for PostgreSQL versions 7.1 and higher.

PostgreSQL Data Type	JDBC Data Type
Path	VARCHAR
Point	VARCHAR
Polygon	VARCHAR
Real	REAL
Serial	INTEGER
Smallint	SMALLINT
Text	LONGVARCHAR
Time	TIMESTAMP
Time with time zone	TIMESTAMP
Timestamp	TIMESTAMP
Timestamp with time zone	TIMESTAMP
TSRANGE[5]	VARCHAR
TSTZRANGE[5]	VARCHAR
TXID_SNAPSHOT[8]	VARCHAR
UUID[9]	VARCHAR
XML[9]	SQLXML

getTypeInfo()

The following table provides getTypeInfo() results for PostgreSQL databases supported by the driver.

[8] Supported for PostgreSQL versions 8.3 and higher.
[9] Supported for PostgreSQL versions 8.3 and higher.

Table 2: getTypeInfo() for PostgreSQL

TYPE_NAME = bigint AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = Bigint MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = bigserial AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = Bigserial MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = bit [10] AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -2 (BINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Bit MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 83886080 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

[10] Bit maps to -7 (BIT) when the length for the bit is 1. If the length is greater than 1, the driver maps the column to BINARY.

TYPE_NAME = bit varying AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *max length* DATA_TYPE = -3 (VARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Bit varying MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 83886080 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = boolean AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 16 (BOOLEAN) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' 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 UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = box AUTO_INCREMENT = false CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Box MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 128 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true

TYPE_NAME = bytea AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = -4 (LONGVARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Bytea MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = character AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *length* DATA_TYPE = 1 (CHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Character MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10485760 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = character varying [11] AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *max length* DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Character varying MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10485760 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

[11] Columns of this type will be described as VARCHAR when precision is 4000 or less. If precision is greater than 4000, columns will be described as LONGVARCHAR.

TYPE_NAME = CIDR AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = INET MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 128 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = circle AUTO_INCREMENT = false CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Circle MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 96 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = citext[12, 13] AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *length* DATA_TYPE = -1 (LONGVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Citext MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1073741823 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

[12] The Citext data type behaves the same as the Text data type, except that it is case-insensitive. The select operations performed on Citext columns return case-insensitive results.

[13] Supported for PostgreSQL versions 9.4 and higher.

TYPE_NAME = date AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 91 (DATE) FIXED_PREC_SCALE = false LITERAL_PREFIX = {d' 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 UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = daterange AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = DATERANGE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 25 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = double precision 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 precision MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 2 PRECISION = 53 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = INET AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = INET MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 128 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = INT4RANGE AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = INT4RANGE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = INT8RANGE AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = INT8RANGE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 56 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true

TYPE_NAME = integer AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = Integer MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = interval AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Interval MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 64 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = json[14] AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = json MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

[14] Supported for PostgreSQL versions 9.2 and higher.

TYPE_NAME = jsonb[13] 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 = jsonb MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = line AUTO_INCREMENT = false CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Line MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10485760 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = LSEG AUTO_INCREMENT = false CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = LSEG MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 128 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true

TYPE_NAME = MACADDR AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = MACADDR MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 17 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = money[15] AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = true LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Money MAXIMUM_SCALE = 2	MINIMUM_SCALE = 2 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 27 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = numeric AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *precision, scale* DATA_TYPE = 2 (NUMERIC) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = Numeric MAXIMUM_SCALE = 0	MINIMUM_SCALE = 999 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1000 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

[15] For PostgreSQL version 8.3 and higher, the precision is 27. For PostgreSQL versions before 8.3, the precision is 15.

TYPE_NAME = NUMRANGE AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = NUMRANGE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 1350 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = path AUTO_INCREMENT = false CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Path MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10485760 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = point AUTO_INCREMENT = false CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Point MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 64 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true

TYPE_NAME = polygon AUTO_INCREMENT = false CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Polygon MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10485760 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = real AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = *precision* DATA_TYPE = 7 (REAL) 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 = 2 PRECISION = 24 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = serial AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = Serial MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = smallint AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 5 (SMALLINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = Smallint MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 5 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = text AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *length* DATA_TYPE = -1 (LONGVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = Text MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1073741823 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = time AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *fractional_seconds_precision* DATA_TYPE = 93 (TIME) FIXED_PREC_SCALE = false LITERAL_PREFIX = {t' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = Time MAXIMUM_SCALE = 6	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 15 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = time with time zone AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *fractional_seconds_precision* DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = {t' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = Time with time zone MAXIMUM_SCALE = 6	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 22 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = timestamp AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *fractional_seconds_precision* DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = {ts' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = Timestamp MAXIMUM_SCALE = 6	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 26 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = timestamp with time zone AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = *fractional_seconds_precision* DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = {ts' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = Timestamp with time zone MAXIMUM_SCALE = 6	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 33 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = TSRANGE AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = TSRANGE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 57 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = TSTZRANGE AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = TSTZRANGE MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 65 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = true
TYPE_NAME = TXID_SNAPSHOT AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = TXID_SNAPSHOT MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10485760 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = UUID[16] AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = uuid MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 36 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = XML [16] AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 2009 (SQLXML) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = XML MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10485760 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

SQL escape sequences

The driver supports the following SQL escape sequences.

- Date, Time, and Timestamp Escape Sequences
- Scalar Functions
- Outer Join Escape Sequences
- LIKE Escape Character Sequence for Wildcards

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

[16] Supported for PostgreSQL versions 8.3 and higher.

Supported scalar functions

You can use scalar functions in SQL statements with the following syntax:

`{fn scalar-function}`

where:

scalar-function

is a scalar function supported by the driver, as listed in the following table.

Example:

```
SELECT id, name FROM emp WHERE name LIKE {fn UCASE('Smith')}
```

Table 3: Supported Scalar Functions

String Functions	Numeric Functions	Timedate Functions	System Functions
ASCII	ABS	CURDATE	USERNAME
BIT_LENGTH	ACOS	CURRENT_DATE	DBNAME
CHAR	ASIN	CURRENT_TIME	IFNULL
CHAR_LENGTH	ATAN	CURRENT_TIMESTAMP	
CHARACTER_LENGTH	ATAN2	CURTIME	
CONCAT	CEILING	EXTRACT	
LCASE	COS	NOW	
LENGTH	COT		
LEFT [17]	DEGREES		
LOCATE	EXP		
LTRIM	FLOOR		
OCTET_LENGTH	LOG		
POSITION	LOG10		
REPEAT	MOD		
REPLACE	PI		
RIGHT	POWER		
RTRIM	RADIANS		
SUBSTRING	RAND		
UCASE	ROUND		
	SIGN		
	SIN		
	SQRT		

[17] Supported for PostgreSQL 9.1 and higher

String Functions	Numeric Functions	Timedate Functions	System Functions
	TAN TRUNCATE		

DataDirect tools

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

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

Troubleshooting

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

Additional information

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

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

Contacting Technical Support

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

https://www.progress.com/support

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

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

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

2

Tutorials

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

For details, see the following topics:

- Tableau
- DbVisualizer

Tableau

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

To use the driver to access data with Tableau:

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

   ```
   postgresql.jar
   ```

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

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

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

3. Open Tableau. From the **Connect** menu, select **Other Databases (JDBC)**.
4. In the **Other Databases (JDBC)** dialog, provide values for the following fields; then, click **Sign In**.
   - **URL**: Copy and paste your connection URL into this field. The following examples show how to connect using either user ID/password or Kerberos authentication.

     User ID/password authentication

     ```
     jdbc:datadirect:postgresql://server1:5432;
     ```

     Note: See User ID and password authentication on page 51 for details.

     Kerberos authentication

     ```
     jdbc:datadirect:postgresql://server1:5432;
     DatabaseName=postgresqlDB;AuthenticationMethod=kerberos;
     ServicePrincipalName=postgres/myserver.example.com@EXAMPLE.COM;
     ```

     Note: See Kerberos authentication on page 52 for details.

   - **Dialect**: Select **SQL92** (the default) from the drop-down box.
   - **Username**: If required by the authentication method being used, enter the user name. Alternatively, this value can be specified with the `User` property in the connection string.
   - **Password**: If required by the authentication method being used, enter the password. Alternatively, this value can be specified with the `Password` property in the connection string.
5. The **Data Source** window appears. In the **Schema** field, select the schema for the service you want to use.
6. In the **Table** field, the tables stored in the selected schema are now exposed and available for selection.

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

DbVisualizer

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

Adding a driver

To add a driver with DbVisualizer:

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

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

   Windows

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

   Linux

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

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

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

     ```
     PostgreSQL
     ```

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

     ```
     jdbc:datadirect:postgresql:
     ```

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

     ```
     com.ddtek.jdbcx.postgresql.PostgreSQLDataSource
     ```

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 either user ID/password or Kerberos authentication.

     User ID/password authentication

     ```
     jdbc:datadirect:postgresql://server1:5432;
     User=test;Password=secret;
     ```

     Note: See User ID and password authentication on page 51 for details.

Kerberos authentication

```
jdbc:datadirect:postgresql://server1:5432;
DatabaseName=postgresqlDB;AuthenticationMethod=kerberos;
ServicePrincipalName=postgres/myserver.example.com@EXAMPLE.COM;
```

Note: See Kerberos authentication on page 52 for details.

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 BLOGS table:

   ```
   SELECT * BLOGS
   ```

You have successfully accessed your data with DbVisualizer.

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
- Connecting using the JDBC Driver Manager
- Connecting using data sources
- Authentication
- Data encryption
- Using client information
- Bulk load
- Performance considerations

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/postgresql.jar`

Windows Example

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

UNIX Example

```
CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/postgresql.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 example shows, this method specifies a string containing a connection URL.

User ID/password authentication

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:postgresql://server1:5432;
    User=test;Password=secret;");
```

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

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 user ID/password authentication.

Connection URL Syntax

The connection URL takes the following form:

```
jdbc:datadirect:postgresql://servername:port;
User=userID;Password=password;[property=value[;...]];
```

where:

servername

is the IP address or name of the server to which you are connecting.

port

is the number of the TCP/IP port.

userID

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

password

specifies a password that is used to connect to your PostgreSQL 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:postgresql://server1:5432;
    User=test;Password=secret;");
```

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

See also

Connection property descriptions on page 77
Connection URL examples on page 14
User ID and password authentication on page 51

Testing the connection

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

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

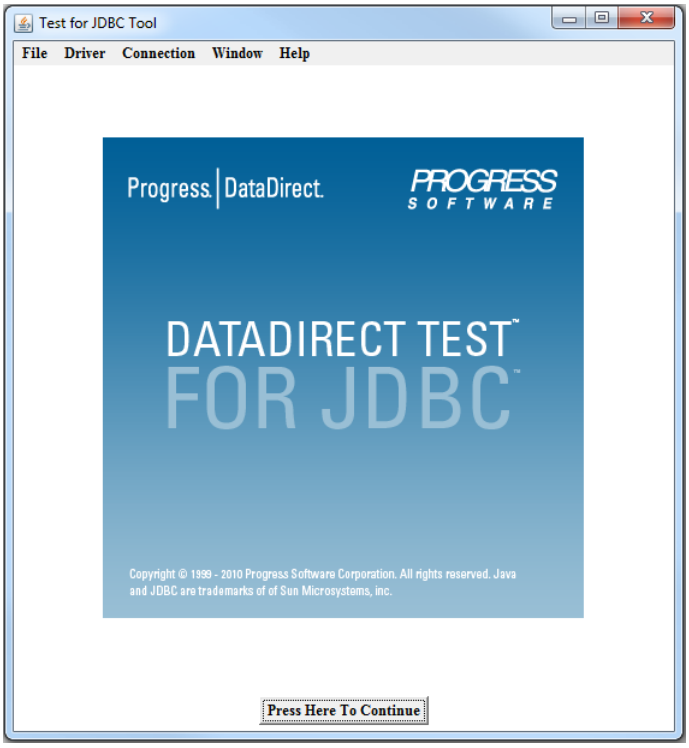
1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

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

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

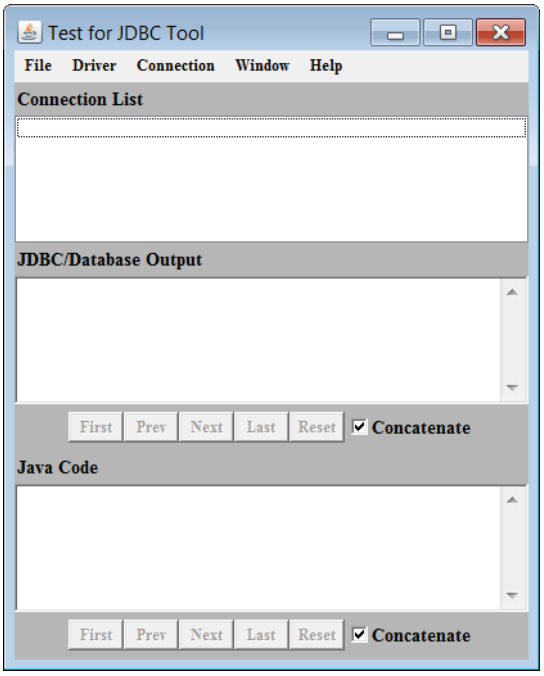
- `testforjdbc.sh` (on UNIX and Linux systems)

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



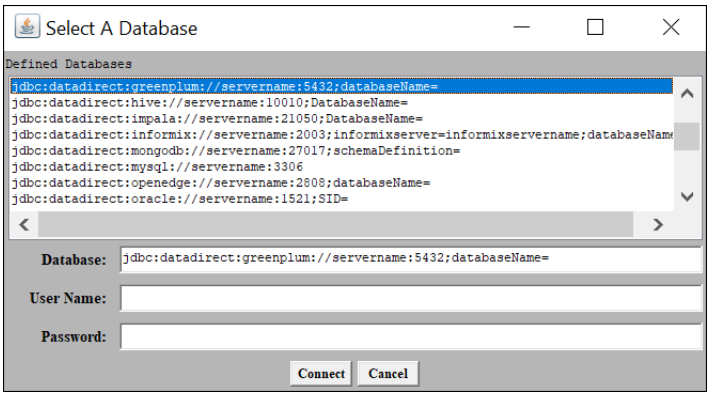
3. Click **Press Here to Continue**.

   The main dialog appears:



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

   The **Select A Database** dialog appears:



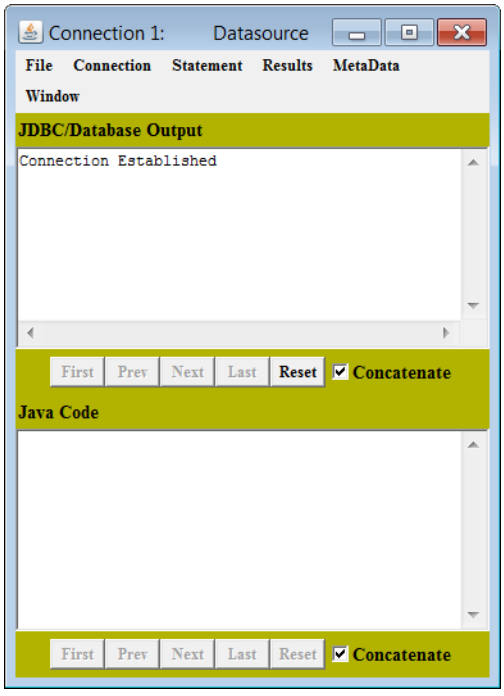
5. Select the appropriate database template from the **Defined Databases** field.
6. In the **Database** field, specify all required connection properties.

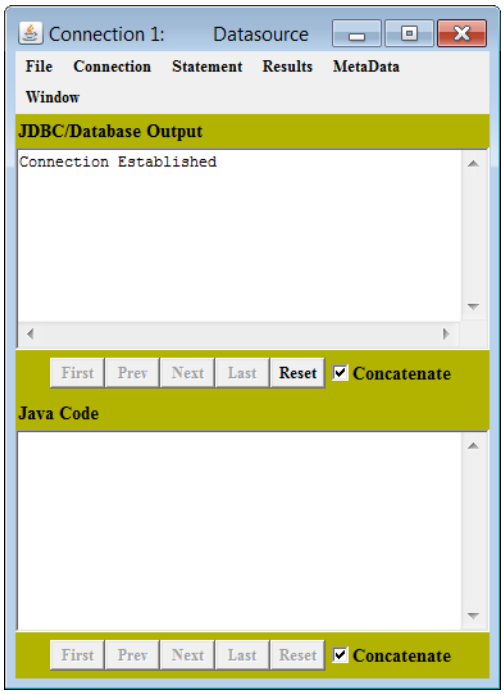
   For example:

   ```
   jdbc:datadirect:postgresql://server1:5432;
   ```

7. Click **Connect**.

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





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

Connecting using data sources

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

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

How data sources are implemented

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

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

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

See also

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.postgresql.PostgreSQLDataSource;
```

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

Note:

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

```
PostgreSQLDataSource mds = new PostgreSQLDataSource();
mds.setDescription("My PostgreSQL Data Source");
mds.setServerName("server1");
mds.setPortNumber(5432);
```

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 instance of PostgreSQLDataSource)
{
PostgreSQLDataSource jmds = (PostgreSQLDataSource) ds;
```

```
System.out.println("description=" + jmds.getDescription());
System.out.println("serverName=" + jmds.getServerName());
System.out.println("portNumber=" + jmds.getPortNumber());
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.

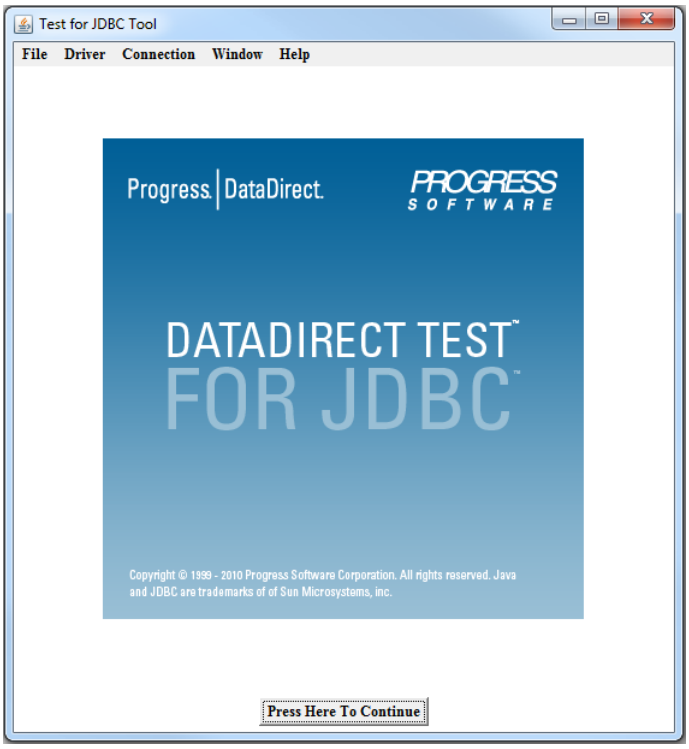
Take the following steps to establish a connection.

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.

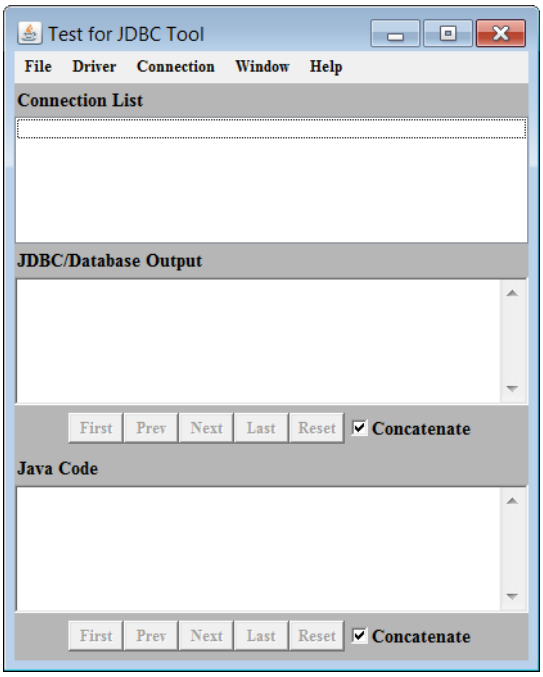
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

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



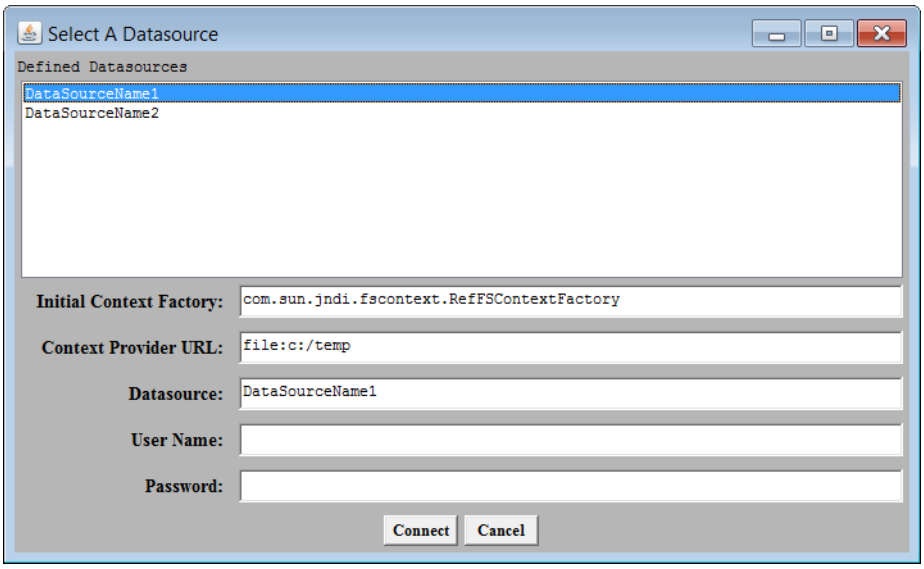
3. Click **Press Here to Continue**.

   The main dialog appears:



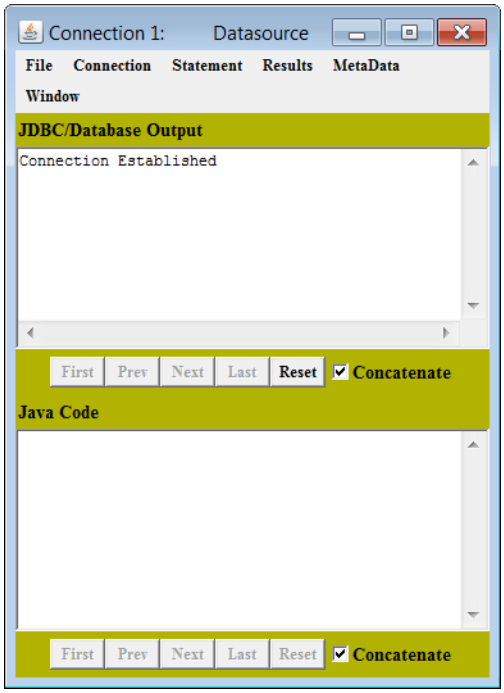
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   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:

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

  This method requires knowledge of how to configure your Kerberos environment and supports Windows Active Directory Kerberos only.
- *Microsoft Entra ID authentication* (formerly Azure Active Directory Authentication) authenticates using Microsoft Entra ID authentication when establishing a connection to Azure. The driver supports the following methods of Entra ID authentication:
  - User and password authentication
  - Service principal authentication
- *AWS IAM Authentication* authenticates using AWS (Amazon Web Services) IAM (Identity and Access Management) credentials.

By default, the driver is configured to use user ID/password authentication (`AuthenticationMethod=userIdPassword`).

See also

AuthenticationMethod on page 90

User ID and password authentication

The driver supports user ID/password authentication. It authenticates the user to the database using a username and password.

The driver supports the MD5, SCRAM-SHA-256, and SCRAM-SHA-256-PLUS password hashing mechanisms for user ID/password authentication. During connection, the driver detects and uses the most secure method supported by the server.

Take the following steps to configure user ID/Password authentication.

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

The following example shows the connection information required to establish a connection using user ID/password authentication.

```
Connection conn = DriverManager.getConnection
          ("jdbc:datadirect:postgresql://server1:5432;
            User=test;Password=secret);
```

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

See also

AuthenticationMethod on page 90
User on page 134
Password on page 117

Kerberos authentication

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

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

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

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

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

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

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

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

   Option 2. Set up a default configuration. Modify the Java security properties file to indicate the URL of the login configuration file with the `login.config.url.`*n* property where *n* is an integer connoting separate, consecutive login configuration files. When more than one login configuration file is specified, then the files are read and concatenated into a single configuration.

a) Open the Java security properties file. The security properties file is the `java.security` file in the `/jre/lib/security` directory of your Java installation.

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

c) Below the `# Default login configuration file` line, add the URL of the login configuration file as the value for a `login.config.url.`*n* property. For example:

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

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

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

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

Oracle JVM

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

IBM JVM

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

3. Set the driver's AuthenticationMethod connection property to `kerberos`. (See "AuthenticationMethod" for details.)
4. Set the ServicePrincipalName connection property to the service principal name registered with the KDC.

   The ServicePrincipalName takes the following form.

   Service_Name/Fully_Qualified_Domain_Name@REALM`.COM`

   The value of the ServicePrincipalName property can include the Kerberos realm name, but it is optional. If you do not specify the realm name, the default realm is used.

   See "ServicePrincipalName" for details on the composition of the service principal name.

5. Optionally, set the LoginConfigName connection property if the name of the JAAS login configuration file entry is different from the driver default `JDBC_DRIVER_01`. (See "The JAAS login configuration file" and "LoginConfigName" for details.)

   `JDBC_DRIVER_01` is the default entry name for the JAAS login configuration file (`JDBCDriverLogin.conf`) installed with the driver. When configuring your Kerberos environment, your network or system administrator may have used a different entry name. Check with your administrator to verify the correct entry name.

See also

Kerberos authentication requirements on page 54

The JAAS login configuration file on page 54
AuthenticationMethod on page 90
ServicePrincipalName on page 127
LoginConfigName on page 111

Kerberos authentication requirements

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

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

Table 4: Kerberos configuration requirements

Component	Requirements
Database server	No restrictions
Kerberos server	The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos key distribution center (KDC). Network authentication must be provided by one of the following methods. • Windows Active Directory on any of the supported operating systems • MIT Kerberos 1.5 or higher
Client	Java SE 8 or higher must be installed.

See also

Kerberos authentication on page 52

The JAAS login configuration file

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

Setting up a default configuration

To set up a default configuration, you must modify the Java security properties file to indicate the URL of the login configuration file with the `login.config.url.`*n* property where *n* is an integer connoting separate, consecutive login configuration files. When more than one login configuration file is specified, then the files are read and concatenated into a single configuration. The following steps summarize how to modify the security properties file.

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

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

JAAS login configuration file entry for the driver

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

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

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

where:

entry_name

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

login_module

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

flag_value

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

module_options

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

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

Oracle JVM

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

IBM JVM

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

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

See also

Kerberos authentication on page 52
LoginConfigName on page 111

Microsoft Entra ID authentication

The driver supports Microsoft Entra ID (Entra ID) authentication (formerly known as Azure Active Directory authentication). Entra ID authentication is an alternate authentication type that allows administrators to centrally manage user permissions to Azure SQL Database data stores. The driver supports the following methods of Entra ID authentication:

- User and password authentication: The driver retrieves an access token by authenticating with the Entra ID user and password.
- Service principal authentication: The driver retrieves an access token by authenticating with the client ID and client secret of the service principal.

Note: When using Entra ID authentication, the driver requires root CA certificates to establish an SSL connection to a database. The driver determines the location of the truststore containing the required certificates by using the default JRE `cacerts` file unless a different file has been specified by the `javax.net.ssl.trustStore` Java system property. The truststore location cannot be specified using the driver's Truststore property.

User and password authentication

To use user and password authentication with Entra ID:

- Set the AuthenticationMethod property to specify a value of `EntraIDPassword`.
- Set the User property to specify your Entra ID username using the `userid@domain.com` format.
- Set the Password property to specify your Entra ID password.
- Set the AzureTenantID property to specify the tenant associated with the PostgreSQL server.
- Specify values for minimum required properties for establishing a connection.
  - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name for your Azure server. For example, `myserver.postgres.database.azure.com`.
  - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.

For example, the following is a connection string with only the required options for making a connection using Entra ID authentication.

Note: If the HostNameInCertificate is not specified, the driver automatically uses the value of the ServerName from the URL as the value for validating the certificate.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:postgresql://myserver.postgres.database.azure.com:5432;
AuthenticationMethod=EntraIDPassword;User=test@mydomain.com;
Password=secret;AzureTenantID=xyz012");
```

Service principal authentication

To use service principal authentication with Entra ID:

Note: The user must configure the API permission `Application.Read.All` for the application associated with the database. Refer to Permission for details.

- Set the AuthenticationMethod property to specify a value of `EntraIDServicePrincipal`.
- Set the User property to specify the client ID of the service principal.
- Set the Password property to specify the client secret of the service principal.
- Set the AzureTenantID property to specify the Azure tenant ID associated with your PostgreSQL server.
- Specify values for minimum required properties for establishing a connection.
  - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name for your Entra ID server. For example, `myserver.postgres.database.azure.com`.
  - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.

Note: If the HostNameInCertificate is not specified, the driver automatically uses the value of the ServerName from the URL as the value for validating the certificate.

For example, the following is a connection string with only the required options for making a connection using Entra ID authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:postgresql://myserver.postgres.database.azure.com:5432;
AuthenticationMethod=EntraIDServicePrincipal;User=1234abcd-1234-abcd-1234-abcd1234abcd;
Password=ABcdEFg/hiJkLmNOPqR01stUvWxyzYx2wvUTsrQpO;AzureTenantID=xyz012");
```

See also

Connection property descriptions on page 77

AWS IAM authentication

The AWS Identity and Access Management (IAM) authentication method can be used to securely connect to the PostgreSQL database deployed on AWS. Instead of username and password credentials, users authenticate using their AWS Access Key and Secret Key, enhancing security and simplifying credential management.

Prerequisites

- Enable IAM Authentication on the RDS instance via the AWS Management Console or CLI.
- Create an IAM user and attach an appropriate IAM policy that grants access to the RDS instance.
- Create the database user that IAM will map to, ensuring it matches the IAM user name.
- Grant the necessary privileges to the database user that corresponds to the IAM role.
- Verify that the instance's parameter group has IAM database authentication enabled.
- Configure IAM policies and roles to authorize the identity that will connect to the RDS instance.

Note: When using the `AWSIAM` authentication method, the `encryptionMethod` is automatically set to `'SSL'`, regardless of the value provided. This is because the driver requires an SSL connection to authenticate with AWS IAM.

To configure the driver to use AWS IAM authentication:

- Specify values for minimum required properties for establishing a connection.
  - Set the ServerName property to specify either the IP address in IPv4 or IPv6 format, or the server name for your server.
  - Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.
- Set the AuthenticationMethod property to `AWSIAM`.
- Set the AccessKey property to specify your access key ID for your IAM user or AWS account root user.
- Set the Region property to specify name of the region that hosts your AWS server .For example, `us-east-1` or `us-east-2`.

  For a list of regions, refer to the AWS documentation.
- Set the SecretKey property to specify your secret access key for an IAM user or AWS account root user.
- Set the TrustStore property to specify the directory of the truststore file to be used when SSL is enabled.
- Set the TrustStorePassword property to specify the password that is used to access the truststore file.
- Optionally, specify values for any additional properties you want to configure.

The following example shows the connection information required to establish a connection with AWS IAM authentication enabled.

For a connection URL:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:postgresql://server1:5432;AuthenticationMethod=AWSIAM;
  AccessKey=ABCDEFGHIJKL1EXAMPLE;Region=us-east-2;
  SecretKey=aBcdeFGhiJKLM/N1OPQRS/tUvWxyzAEXAMPLEKEY");
```

See also

Connection property descriptions on page 77

Data encryption

If your database connection is not configured to use data encryption, data is sent across the network in a format that is designed for fast transmission and can be decoded by interceptors, given some time and effort. For example, text data is often sent across the wire as clear text. Because this format does not provide complete protection from interceptors, you may want to use data encryption to provide a more secure transmission of data. For example, you may want to use data encryption in the following scenarios:

- You have offices that share confidential information over an intranet.
- You send sensitive data, such as credit card numbers, over a database connection.
- You need to comply with government or industry privacy and security requirements.

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

The driver supports the Secure Sockets Layer (SSL) encryption. SSL is an industry-standard protocol for sending encrypted data over database connections. SSL secures the integrity of your data by encrypting information and providing client/server authentication.

TLS/SSL encryption

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

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

The version of TLS/SSL that is used and which TLS/SSL cryptographic algorithm is used depends on which JVM you are using. Refer to your JVM documentation for more information about its TLS/SSL support.

See also

Configuring TLS/SSL encryption on page 61

TLS/SSL server authentication

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

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

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

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

and

```
java -Djavax.net.ssl.trustStorePassword=MyTruststorePassword
```

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

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

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

  and

  ```
  TrustStorePassword=MyTruststorePassword
  ```

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

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

TLS/SSL client authentication

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

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

The drivers can use the following types of keystores:

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

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

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

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

  and

  ```
  java -Djavax.net.ssl.keyStorePassword=MyKeystorePassword
  ```

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

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

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

  `KeyStore=C:\Certficates\MyKeyStore`

  and

  `KeyStorePassword=MyKeystorePassword`

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

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

Configuring TLS/SSL encryption

The following steps outline how to configure TLS/SSL encryption.

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

To configure SSL encryption:

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

---

1. Choose the type of encryption for your application.

   If you want the driver to encrypt all data, set the EncryptionMethod property to one of the following:

   - SSL: Data is encrypted using TLS/SSL. If the database server does not support TLS/SSL, the connection fails and the driver throws an exception.
   - requestSSL: Data is encrypted using TLS/SSL. If the database server does not support TLS/SSL, the driver establishes an unencrypted connection.

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

a) Specify the location and password of the keystore file. Either set the KeyStore and KeyStorePassword properties or their corresponding Java system properties (javax.net.ssl.keyStore and javax.net.ssl.keyStorePassword, respectively).

b) If any key entry in the keystore file is password-protected, set the KeyPassword property to the key password.

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.

Using client information

Many databases allow applications to store client information associated with a connection, which can be useful for database administration and monitoring purposes. The driver allows applications to store and return the following types of client information.

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

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

Bulk load

As PostgreSQL does not have native bulk load support, the PostgreSQL driver emulates bulk load using the standard batch mechanism.

The BulkLoadBatchSize connection property affects how bulk load works with the PostgreSQL driver. It suggests the number of rows to be loaded to the database when bulk loading data. The default is `1000`.

See also

BulkLoadBatchSize on page 93

Performance considerations

You can optimize application performance by adopting guidelines described in this section.

BatchMechanism: Setting BatchMechanism to `copy` leverages the PostgreSQL `COPY` command for substantial performance gains. When `BatchMechanism=copy`, the driver creates an in-memory representation of a CSV file based on all the rows contained in a parameter array, and the PostgreSQL `COPY` command is executed to insert the rows from the CSV file into the target table.

BulkLoadBatchSize: The BulkLoadBatchSize property is used to specify the number of rows the driver loads at a time when bulk loading data. Performance can be improved by increasing the number of rows because fewer network round trips are required. For example, if BulkLoadBatchSize is set to 10,000 rows and you are inserting 100,000 rows, the driver executes 10 batch inserts that require separate round trips to complete the bulk operation. Be aware that increasing the number of rows that are loaded also causes the driver to consume more memory on the client.

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

InsensitiveResultSetBufferSize: To improve performance, result set data can be cached instead of written to disk. If the size of the result set data is greater than the size allocated for the cache, the driver writes the result set to disk. The maximum cache size setting is 2 GB.

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

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

ResultSetMetaDataOptions: The driver’s performance may be adversely affected if you set this option to `1`. If set to `1` and the ResultSetMetaData.getTableName method is called, the driver performs emulations which take additional processing.

VarcharClobThreshold: There are performance penalties when enabling CLOB functionality. To provide the benefits associated with Clobs, data must be cached. Because data is cached, your application will incur a performance penalty, particularly if data is read once sequentially. This performance penalty can be severe if the size of the long data is larger than available memory. If you want to avoid the performance penalties associated with CLOB functionality, you should set this value at a value greater than the maximum *Character varying* column width your application handles.

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:

- Statement pooling
- Connection pooling
- Failover
- Refcursors
- Returning and inserting/updating XML data
- Isolation levels
- Using scrollable cursors
- JTA support
- Batch inserts
- Large object (LOB) support
- Using COPY command
- Parameter metadata support
- ResultSet metadata support
- Rowset support
- Auto-generated keys support

Statement pooling

Most applications have a certain set of SQL statements that are executed multiple times and a few SQL statements that are executed only once or twice during the life of the application. Similar to connection pooling, *statement pooling* provides performance gains for applications that execute the same SQL statements multiple times over the life of the application.

A *statement pool* is a group of prepared statements that can be reused by an application. If you have an application that repeatedly executes the exact same SQL statements, statement pooling can improve performance because the database server does not have to repeatedly parse and create cursors for the same statement. In addition, the associated network round trips to the database server are avoided.

The drivers have an internal prepared statement pooling mechanism, which allows you to realize the performance benefits of statement pooling when you are not running from within an application server or another application that provides its own statement pooling. You can enable this driver-based internal statement pooling with the MaxPooledStatements connection property.

The DataDirect for JDBC drivers also support the DataDirect Statement Pool Monitor. You can use the Statement Pool Monitor to load statements into and remove statements from the statement pool as well as generate information to help you troubleshoot statement pooling performance. The Statement Pool Monitor is an integrated component of the driver, and you can manage statement pooling directly with DataDirect-specific methods. In addition, the Statement Pool Monitor can be enabled as a Java Management Extensions (JMX) MBean. When enabled as a JMX MBean, the Statement Pool Monitor can be used to manage statement pooling with standard JMX API calls, and it can easily be used by JMX-compliant tools, such as JConsole. To enable the Statement Pool Monitor as a JMX MBean, you must register the Statement Pool Monitor MBean with the RegisterStatementPoolMonitorMBean connection property.

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

See also

MaxPooledStatements on page 115
RegisterStatementPoolMonitorMBean on page 124

Connection pooling

Progess DataDirect for JDBC drivers support connection pooling using the DataDirect Connection Pool Manager. Typically, creating a connection is the most performance-expensive operations that an application performs. Connection pooling allows you to reuse connections rather than create a new one every time a driver needs to connect to the database. Further, connection pooling manages connection sharing across multiple user requests to maintain performance and reduce the number of new connections to be created.

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

Failover

The driver provides connection failover support to ensure continuous, uninterrupted access to data.

Note:

For implementation and configuration details, see Configuring failover on page 67.

Note: For more information on connection failover and different levels of failover protection provided by the driver, refer to "Failover" in the *Progress DataDirect for JDBC Drivers Reference*.

Configuring failover

The driver allows you to specify a list of alternate database servers that are tried at connection time if the primary server is not accepting connections. Connection attempts continue until a connection is successfully established or until all the database servers in the list have been tried the specified number of times.

Specifying primary and alternate servers

Connection information for primary and alternate servers can be specified using either a connection URL through the JDBC Driver Manager or a JDBC data source. For example, the following connection URL for the PostgreSQL driver specifies connection information for the primary and alternate servers using a connection URL:

```
jdbc:datadirect:postgresql://server1:5432;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:5432;DatabaseName=TEST2,
server3:5432;DatabaseName=TEST3)
```

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

In this example:

```
...server1:5432;DatabaseName=TEST...
```

is the part of the connection URL that specifies connection information for the primary server. Alternate servers are specified using the AlternateServers property. For example:

```
...;AlternateServers=(server2:5432;DatabaseName=TEST2,server3:5432;
DatabaseName=TEST3)
```

Similarly, the same connection information for the primary and alternate servers specified using a JDBC data source would look like this:

```
PostgreSQLDataSource mds = new PostgreSQLDataSource();
mds.setDescription("My PostgreSQLDataSource");
mds.setServerName("server1");
mds.setPortNumber(5432);
mds.setDatabaseName("TEST");
mds.setUser("test");
mds.setPassword("secret");
mds.setAlternateServers("server2:5432;DatabaseName=TEST2,server3:5432;
   DatabaseName=TEST3")
```

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 this example, connection information for the primary server is specified using the ServerName, PortNumber, and DatabaseName properties. Connection information for the alternate servers is specified using the AlternateServers property.

The value of the AlternateServers property is a string that has the format:

```
(servername1[:port1][;property=value[;...]][,servername2[:port2]
[;property=value[;...]]]...)
```

where:

servername1 is the IP address or server name of the first alternate database server, *servername2* is the IP address or server name of the second alternate database server, and so on. The IP address or server name is required for each alternate server entry.

port1 is the port number on which the first alternate database server is listening, *port2* is the port number on which the second alternate database server is listening, and so on. Port numbers are optional for each alternate server entry. If unspecified, the port number specified for the primary server is used. If a port number is unspecified for the primary server, a default port number of 5432 is used.

property=value is the DatabaseName connection property. This property is optional for each alternate server entry. For example:

```
jdbc:datadirect:postgresql://server1:5432;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:5432;DatabaseName=TEST2,
server3:5432;DatabaseName=TEST3)
```

If you do not specify the DatabaseName connection property in an alternate server entry, the connection to that alternate server uses the property specified in the URL for the primary server. For example, if you specify DatabaseName=TEST for the primary server, but do not specify a database name in the alternate server entry as shown in the following URL, the driver tries to connect to the TEST database on the alternate server.

```
jdbc:datadirect:postgresql://server1:5432;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:5432,server3:5432)
```

Specifying connection retry

Connection retry allows the PostgreSQL driver to retry connections to the primary database server, and if specified, alternate servers until a successful connection is established. You use the ConnectionRetryCount and ConnectionRetryDelay properties to enable and control how connection retry works. For example:

```
jdbc:datadirect:postgresql://server1:5432;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:5432;DatabaseName=TEST2,
server3:5432;DatabaseName=TEST3);ConnectionRetryCount=2;
ConnectionRetryDelay=5
```

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

In this example, if a successful connection is not established on the PostgreSQL driver's first pass through the list of database servers (primary and alternate), the driver retries the list of servers in the same sequence twice (`ConnectionRetryCount=2`). Because the connection retry delay has been set to five seconds (`ConnectionRetryDelay=5`), the driver waits five seconds between retry passes.

Refcursors

The driver returns the cursor name for refcursors. The application must fetch the actual data from the refcursor using the cursor name and must close the cursor before additional processing can be done on the statement. The application must close the cursor regardless of whether it actually fetches data from the cursor.

Returning and inserting/updating XML data

The driver supports the XML data type for PostgreSQL 8.3 and higher.

Returning XML data

The driver returns XML data as character data. Your application can use the following methods to return data stored in XML columns as character data:

- ResultSet.getString()
- ResultSet.getCharacterStream()
- ResultSet.getClob()
- CallableStatement.getString()
- CallableStatement.getClob()

The driver converts the XML data returned from the database server from the UTF-8 encoding used by the database server to the UTF-16 Java String encoding.

Your application can use the following method to return data stored in XML columns as ASCII data:

- ResultSet.getAsciiStream()

The driver converts the XML data returned from the database server from the UTF-8 encoding to the ISO-8859-1 (latin1) encoding.

Note: The conversion caused by using the getAsciiStream() method may create XML that is not well-formed because the content encoding is not the default encoding and does not contain an XML declaration specifying the content encoding. Do not use the getAsciiStream() method if your application requires well-formed XML.

Inserting/updating XML data

The driver can insert or update XML data as character or binary data.

Character data

Your application can use the following methods to insert or update XML data as character data:

- PreparedStatement.setString()
- PreparedStatement.setCharacterStream()
- PreparedStatement.setClob()
- PreparedStatement.setObject()
- ResultSet.updateString()
- ResultSet.updateCharacterStream()
- ResultSet.updateClob()
- ReultSet.updateObject()

The driver converts the character representation of the data to the XML character set used by the database server and sends the converted XML data to the server. The driver does not parse or remove any XML processing instructions.

Your application can update XML data as ASCII data using the following methods:

- PreparedStatement.setAsciiStream()
- ResultSet.updateAsciiStream()

The driver interprets the data supplied to these methods using the ISO-8859-1 (latin 1) encoding. The driver converts the data from ISO-8859-1 to the XML character set used by the database server and sends the converted XML data to the server.

Binary data

Your application can use the following methods to insert or update XML data as binary data:

- PreparedStatement.setBytes()
- PreparedStatement.setBinaryStream()
- PreparedStatement.setBlob()
- PreparedStatement.setObject()
- ResultSet.updateBytes()
- ResultSet.updateBinaryStream()
- ResultSet.updateBlob()
- ReultSet.updateObject()

The driver does not apply any data conversions when sending XML data to the database server.

Isolation levels

The driver supports the Read Committed, Read Uncommitted, Repeatable Read, and Serializable isolation levels. The default is Read Committed.

Using scrollable cursors

The driver supports scroll-insensitive result sets and updatable result sets.

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

JTA support

JDBC distributed transactions through JTA are not supported by the driver.

Batch inserts

The BatchMechanism connection property determines how the driver manages batch inserts. When BatchMechanism is set to `nativeBatch`, the driver uses the PostgreSQL native batch protocol to insert all batched parameters. When BatchMechanism is set to `copy`, the driver leverages the PostgresSQL COPY command for substantial performance gains. See "BatchMechanism" for details.

See also

BatchMechanism on page 91

Large object (LOB) support

The PostgreSQL driver allows you to retrieve and update long data, specifically LONGVARBINARY and LONGVARCHAR [18] data, using JDBC methods designed for Blobs and Clobs. When using these methods to update long data as Blobs or Clobs, the updates are made to the local copy of the data contained in the Blob or Clob object.

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

- Provides random access to data

[18] You may determine whether *Character varying* columns are described as VARCHAR or LONGVARCHAR by setting the VarcharClobThreshold on page 135 connection property.

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

To provide these benefits normally associated with Blobs and Clobs, data must be cached. Because data is cached, your application will incur a performance penalty, particularly if data is read once sequentially. This performance penalty can be severe if the size of the long data is larger than available memory.

Using COPY command

The driver supports a customized version of the PostgreSQL COPY command. It provides an additional keyword, LOCALFILE, to allow you to copy data from or to standard file-system files that are stored anywhere on your network, not just on the database server. The file types supported by the driver are .txt and .csv.

Note: The customized COPY command supports all the options supported by the PostgreSQL COPY command.

Syntax

To copy data from a file to an existing table in the database, use the following syntax:

```
COPY tablename[(columnname[,...])] from {LOCALFILE 'filepath'} [[WITH](option[,...])]
```

where:

tablename

is the name of the table that you are copying data to.

columnname

is the name of the column available in the table you are copying data to.

filepath

is the absolute path of the file you are copying data from.

option

can be one of the following: FORMAT, OIDS, DELIMITER, NULL, HEADER, QUOTE, ESCAPE, FORCE_QUOTE, FORCE_NOT_NULL, and ENCODING.

Example:

```
COPY testcopytable(intcol, varcharcol, charcol) from {LOCALFILE 'C:\\Users\\abc\\data.txt'}
 with DELIMITER ','
```

To copy data from an existing table in the database to a file, use the following syntax:

```
COPY tablename[(columnname[,...])] to {LOCALFILE 'filepath'} [[WITH](option[,...])]
```

where:

tablename

is the name of the table that you are copying data from.

columnname

is the name of the column available in the table you are copying data from.

filepath

is the absolute path of the file you are copying data to.

option

can be one of the following: FORMAT, OIDS, DELIMITER, NULL, HEADER, QUOTE, ESCAPE, FORCE_QUOTE, FORCE_NOT_NULL, and ENCODING.

Example:

```
COPY testcopytable(intcol, varcharcol, charcol) to {LOCALFILE
'/home/users/abc/copydata.csv'} with DELIMITER ','
```

Parameter metadata support

The driver supports returning parameter metadata. The driver returns only the parameter's data type.

User-defined function results

PostgreSQL provides functionality to create user-defined functions. PostgreSQL does not define a call mechanism for invoking a user-defined function. User-defined functions must be invoked via a SQL statement. For example, given a function defined as:

```
CREATE table foo (intcol int, varcharcol varchar(123))
CREATE or REPLACE FUNCTION insertFoo
  (IN idVal int, IN nameVal varchar) RETURNS void
  AS $$
    insert into foo values ($1, $2);
  $$
  LANGUAGE SQL;
```

must be invoked natively as:

```
SELECT * FROM insertFoo(100, 'Mark')
```

even though the function does not return a value or results. The Select SQL statement returns a result set that has one column named insertFoo and no row data.

The PostgreSQL driver supports invoking user-defined functions using the JDBC call Escape. The previously described function can be invoked using:

```
{call insertFoo(100, 'Mark')}
```

PostgreSQL functions return data from functions as a result set. If multiple output parameters are specified, the values for the output parameters are returned as columns in the result set. For example, the function defined as:

```
CREATE or REPLACE FUNCTION addValues(in v1 int, in v2 int)
  RETURNS int
  AS $$
    SELECT $1 + $2;
  $$
  LANGUAGE SQL;
```

returns a result set with a single column of type INTEGER, whereas the function defined as:

```
CREATE or REPLACE FUNCTION selectFooRow2
  (IN idVal int, OUT id int, OUT name varchar)
  AS $$
    select intcol, varcharcol from foo where intcol = $1;
  $$
  LANGUAGE SQL
```

returns a result set that contains two columns, a INTEGER id column and a VARCHAR name column.

In addition, when calling PostgreSQL functions that contain output parameters, the native syntax requires that the output parameter values be omitted from the function call. This, in addition to output parameter values being returned as a result set, makes the PostgreSQL behavior of calling functions different from most other databases.

Note: When using the ?= version of the call escape on a function that returns a set of values, only the first result in the set will be returned.

The PostgreSQL driver provides a mechanism that makes the invoking of functions more consistent with how other databases behave. In particular, the PostgreSQL driver allows parameter markers for output parameters to be specified in the function argument list when the Escape call is used. The driver extracts the output parameter values from the result set returned by the server and makes the values available via the getxxx methods on a CallableStatement.

For example, the function `selectFooRow2` described previously can be invoked as:

```
sql = "{call selectFooRow2 (?,?,?)}";
CallableStatement cSTMT = connection.prepareCall(sql);
cSTMT.setInt(1, idVal);
cSTMT.registerOutParameter(2, Types.INTEGER);
cSTMT.registerOutParameter(3, Types.VARCHAR);
cSTMT.execute();
int myID = cSTMT.getInt(2);
String myName = cSTMT.getString(3);
```

The values of the id and name output parameters are returned in the `myID` and `myName` variables.

An error is returned if the number of output parameters registered when the function is executed is less than the number of output parameters defined in the function. If no output parameters are registered to a function call, the driver returns the output parameters as a result set.

PostgreSQL can also return results from a function as a refcursor. There can be, at most, one refcursor per result; however, a function can return multiple results where each result is a refcursor. See Refcursors on page 69 for more information.

ResultSet metadata support

If your application requires table name information, the driver can return table name information in ResultSet metadata for Select statements. If you set the ResultSetMetaDataOptions property to 1, the driver performs additional processing to determine the correct table name for each column in the result set when the ResultSetMetaData.getTableName() method is called. Otherwise, the getTableName() method may return an empty string for each column in the result set.

When the ResultSetMetaDataOptions property is set to 1 and the ResultSetMetaData.getTableName() method is called, the table name information that is returned by the driver depends on whether the column in a result set maps to a column in a table in the database. For each column in a result set that maps to a column in a table in the database, the driver returns the table name associated with that column. For columns in a result set that do not map to a column in a table (for example, aggregates and literals), the driver returns an empty string.

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

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

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

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

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

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

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

Rowset support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets

- JoinRowSets
- JDBCRowSets

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

Auto-generated keys support

The PostgreSQL driver supports retrieving the values of auto-generated keys. An auto-generated key returned by the PostgreSQL driver is the value of a ROWID pseudo column.

An application can return values of auto-generated keys when it executes an Insert statement. How you return these values depends on whether you are using an Insert statement with a Statement object or with a PreparedStatement object, as outlined in the following scenarios:

- When using an Insert statement with a Statement object, the driver supports the following form of the Statement.execute and Statement.executeUpdate methods to instruct the driver to return values of auto-generated keys:
  - `Statement.execute(String sql, int autoGeneratedKeys)`
  - `Statement.execute(String sql, int[] columnIndexes)`
  - `Statement.execute(String sql, String[] columnNames)`
  - `Statement.executeUpdate(String sql, int autoGeneratedKeys)`
  - `Statement.executeUpdate(String sql, int[] columnIndexes)`
  - `Statement.executeUpdate(String sql, String[] columnNames)`
- When using an Insert statement with a PreparedStatement object, the driver supports the following form of the Connection.prepareStatement method to instruct the driver to return values of auto-generated keys:
  - `Connection.prepareStatement(String sql, int autoGeneratedKeys)`
  - `Connection.prepareStatement(String sql, int[] columnIndexes)`
  - `Connection.prepareStatement(String sql, String[] columnNames)`

Note: When returning auto-generated keys, using column names provides better performance than using column indexes.

An application can retrieve values of auto-generated keys using the Statement.getGeneratedKeys() method. This method returns a ResultSet object with a column for each auto-generated key.

Refer to "Designing JDBC applications for performance optimization" in the *Progress DataDirect for JDBC Drivers Reference* for information about how auto-generated keys can improve performance.

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
- User ID/password properties
- Kerberos properties
- Entra ID user and password properties
- Entra ID service principal properties

- AWS IAM properties
- Proxy server properties
- Data encryption properties
- Bulk load properties
- Failover properties
- Timeout properties
- Client information properties
- Statement pooling properties
- Additional properties

General properties

The following table summarizes connection properties required to connect to a database.

Property	Data Source Method	Default
PortNumber on page 117	`setPortNumber` `getPortNumber`	5432
ServerName on page 126	`setServerName` `getServerName`	No default value
DatabaseName on page 101	`setDatabaseName` `getDatabaseName`	No default value

User ID/password authentication properties

The following table summarizes the connection properties required for user ID/password authentication.

Property	Data Source Method	Default
AuthenticationMethod on page 90	`setAuthenticationMethod` `getAuthenticationMethod`	`userIdPassword`
User on page 134	`setUser` `getUser`	No default value
Password on page 117	`setPassword` `getPassword`	No default value

Kerberos authentication properties

The following table summarizes the connection properties required for Kerberos authentication.

Property	Data Source Method	Default
AuthenticationMethod on page 90	setAuthenticationMethod getAuthenticationMethod	userIdPassword
DatabaseName on page 101	setDatabaseName getDatabaseName	No default value
ServicePrincipalName on page 127	setServicePrincipalName getServicePrincipalName	No default value
LoginConfigName on page 111	setLoginConfigName getLoginConfigName	JDBC_DRIVER_01

Entra ID user and password authentication properties

The following table summarizes the connection properties required for Entra ID user and password authentication.

Property	Data Source Method	Default
AuthenticationMethod on page 90	setAuthenticationMethod getAuthenticationMethod	userIdPassword
User on page 134	setUser getUser	No default value
Password on page 117	setPassword getPassword	No default value
AzureTenantID on page 91	getAzureTenantID setAzureTenantID	No default value

Entra ID service principal authentication properties

The following table summarizes the connection properties required for Entra ID service principal authentication.

Property	Data Source Method	Default
AuthenticationMethod on page 90	setAuthenticationMethod getAuthenticationMethod	userIdPassword
User on page 134	setUser getUser	No default value

Property	Data Source Method	Default
Password on page 117	setPassword getPassword	No default value
AzureTenantID on page 91	public String getAzureTenantID setAzureTenantID	No default value

AWS IAM authentication properties

The following table summarizes the connection properties required for AWS IAM authentication.

Property	Data Source Method	Default
AuthenticationMethod on page 90	setAuthenticationMethod getAuthenticationMethod	userIdPassword
AccessKey on page 87	getAccessKey() setAccessKey(String)	No default value
Region on page 123	getRegionName() setRegionName(String)	No default value
SecretKey on page 126	getSecretKey() setSecretKey(String)	No default value
TrustStore on page 132	setTrustStore getTrustStore	No default value
TrustStorePassword on page 133	setTrustStorePassword getTrustStorePassword	No default value

Proxy server properties

The following table summarizes proxy server connection properties.

Property	Data Source Method	Default
ProxyHost on page 122	getProxyHost setProxyHost	No default value
ProxyPassword on page 120	getProxyPassword setProxyPassword	No default value

Property	Data Source Method	Default
ProxyPort on page 120	`getProxyPort` `setProxyPort`	`0` which means the default is determined by the ProxyHost property. For HTTP URLs: `80` For HTTPS URLs: `443`
ProxyUser on page 121	`getProxyUser` `setProxyUser`	No default value

Data encryption properties

The following table summarizes connection properties that can be used to enable SSL.

Property	Data Source Method	Default
CryptoProtocolVersion on page 99	`setCryptoProtocolVersion` `getCryptoProtocolVersion`	No default value
EncryptionMethod on page 103	`setEncryptionMethod` `getEncryptionMethod`	`noEncryption`
HostNameInCertificate on page 104	`setHostNameInCertificate` `getHostNameInCertificate`	No default value
KeyStore on page 109	`setKeyStore` `getKeyStore`	No default value
KeyStorePassword on page 110	`setKeyStorePassword` `getKeyStorePassword`	No default value
KeyPassword on page 108	`setKeyPassword` `getKeyPassword`	No default value
TrustStore on page 132	`setTrustStore` `getTrustStore`	No default value
TrustStorePassword on page 133	`setTrustStorePassword` `getTrustStorePassword`	No default value
ValidateServerCertificate on page 134	`setValidateServerCertificate` `getValidateServerCertificate`	`true`

Bulk load properties

The following table contains the only connection property that affects how bulk load works with the driver.

Property	Data Source Method	Default
BulkLoadBatchSize on page 93	setBulkLoadBatchSize getBulkLoadBatchSize	1000

Failover properties

The following table summarizes the connection properties used for configuring failover.

Property	Data Source Method	Default
AlternateServers on page 88	setAlternateServers getAlternateServers	No default value
ConnectionRetryCount on page 97	setConnectionRetryCount getConnectionRetryCount	5
ConnectionRetryDelay on page 98	setConnectionRetryDelay getConnectionRetryDelay	1 (second)
DatabaseName on page 101	setDatabaseName getDatabaseName	No default value
LoadBalancing on page 111	setLoadBalancing getLoadBalancing	false

Timeout properties

The following table summarizes timeout connection properties.

Property	Data Source Method	Default
EnableCancelTimeout on page 101	setEnableCancelTimeout getEnableCancelTimeout	false
LoginTimeout on page 112	setLoginTimeout getLoginTimeout	0
QueryTimeout on page 123	setQueryTimeout getQueryTimeout	0

Client information properties

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

Property	Data Source Method	Default
AccountingInfo on page 88	setAccountingInfo getAccountingInfo	No default value
ApplicationName on page 89	setApplicationName getApplicationName	No default value
ClientHostName on page 95	setClientHostName getClientHostName	No default value
ClientUser on page 96	setClientUser getClientUser	No default value
ProgramID on page 119	setProgramID getProgramID	No default value

Statement pooling properties

The following table summarizes statement pooling connection properties.

Property	Data Source Method	Default
ImportStatementPool on page 105	setImportStatementPool getImportStatementPool	No default value
MaxPooledStatements on page 115	setMaxPooledStatements getMaxPooledStatements	0
RegisterStatementPoolMonitorMBean	setRegisterStatementPoolMonitorMBean getRegisterStatementPoolMonitorMBean	false

Additional properties

The following table summarizes additional connection properties.

Property	Data Source Method	Default
BatchMechanism on page 91	setBatchMechanism getBatchMechanism	nativeBatch
CallEscapeBehavior on page 94	setCallEscapeBehavior getCallEscapeBehavior	callIfNoReturn

Property	Data Source Method	Default
CodePageOverride on page 96	setCodePageOverride getCodePageOverride	No default value
CatalogOptions on page 94	setCatalogOptions getCatalogOptions	2
ConvertNull on page 99	setConvertNull getConvertNull	1
EnablePrepareThreshold on page 102	setEnablePrepareThreshold getEnablePrepareThreshold	false
ExtendedColumnMetadata on page 104	setExtendedColumnMetadata getExtendedColumnMetadata	false
InitializationString on page 106	setInitializationString getInitializationString	No default value
InsensitiveResultSetBufferSize on page 107	setInsensitiveResultSetBufferSize getInsensitiveResultSetBufferSize	2048
JavaDoubleToString on page 108	setJavaDoubleToString getJavaDoubleToString	false
MaxLongVarcharSize on page 113	setMaxLongVarcharSize getMaxLongVarcharSize	1,073,741,823
MaxNumericPrecision on page 113	setMaxNumericPrecision getMaxNumericPrecision	1000
MaxNumericScale on page 114	setMaxNumericScale getMaxNumericScale	998
MaxStatements on page 116	setMaxStatements getMaxStatements	0
MaxVarcharSize on page 116	setMaxVarcharSize getMaxVarcharSize	10,485,760
PrepareThreshold on page 118	setPrepareThreshold getPrepareThreshold	1

Property	Data Source Method	Default
ResultSetMetaDataOptions on page 125	`setResultSetMetaDataOptions` `getResultSetMetaDataOptions`	`0`
SpyAttributes on page 128	`setSpyAttributes` `getSpyAttributes`	No default value
SupportsCatalogs on page 130	`setSupportsCatalogs` `getSupportsCatalogs`	`true`
TransactionErrorBehavior on page 131	`setTransactionErrorBehavior` `getTransactionErrorBehavior`	`RollbackTransaction`
VarcharClobThreshold on page 135	`setVarcharClobThreshold` `getVarcharClobThreshold`	`32768`

For details, see the following topics:

- AccessKey
- AccountingInfo
- AlternateServers
- ApplicationName
- AuthenticationMethod
- AzureTenantID
- BatchMechanism
- BulkLoadBatchSize
- CallEscapeBehavior
- CatalogOptions
- ClientHostName
- ClientUser
- CodePageOverride
- ConnectionRetryCount
- ConnectionRetryDelay
- ConvertNull
- CryptoProtocolVersion
- DatabaseName

- EnableCancelTimeout
- EnablePrepareThreshold
- EncryptionMethod
- ExtendedColumnMetadata
- HostNameInCertificate
- ImportStatementPool
- InitializationString
- InsensitiveResultSetBufferSize
- JavaDoubleToString
- KeyPassword
- KeyStore
- KeyStorePassword
- LoadBalancing
- LoginConfigName
- LoginTimeout
- MaxLongVarcharSize
- MaxNumericPrecision
- MaxNumericScale
- MaxPooledStatements
- MaxStatements
- MaxVarcharSize
- Password
- PortNumber
- PrepareThreshold
- ProgramID
- ProxyPassword
- ProxyPort
- ProxyUser
- ProxyHost
- QueryTimeout
- Region
- RegisterStatementPoolMonitorMBean
- ResultSetMetaDataOptions

- SecretKey
- ServerName
- ServicePrincipalName
- SpyAttributes
- SupportsCatalogs
- TransactionErrorBehavior
- TrustStore
- TrustStorePassword
- User
- ValidateServerCertificate
- VarcharClobThreshold

AccessKey

Purpose

Specifies the access key ID used for authenticating with AWS credentials (`AuthenticationMethod=AWSIAM`).

Valid Values

`String`

where:

`String`

is the access key ID for your IAM user or AWS account root user.

Data Source Methods

```
public String getAccessKey()
```

```
public void setAccessKey(String)
```

Default

No default value

Data Type

String

See also

AWS IAM authentication on page 57

AccountingInfo

Purpose

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

Valid Values

string

where:

string

is the accounting information.

Data Source Methods

```
public String getAccountingInfo()
```

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

Default

No default value

Data Type

String

See also

Using client information on page 62

AlternateServers

Purpose

Defines a list of alternate database servers that is used to failover new or lost connections, depending on the failover method selected.

Valid Values

```
(servername1[:port1][;DatabaseName=value][,servername2[:port2]
[;DatabaseName=value]]...)
```

The server name (*servername1*, *servername2*, and so on) is required for each alternate server entry. Port number (*port1*, *port2*, and so on) and connection properties (*property=value*) are optional for each alternate server entry. If the port is unspecified, the port number of the primary server is used. If the port number of the primary server is unspecified, the default port number of 5432 is used.

DatabaseName is an optional connection property.

Data Source Methods

```
public String getAlternateServers()
```

```
public void setAlternateServers(String)
```

Default

No default value

Data Type

String

Example

The following URL contains alternate server entries for server2 and server3. The alternate server entries contain the optional DatabaseName property.

```
jdbc:datadirect:postgresql://server1:5432;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:5432;DatabaseName=TEST2,
server3:5432;DatabaseName=TEST3)
```

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

See also

Configuring failover on page 67

ApplicationName

Purpose

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

Valid Values

string

where:

string

is the name of the application.

Data Source Methods

```
public String getApplicationName()
```

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

Default

No default value

Data Type

String

See also

Using client information on page 62

AuthenticationMethod

Purpose

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

Valid Values

`userIdPassword` | `kerberos` | `entraIDPassword` | `entraIDServicePrincipal` | `AWSIAM`

Behavior

If set to `userIdPassword`, the driver uses user ID/password authentication. The driver supports the MD5, SCRAM-SHA-256, and SCRAM-SHA-256-PLUS password hashing mechanisms. During connection, the driver detects and uses the most secure method supported by the server. You must also provide a value for the User and Password connection properties. If a user ID and password is not specified, the driver throws an exception.

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

If set to `entraIDPassword`, the driver uses the Entra ID user and password for authentication. The AzureTenantID property must be specified.

If set to `entraIDServicePrincipal`, the driver retrieves an access token by authenticating using the client ID of the logical server and the client secret of your Entra ID application. The AzureTenantID property must be specified.

If set to `AWSIAM`, the driver uses AWS (Amazon Web Services) credentials for authentication. You must also configure the AccessKey, Region, and SecretKey properties.

Notes

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

Data Source Methods

```
public String getAuthenticationMethod()
```

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

Default

`userIdPassword`

Data Type

String

See also

- Authentication on page 51
- ServicePrincipalName on page 127

AzureTenantID

Purpose

Specifies the Azure tenant ID associated with your PostgreSQL server. This property is used when authenticating with Microsoft Entra ID authentication using service principal users (AuthenticationMethod=`entraIDServicePrincipal`)

Valid Values

`string`

where:

`string`

is the ID of the tenant.

Data Source Methods

```
public String getAzureTenantID()
```

```
public void setAzureTenantID(String)
```

Default

No default value

Data Type

String

See also

Microsoft Entra ID authentication on page 56

BatchMechanism

Purpose

Determines the mechanism that is used to execute batch inserts.

Valid Values

`nativeBatch` | `copy`

Behavior

If set to `nativeBatch`, the driver uses the PostgreSQL native batch protocol to insert all batched parameters.

If set to `copy`, the driver creates an in-memory representation of a CSV file based on all the rows contained in a parameter array, and the PostgreSQL `COPY` command is executed to insert the rows from the CSV file into the target table.

Notes

- BatchMechanism determines the mechanism used to perform batch inserts only. For update and delete batch operations, the driver uses the native batch mechanism to handle the request.
- When `BatchMechanism=nativeBatch`, individual update counts are returned for each statement or parameter set in the batch as required by the JDBC 3.0 specification.
- When `BatchMechanism=copy`, substantial performance gains can be made. However, the following limitations apply.
  - Individual update counts are not returned. However, the total number of update counts are returned for each statement or parameter set in the batch.
  - All columns in the insert command should be bound with parameters. Mixed column binding statements (some columns bound as parameters while others are bound with literals) are not supported. Mixed column binding statements will result in the following error.

    ```
    [DataDirect][PostgreSQL JDBC Driver][PostgreSQL] missing data for column "column_name"
    ```

  - The entire batch insert is ATOMIC. If any issues are encountered, the entire operation fails and no rows are inserted.

Data Source Methods

```
public String getBatchMechanism()
```

```
public void setBatchMechanism(String)
```

Default

`nativeBatch`

Data Type

String

See also

- Performance considerations on page 63
- Batch inserts on page 71

BulkLoadBatchSize

Purpose

Provides a suggestion to the driver for the number of rows to load to the database at a time when bulk loading data. Performance can be improved by increasing the number of rows the driver loads at a time because fewer network round trips are required. Be aware that increasing the number of rows that are loaded also causes the driver to consume more memory on the client.

Valid Values

x

where:

x

is a positive integer that represents a number of rows.

Notes

- This property suggests the number of rows regardless of which bulk load method is used: using a DDBulkLoad object or using bulk load for batch inserts.
- The DDBulkObject.setBatchSize() method overrides the value that is set by this property.

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

Data Source Methods

```
public Long getBulkLoadBatchSize()
```

```
public void setBulkLoadBatchSize(Long)
```

Default

```
1000
```

Data Type

long

See also

Bulk load on page 62

Performance considerations on page 63

CallEscapeBehavior

Purpose

Determines whether the driver calls a user-defined function or a stored procedure when JDBC Call escape syntax ({CALL PROC_NAME(...)} or {?=CALL FUNC_NAME(...)}) is used in a SQL statement.

Valid Values

select | call| callIfNoReturn

Behavior

If set to select, the driver calls a user-defined function.

If set to call, the driver calls a stored procedure.

If set to callIfNoReturn, the driver determines whether to call a user-defined function or a stored procedure based on whether a return value parameter (?=) is specified in the JDBC Call escape syntax. If a return value parameter is specified, the driver calls a user-defined function. If not, the driver calls a stored procedure.

Data Source Methods

```
public String getCallEscapeBehavior()
```

```
public void setCallEscapeBehavior(String)
```

Default

callIfNoReturn

Data Type

String

CatalogOptions

Purpose

Determines which type of metadata information is included in result sets when an application calls DatabaseMetaData methods.

Valid Values

2 | 4

Behavior

If set to 2, the driver queries database catalogs for column information.

If set to 4, a hint is provided to the driver to emulate getColumns() calls using the ResultSetMetaData object instead of querying database catalogs for column information. Using emulation can improve performance because the SQL statement that is formulated by the emulation is less complex than the SQL statement that is formulated using getColumns(). The argument to getColumns() must evaluate to a single table. If it does not, because of a wildcard or null value, for example, the driver reverts to the default behavior for getColumns() calls.

Data Source Methods

```
public Integer getCatalogOptions()
```

```
public void setCatalogOptions(Integer)
```

Default

2

Data Type

int

ClientHostName

Purpose

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

Valid Values

string

where:

string

is the host name of the client machine.

Data Source Methods

```
public String getClientHostName()
```

```
public void setClientHostName(String)
```

Default

No default value

Data Type

String

See also

Using client information on page 62

ClientUser

Purpose

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

Valid Values

string

where:

string

is a valid user ID.

Data Source Methods

```
public String getClientUser()
public void setClientUser(String)
```

Default

No default value

Data Type

String

See also

Using client information on page 62

CodePageOverride

Purpose

Specifies the code page to be used by the driver to convert Character data. The specified code page overrides the default database code page or column collation. All Character data returned from or written to the database is converted using the specified code page.

Valid values

string

where:

string

is the name of a valid code page that is supported by your JVM. For example, `CP950`.

Notes

- By default, the driver automatically determines which code page to use to convert Character data. Use this property only if you need to change the driver's default behavior.

Data Source Methods

```
public String getCodePageOverride()
```

```
public void setCodePageOverride(String)
```

Default

No default value

Data type

String

ConnectionRetryCount

Purpose

Specifies the number of times the driver retries connection attempts to the primary database server, and if specified, alternate servers before returning a connection failure.

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 database server to which it tried to connect.

Example

If this property is set to 2 and alternate servers are specified using the AlternateServers property, the driver retries the list of servers (primary and alternate) 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.
- The driver will not try to reconnect to a server which is not found.

Data Source Methods

```
public Integer getConnectionRetryCount()
```

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

Default

5

Data Type

int

See also

Configuring failover

ConnectionRetryDelay

Purpose

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

Valid Values

0 | *x*

where:

x

is a number of seconds.

Behavior

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

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

Example

If ConnectionRetryCount is set to 2, this property is set to 3, and alternate servers are specified using the AlternateServers property, the driver retries the list of servers (primary and alternate) 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

1 (second)

Data Type

int

See also

Configuring failover

ConvertNull

Purpose

Controls how data conversions are handled for null values.

Valid Values

`0 | 1`

Behavior

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

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

Data Source Methods

```
public Integer getConvertNull()
```

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

Default

`1`

Data Type

int

CryptoProtocolVersion

Purpose

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

Valid Values

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

where:

cryptographic_protocol

is one of the following cryptographic protocols:

TLSv1.3 | TLSv1.2 | TLSv1.1 | TLSv1 | SSLv3 | SSLv2

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

Example

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

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

Notes

- The TLSv1.3 protocol works with Java SE 11 or higher by default.
- To enable the TLSv1.3 protocol when using Java SE 8, set the `jdk.tls.client.protocols` Java system property to `TLSv1.3`. For example, `$ java -Djdk.tls.client.protocols="TLSv1.3" myApp`.

  In the following versions of Oracle JDK and OpenJDK, support for the TLSv1.3 protocol is not enabled by default.

  - Oracle JDK 8u261 or later but earlier than Oracle JDK 8u341
  - OpenJDK 8u272 or later but earlier than OpenJDK 8u352

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

Data Source Methods

```
public String getCryptoProtocolVersion()
```

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

Default

No default value

Data Type

String

See also

- EncryptionMethod on page 103
- Data encryption on page 59

DatabaseName

Purpose

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

Valid Values

string

where:

`string`

is the name of a PostgreSQL database.

Data Source Methods

```
public String getDatabaseName()
```

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

Default

No default value

Data Type

String

EnableCancelTimeout

Purpose

Determines whether a cancel request that is sent by the driver as the result of a query timing out is subject to the same query timeout value as the statement it cancels.

Valid Values

`true` | `false`

If set to `true`, the cancel request times out using the same timeout value, in seconds, that is set for the statement it cancels. For example, if your application calls `Statement.setQueryTimeout(5)` on a statement and that statement is cancelled because its timeout value was exceeded, the driver sends a cancel request that also will time out if its execution exceeds 5 seconds. If the cancel request times out, because the server is down, for example, the driver throws an exception indicating that the cancel request was timed out and the connection is no longer valid.

If set to `false`, the cancel request does not time out.

Data Source Methods

```
public Boolean getEnableCancelTimeout()
public void setEnableCancelTimeout(Boolean)
```

Default

`false`

Data Type

Boolean

EnablePrepareThreshold

Purpose

Determines whether server-side prepared statements are created during the prepareStatement() API call or deferred to the execute() API call.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver defers server-side prepared statements to the execute() API call.

If set to `false`, the driver creates server-side prepared statements during the prepareStatement() API call.

Notes

- When this property is set to `true`, the use of server-side prepared statements is defined by the `PrepareThreshold` property value.
- When this property is set to `false`, the existing prepared statement is used during the execute() API call.

Data Source Methods

```
public Boolean getEnablePrepareThreshold()
public void setEnablePrepareThreshold(Boolean)
```

Default

`false`

Data Type

Boolean

See also

PrepareThreshold

EncryptionMethod

Purpose

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

Valid Values

`noEncryption | SSL | requestSSL`

Behavior

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

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

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

Notes

- When SSL is enabled, the following properties also apply:
  - CryptoProtocolVersion
  - HostNameInCertificate
  - KeyStore (for SSL client authentication)
  - KeyStorePassword (for SSL client authentication)
  - KeyPassword (for SSL client authentication)
  - TrustStore
  - TrustStorePassword
  - ValidateServerCertificate

Data Source Methods

```
public String getEncryptionMethod()
```

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

Default

`noEncryption`

Data Type

String

See also

Data encryption

Performance considerations

ExtendedColumnMetadata

Purpose

Determines how the driver returns column metadata when retrieving results with ResultSetMetaData methods.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver makes an additional roundtrip to the database to retrieve actual values for column metadata. For example, the driver returns nullability as `IS_NULLABLE`, `NOT_NULLABLE`, or `NULLABILITY_UNKNOWN`, depending on the actual status of the column.

If set to `false`, the driver returns only the values for column metadata hardcoded in the driver. For example, the driver returns nullability as `NULLABILITY_UNKNOWN`.

Notes

Setting ExtendedColumnMetadata to enabled may diminish performance because an additional roundtrip to the database is required to retrieve actual values for the column metadata.

Data Source Methods

```
public Boolean getExtendedColumnMetadata()
```

```
public void setExtendedColumnMetadata(Boolean)
```

Default

`false`

Data Type

Boolean

HostNameInCertificate

Purpose

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

Valid Values

`host_name` | #SERVERNAME#

where:

host_name

is a valid host name.

Behavior

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

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

Notes

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

Data Source Methods

```
public String getHostNameInCertificate()
```

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

Default

No default value

Data Type

String

ImportStatementPool

Purpose

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

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

Valid Values

string

where:

string

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

Data Source Methods

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

Default

No default value

Data Type

String

See also

- Statement pooling on page 66
- Performance considerations on page 63
- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

InitializationString

Purpose

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

Valid Values

string

where:

string

is one or multiple SQL commands.

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

Example

```
jdbc:datadirect:postgresql://server1:5432;DatabaseName=test;
InitializationString=(command1;command2)
```

Data Source Methods

```
public String getInitializationString()
```

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

Default

No default value

Data Type

String

InsensitiveResultSetBufferSize

Purpose

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

Valid Values

`-1 | 0 |` *x*

where:

x

is a positive integer that represents the size of the memory buffer.

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 GB. If the size of the result set data exceeds available memory, the driver pages the result set data to disk. 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, the driver pages the result set data to disk. 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

`2048`

Data Type

Integer

See also

Performance considerations

JavaDoubleToString

Purpose

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

Valid Values

`true` | `false`

Behavior

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

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

Data Source Methods

```
public Boolean getJavaDoubleToString()
```

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

Default

`false`

Data Type

Boolean

KeyPassword

Purpose

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

Valid Values

string

where:

string

is a valid password.

Data Source Methods

```
public String getKeyPassword()
```

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

Default

No default value

Data Type

String

KeyStore

Purpose

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

This value overrides the directory of the keystore file that is specified by the javax.net.ssl.keyStore Java system property. If this property is not specified, the keystore directory is specified by the javax.net.ssl.keyStore Java system property.

Valid Values

string

where:

string

is a valid directory of a keystore file.

Notes

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

Data Source Methods

```
public String getKeyStore()
```

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

Default

No default value

Data Type

String

KeyStorePassword

Purpose

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

This value overrides the password of the keystore file that is specified by the javax.net.ssl.keyStorePassword Java system property. If this property is not specified, the keystore password is specified by the javax.net.ssl.keyStorePassword Java system property.

Notes

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

Valid Values

string

where:

string

is a valid password.

Data Source Methods

```
public String getKeyStorePassword()
```

```
public void setKeyStorePassword(String)
```

Default

No default value

Data Type

String

LoadBalancing

Purpose

Determines whether the driver uses client load balancing in its attempts to connect to the database servers (primary and alternate). You can specify one or multiple alternate servers by setting the AlternateServers property.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver uses client load balancing and attempts to connect to the database servers (primary and alternate) in random order. The driver randomly selects from the list of primary and alternate servers which server to connect to first. If that connection fails, the driver again randomly selects from this list of servers until all servers in the list have been tried or a connection is successfully established.

If set to `false`, the driver does not use client load balancing and connects to each server based on their sequential order (primary server first, then, alternate servers in the order they are specified).

Data Source Methods

```
public Boolean getLoadBalancing()
```

```
public void setLoadBalancing(Boolean)
```

Default

`false`

Data Type

Boolean

See also

Configuring failover

LoginConfigName

Purpose

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

Valid Values

`entry_name`

where:

entry_name

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

Example

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

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

Data Source Methods

```
public String getLoginConfigName()
```

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

Default

JDBC_DRIVER_01

Data Type

String

See also

- AuthenticationMethod on page 90
- The JAAS login configuration file on page 54

LoginTimeout

Purpose

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

Valid Values

0 | *x*

where:

x

is a positive integer that represents a number of seconds.

Behavior

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

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

Data Source Methods

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

Default

0

Data Type

Integer

MaxLongVarcharSize

Purpose

Determines the maximum size of LONGVARCHAR columns when described within the result set metadata.

Valid Values

x

where:

x

is an integer greater than or equal to 1 and less than or equal to 1,073,741,823.

Data Source Methods

```
public Integer getMaxLongVarcharSize()
public void setMaxLongVarcharSize(Integer)
```

Default

1,073,741,823

Data Type

Integer

MaxNumericPrecision

Purpose

Determines the maximum precision of NUMERIC columns when described within the result set metadata.

Valid Values

x

where:

x

is an integer greater than or equal to 1 and less than or equal to 1000.

Data Source Methods

```
public Integer getMaxNumericPrecision()
public void setMaxNumericPrecision(Integer)
```

Default

1000

Data Type

Integer

MaxNumericScale

Purpose

Determines the maximum scale of NUMERIC columns when described within the result set metadata.

Valid Values

x

where:

x

is an integer greater than or equal to 0 and less than or equal to 998.

Data Source Methods

```
public Integer getMaxNumericScale()
public void setMaxNumericScale(Integer)
```

Default

998

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.

Notes

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

Example

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

Data Source Methods

```
public Integer getMaxPooledStatements()
```

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

Default

`0`

Data Type

Integer

See also

- Statement pooling on page 66
- Performance considerations on page 63
- RegisterStatementPoolMonitorMBean on page 124
- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

MaxStatements

Purpose

An alias for the MaxPooledStatements property.

MaxVarcharSize

Purpose

Determines the maximum size of VARCHAR columns when described within the result set metadata.

Valid Values

x

where:

x

is an integer greater than or equal to `1` and less than or equal to `10485760`.

Notes

When string functions are used within the Select list of a Select statement, the driver describes the resulting string value with a size of 10,485,760. For instance, concatenating two columns that are each defined as VARCHAR(10) will result in a VARCHAR(10485760) rather than a VARCHAR(20). The unnecessarily long size can result in undesirable behavior in some JDBC applications.

Data Source Methods

```
public Integer getMaxVarcharSize()
```

```
public void setMaxVarcharSize(Integer)
```

Default

10,485,760

Data Type

Integer

Password

Purpose

Specifies a valid password that is used to connect to your database. A password is required if user ID/Password or Entra ID authentication is enabled on your database. The following values are used for authentication:

- The password that is used to connect to your database. This is used when `userIdPassword` authentication is enabled.
- The password of the Entra ID user. This is used when `entraIDPassword` authentication is enabled.
- The client secret of the service principal. This is used when `entraIDServicePrincipal` authentication is enabled.

Valid Values

`string`

where:

`string`

is a valid password or the client secret. The password is case-sensitive.

Data Source Methods

```
public String getPassword()
```

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

Default

No default value

Data Type

String

PortNumber

Purpose

The TCP port of the primary database server that is listening for connections to the PostgreSQL database.

This property is supported only for data source connections.

Valid Values

`port`

where:

port

is the port number.

Data Source Methods

```
public Integer getPortNumber()
public void setPortNumber(Integer)
```

Default

5432

Data Type

Integer

PrepareThreshold

Purpose

Specifies the number of PreparedStatement executions that must occur before the driver begins using server-side prepared statements.

Valid Values

0 | 1 | *x*

where:

x

is a positive integer that represents the number of PreparedStatement executions at which the driver begins using server-side prepared statements.

Behavior

If set to 0, the driver never uses server-side prepared statements.

If set to 1, the driver uses server-side prepared statements by default.

If set to *x*, the driver uses server-side prepared statements when the number of PreparedStatement executions reaches the specified value.

Example

If PrepareThreshold is set to 5, the driver starts using server-side prepared statements on the fifth execution of the same PreparedStatement object.

Note

- This property is applicable when the EnablePrepareThreshold property is set to true.
- The value 0 is recommended when using a load balancer with a server that cannot support server-side prepare operations.

- The value `1` is recommended when an application executes the same parameterized DML operation multiple times.
- The value *x* is recommended when the server-side prepare operation is used only for frequently executing queries.

Data Source Methods

```
public Integer getPrepareThreshold()
```

```
public void setPrepareThreshold(Integer)
```

Default

`1`

Data Type

int

See also

EnablePrepareThreshold

QueryTimeout

ProgramID

Purpose

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

Valid Values

`string`

where:

`string`

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

Example

```
DDJ04200
```

Data Source Methods

```
public String getProgramID()
```

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

Default

No default value

Data Type

String

See also

Using client information on page 62

ProxyPassword

Purpose

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

Valid Values

password

where:

password

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

Data Source Methods

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

Default Value

No default value

Data Type

String

See also

ProxyPort on page 120
ProxyUser on page 121
ProxyHost on page 122
Connection URL examples on page 14

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

ProxyUser on page 121
ProxyHost on page 122
ProxyPassword on page 120
Connection URL examples on page 14

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 122
ProxyPort on page 120
ProxyPassword on page 120
Connection URL examples on page 14

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

ProxyUser on page 121
ProxyPort on page 120
ProxyPassword on page 120
Connection URL examples on page 14

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 positive integer that represents 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 set by this connection option, 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

`0`

Data Type

Integer

Region

Purpose

Specifies the name of the region that hosts your AWS server when using AWS credentials to authenticate (`AuthenticationMethod=AWSIAM`).

Valid Values

`String`

where:

`String`

is the name of the region that hosts your AWS server. For example, `us-east-1` or `us-east-2`.

Data Source Methods

```
public String getRegionName()
```

```
public void setRegionName(String)
```

Default Value

No default value

Data Type

String

See also

AWS IAM authentication on page 57

RegisterStatementPoolMonitorMBean

Purpose

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

Valid Values

`true` | `false`

Behavior

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

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

Notes

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

Data Source Methods

```
public Boolean getRegisterStatementPoolMonitorMBean()
```

```
public void setRegisterStatementPoolMonitorMBean(Boolean)
```

Default

`false`

Data Type

Boolean

See also

- MaxPooledStatements on page 115
- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

ResultSetMetaDataOptions

Purpose

Determines whether the driver returns table name information in the ResultSet metadata for Select statements.

Valid Values

`0 | 1`

Behavior

If set to `0` and the ResultSetMetaData.getTableName() method is called, the driver does not perform additional processing to determine the correct table name for each column in the result set. The getTableName() method may return an empty string for each column in the result set.

If set to `1` and the ResultSetMetaData.getTableName() method is called, the driver performs additional processing to determine the correct table name for each column in the result set. The driver returns schema name and catalog name information when the ResultSetMetaData.getSchemaName() and ResultSetMetaData.getCatalogName() methods are called if the driver can determine that information.

Data Source Methods

```
public Integer getResultSetMetaDataOptions()
```

```
public void setResultSetMetaDataOptions(Integer)
```

Default

`0`

Data Type

Integer

See also

Performance considerations

SecretKey

Purpose

Specifies the secret access key used for authenticating with AWS credentials (`AuthenticationMethod=AWSIAM`).

Valid Values

`String`

where:

`String`

is the secret access key for your IAM user or AWS account root user.

Data Source Methods

```
public String getSecretKey()
```

```
public void setSecretKey(String)
```

Default

No default value

Data Type

String

See also

AWS IAM authentication on page 57

ServerName

Purpose

REQUIRED. Specifies either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the primary database server.

This property is supported only for data source connections.

Valid Values

`string`

where:

`string`

is a valid IP address or server name.

Example

`122.23.15.12` or `PostgeSQLServer`

Data Source Methods

```
public String getServerName()
```

```
public void setServerName(String)
```

Default

No default value

Data Type

String

ServicePrincipalName

Purpose

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

Valid Values

`ServicePrincipalName`

where:

`ServicePrincipalName`

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

Specify the service principal name using the following format:

`Service_Name/Fully_Qualified_Domain_Name@REALM``.COM`

where:

`Service_Name`

is the name of the service hosting the instance. It is the same value as the `krb_svrname` configuration parameter on the server. The default is `postgres.`

`Fully_Qualified_Domain_Name`

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

`REALM.COM`

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

Example

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

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

Notes

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

Data Source Methods

```
public String getServicePrincipalName()
```

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

Default

No default value

See also

- Authentication on page 51
- AuthenticationMethod on page 90
- Kerberos authentication on page 52

SpyAttributes

Purpose

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

Valid Values

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

where:

spy_attribute

is any valid DataDirect Spy attribute.

Behavior

Attribute	Description
`linelimit=`*`numberofchars`*	Sets the maximum number of characters that DataDirect Spy logs on a single line. The default is 0 (no maximum limit).

Attribute	Description
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.
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.

Attribute	Description
logTName= { yes | no }	Specifies whether DataDirect Spy logs the name of the current thread. The default is no.
timestamp= { yes | no }	Specifies whether a timestamp is included on each line of the DataDirect Spy log. The default is yes.

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.
- For more information, refer to "Tracking JDBC calls with DataDirect Spy" in the *Progress DataDirect for JDBC Drivers Reference*.

Data Source Methods

```
public String getSpyAttributes()
```

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

Default

No default value

Data Type

String

SupportsCatalogs

Purpose

Enables support for catalogs. While PostgreSQL has the notion of catalogs (or databases), you can only select from tables that reside within the catalog you specified at connect time. Catalogs cannot be changed after connecting. Therefore, most applications behave better if you do not indicate support for catalogs.

Valid Values

`true | false`

Behavior

If set to `true`, the driver returns the database as the catalog for catalog calls, for example, getTables and getColumns.

If set to `false`, the driver returns NULL for the catalog in catalog calls.

Notes

The SupportsCatalogs connection property affects the following catalog methods:

- getCatalogSeparator
- getCatalogTerm
- getMaxCatalogNameLength
- isCatalogAtStart
- supportsCatalogsInDataManipulation
- supportsCatalogsInProcedureCalls
- supportsCatalogsInTableDefinitions
- supportsCatalogsInIndexDefinitions
- supportsCatalogsInPrivilegeDefinitions

Data Source Methods

```
public Boolean getSupportsCatalogs()
public void setSupportsCatalogs(Boolean)
```

Default

`true`

Data Type

Boolean

TransactionErrorBehavior

Purpose

Determines how the driver handles errors that occur within a transaction. When an error occurs in a transaction, the PostgreSQL server does not allow any operations on the connection except for rolling back the transaction.

Valid Values

```
none|RollbackTransaction|RollbackSavepoint
```

Behavior

If set to `none`, the driver does not roll back the transaction when an error occurs. The application must handle the error and roll back the transaction. Any operation on the statement other than a rollback results in an error.

If set to `RollbackTransaction`, the driver rolls back the transaction when an error occurs. In addition to the original error message, the driver posts an error message indicating that the transaction has been rolled back.

If set to RollbackSavepoint, the driver rolls back the transaction to the last savepoint when an error is detected. In manual commit mode, the driver automatically sets a savepoint after each statement issued. This value makes transaction behavior resemble that of most other database system types, but uses more resources on the database server and may incur a slight performance penalty.

Data Source Methods

```
public String getTransactionErrorBehavior()
```

```
public void setTransactionErrorBehavior(String)
```

Default

`RollbackTransaction`

Data Type

String

TrustStore

Purpose

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

This value overrides the directory of the truststore file that is specified by the javax.net.ssl.trustStore Java system property. If this property is not specified, the truststore directory is specified by the javax.net.ssl.trustStore Java system property.

This property is ignored if ValidateServerCertificate=`false`.

Valid Values

`string`

The directory of the truststore file.

Data Source Methods

```
public String getTrustStore()
```

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

Default

No default value

Data Type

String

TrustStorePassword

Purpose

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

This value overrides the password of the truststore file that is specified by the javax.net.ssl.trustStorePassword Java system property. If this property is not specified, the truststore password is specified by the javax.net.ssl.trustStorePassword Java system property.

This property is ignored if ValidateServerCertificate=`false`.

Valid Values

`string`

where:

`string`

is a valid password for the truststore file.

Data Source Methods

```
public String getTrustStorePassword()
```

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

Default

No default value

Data Type

String

User

Purpose

Specifies the user name that is used to connect to the database. A user name is required if user ID/password or Entra ID authentication is enabled on your database. The following identifiers are used for authentication:

- The user name that is used to connect to your database. This is used when `userIdPassword` authentication is enabled.
- The Entra ID user name. This is used when `entraIDPassword` authentication is enabled.
- The client ID of the service principal. This is used when `entraIDServicePrincipal` authentication is enabled.

Valid Values

`string`

where:

`string`

is a valid user name or the client ID. The user name is case-sensitive.

Data Source Methods

`public String getUser()`

`public void setUser(String)`

Default

No default value

Data Type

String

ValidateServerCertificate

Purpose

Determines whether the driver validates the certificate that is sent by the database server when SSL encryption is enabled (EncryptionMethod=`SSL`). When using SSL server authentication, any certificate that is sent by the server must be issued by a trusted Certificate Authority (CA). Allowing the driver to trust any certificate that is returned from the server even if the issuer is not a trusted CA is useful in test environments because it eliminates the need to specify truststore information on each client in the test environment.

Valid Values

`true` | `false`

Behavior

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

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

Notes

- Truststore information is specified using the TrustStore and TrustStorePassword properties or by using Java system properties.

Data Source Methods

```
public Boolean getValidateServerCertificate()
```

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

Default

`true`

Data Type

Boolean

VarcharClobThreshold

Purpose

Enables CLOB functionality when handling character data. Determines whether columns of the *Character varying* data type will be described as VARCHAR or LONGVARCHAR (CLOB).

Valid Values

x

where:

x

is a number of characters.

Behavior

If the width of a *Character varying* column is greater than *x*, the *Character varying* data type is described as LONGVARCHAR and CLOB functionality is enabled. The driver allows you to retrieve and update long data by using JDBC methods designed for Clobs. This functionality provides random access to data and allows searching for patterns in the data. To provide these benefits, data must be cached. Because data is cached, your application will incur a performance penalty, particularly if data is read once sequentially.

If the width of a *Character varying* column is less than or eqal to *x*, the *Character varying* data type is described as VARCHAR. If you want to avoid the performance penalties associated with CLOB functionality, you should set this value at a value greater than the maximum *Character varying* column width your application handles.

Data Source Methods

```
public Varchar getVarcharClobThreshold()
```

```
public void setVarcharClobThreshold(Varchar)
```

Default

32768

Data Type

VARCHAR or LONGVARCHAR

See also

Performance considerations on page 63

Large object (LOB) support on page 71