



**Progress®**

**DataDirect Connect®**

for JDBC™

User's Guide and Reference

Release 4.1  
October 2009

© 2009 Progress Software Corporation. All rights reserved. Printed in the U.S.A.

DataDirect, DataDirect Connect, DataDirect Connect64, DataDirect Spy, DataDirect Test, DataDirect XML Converters, DataDirect XQuery, OpenAccess, Sequelink, Stylus Studio, and SupportLink are trademarks or registered trademarks of Progress Software Corporation or one of its subsidiaries or affiliates in the United States and other countries. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. MySQL and MySQL Enterprise are registered trademarks of MySQL AB in the United States, the European Union and other countries.

Other company or product names mentioned herein may be trademarks or registered trademarks of their respective companies.

**DataDirect products for the Microsoft SQL Server database:**

These products contain a licensed implementation of the Microsoft TDS Protocol.

**DataDirect Connect for ODBC, DataDirect Connect64 for ODBC, and DataDirect Sequelink include:**

ICU Copyright © 1995-2003 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

Software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>). Copyright © 1998-2006 The OpenSSL Project. All rights reserved. And Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved.

**DataDirect Sequelink includes:**

Portions created by Eric Young are Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All Rights Reserved. OpenLDAP, Copyright © 1999-2003 The OpenLDAP Foundation, Redwood City, California, US. All rights reserved.

**DataDirect OpenAccess SDK client for ODBC, DataDirect OpenAccess SDK client for ADO, DataDirect Open Access SDK client for JDBC, and DataDirect OpenAccess SDK server include:** DataDirect Sequelink.

No part of this publication, with the exception of the software product user documentation contained in electronic format, may be copied, photocopied, reproduced, transmitted, transcribed, or reduced to any electronic medium or machine-readable form without prior written consent of DataDirect Technologies.

Licensees may duplicate the software product user documentation contained on a CD-ROM or DVD, but only to the extent necessary to support the users authorized access to the software under the license agreement. Any reproduction of the documentation, regardless of whether the documentation is reproduced in whole or in part, must be accompanied by this copyright statement in its entirety, without modification.

# Table of Contents

<b>Preface .....</b>	<b>19</b>
What Is DataDirect Connect® for JDBC? .....	19
Using This Book .....	20
About the Product Documentation .....	22
HTML Version.....	22
PDF Version .....	23
Typographical Conventions.....	25
Contacting Technical Support.....	26
<b>1 Quick Start Connect.....</b>	<b>29</b>
Connecting to a Database.....	29
1. Setting the Classpath.....	29
2. Registering the Driver .....	30
3. Passing the Connection URL .....	31
Testing the Connection .....	32
Using the Performance Tuning Wizard .....	33
Starting the Wizard.....	33
Tuning Performance Using the Wizard .....	34
<b>2 Using DataDirect Connect® for JDBC Drivers .....</b>	<b>35</b>
About the Product.....	35
The Drivers .....	36
DataDirect Test™ .....	36
DataDirect Spy™ .....	37
DataDirect Connection Pool Manager .....	37
J2EE Connector Architecture Resource Adapters .....	37

Specifying Connection Properties . . . . .	43
Providing Connection Information . . . . .	43
Optimizing Performance . . . . .	44
Connecting Through the JDBC Driver Manager . . . . .	44
Registering the Drivers . . . . .	45
Specifying Connection URLs . . . . .	46
Connecting Through Data Sources . . . . .	48
How Data Sources Are Implemented . . . . .	48
Creating Data Sources . . . . .	49
Calling a Data Source in an Application . . . . .	49
Using IP Addresses . . . . .	50
Using Connection Pooling . . . . .	52
Understanding Connection Pooling . . . . .	53
Using Reauthentication . . . . .	55
Using Statement Pooling . . . . .	57
Using Failover . . . . .	58
Connection Failover . . . . .	59
Extended Connection Failover . . . . .	61
Select Failover . . . . .	62
Guidelines for Primary and Alternate Servers . . . . .	63
Using Client Load Balancing . . . . .	65
Using Connection Retry . . . . .	66
Using Security . . . . .	67
Authentication . . . . .	67
Data Encryption Across the Network . . . . .	69
SSL Encryption . . . . .	70
Storing and Returning Client Information for Connections . . . . .	75
Using DataDirect Bulk Load . . . . .	75
Testing Connections . . . . .	76

Required Permissions for the Java 2 Platform .....	77
Permissions for Establishing Connections .....	77
Granting Access to Java Properties.....	78
Granting Access to Temporary Files .....	78
Granting Access to Oracle tnsnames.ora Files .....	79
Permissions for Kerberos Authentication.....	79
Unicode Support .....	82
Error Handling .....	82
Driver Errors.....	83
Database Errors .....	83
<b>3 The DB2 Driver.....</b>	<b>85</b>
Data Source and Driver Classes.....	86
Connection URL .....	86
J2EE Connector Architecture Resource Adapter Class.....	87
Connection Properties.....	88
Performance Considerations.....	131
DB2 Packages.....	134
Creating DB2 Packages Using the Package Manager ....	135
Creating DB2 Packages Using Connection Properties....	136
Creating DB2 Packages Using Package Creation	
List Files .....	137
Copying the DB2 Packages (z/OS and iSeries) .....	139
Data Types .....	139
Returning and Inserting/Updating XML Data .....	142
Returning XML Data .....	142
Inserting/Updating XML Data.....	145
Authentication.....	146
Using the AuthenticationMethod Property.....	147
Configuring User ID/Password Authentication .....	149
Configuring Kerberos Authentication .....	149

Specifying User Credentials for Kerberos Authentication (Delegation of Credentials).....	152
Obtaining a Kerberos Ticket Granting Ticket.....	154
Configuring Client Authentication .....	155
Data Encryption .....	155
Configuring DB2-Specific Encryption .....	155
Configuring SSL Encryption .....	156
Non-Default Schemas for Catalog Methods.....	157
Reauthentication .....	158
Client Information for Connections .....	160
Workload Manager (WLM) .....	160
SQL Escape Sequences .....	161
Isolation Levels .....	162
Using Scrollable Cursors.....	162
JTA Support.....	163
Large Object (LOB) Support.....	163
Batch Inserts and Updates .....	163
Parameter Metadata Support .....	164
Insert and Update Statements.....	164
Select Statements .....	165
Stored Procedures .....	167
ResultSet Metadata Support .....	167
Rowset Support .....	169
Auto-Generated Keys Support .....	169
Configuring Failover .....	171
Specifying Primary and Alternate Servers.....	172
Specifying Connection Retry .....	174
Failover Properties .....	175
Bulk Load .....	176

<b>4 The Informix Driver . . . . .</b>	<b>177</b>
Data Source and Driver Classes . . . . .	177
Connection URL . . . . .	177
J2EE Connector Architecture Resource Adapter Class . . . . .	178
Connection Properties . . . . .	179
Performance Considerations . . . . .	201
Data Types . . . . .	202
Client Information for Connections . . . . .	204
SQL Escape Sequences . . . . .	204
Isolation Levels . . . . .	204
Using Scrollable Cursors . . . . .	205
Parameter Metadata Support . . . . .	205
Insert and Update Statements . . . . .	205
Select Statements . . . . .	205
Stored Procedures . . . . .	207
ResultSet MetaData Support . . . . .	207
Rowset Support . . . . .	209
Blob and Clob Searches . . . . .	209
Auto-Generated Keys Support . . . . .	210
Configuring Failover . . . . .	211
Specifying Primary and Alternate Servers . . . . .	212
Specifying Connection Retry . . . . .	214
Failover Properties . . . . .	215
<b>5 The MySQL Driver . . . . .</b>	<b>217</b>
Data Source and Driver Classes . . . . .	217
Connection URL . . . . .	218
J2EE Connector Architecture Resource Adapter Class . . . . .	218

Connection Properties .....	219
Performance Considerations .....	247
Data Types.....	248
Data Encryption .....	250
Client Information for Connections .....	251
SQL Escape Sequences .....	251
Isolation Levels.....	251
Using Scrollable Cursors.....	252
Large Object (LOB) Support.....	252
Parameter Metadata Support .....	253
Insert and Update Statements.....	253
Select Statements.....	253
Stored Procedures .....	255
ResultSet MetaData Support.....	255
Rowset Support .....	257
Auto-Generated Keys Support .....	257
Configuring Failover .....	258
Specifying Primary and Alternate Servers.....	259
Specifying Connection Retry .....	261
Failover Properties .....	262
<b>6 The Oracle Driver.....</b>	<b>265</b>
Data Source and Driver Classes .....	265
Connection URL .....	266
J2EE Connector Architecture Resource Adapter Class .....	267
Connection Properties .....	267
Performance Considerations .....	314
Support for Oracle RAC .....	318

Using tnsnames.ora Files . . . . .	319
Connecting to the Database . . . . .	321
Configuring the tnsnames.ora File . . . . .	322
Connecting to Oracle Instances Running in Restricted Mode. . . . .	329
Data Types . . . . .	330
Using Oracle Date/Time Data Types . . . . .	332
Date/Time Session Parameters . . . . .	332
TIMESTAMP Data Type . . . . .	333
TIMESTAMP WITH LOCAL TIME ZONE Data Type . . . . .	333
Returning and Inserting/Updating XML Data . . . . .	335
Returning XML Data . . . . .	335
Inserting/Updating XML Data . . . . .	336
REF CURSOR Data Type . . . . .	338
Authentication. . . . .	340
Using the AuthenticationMethod Property . . . . .	341
Configuring User ID/Password Authentication . . . . .	343
Configuring Kerberos Authentication . . . . .	343
Specifying User Credentials for Kerberos Authentication (Delegation of Credentials) . . . . .	346
Obtaining a Kerberos Ticket Granting Ticket . . . . .	348
Configuring NTLM Authentication. . . . .	350
Configuring Client Authentication. . . . .	352
Data Encryption. . . . .	352
Reauthentication. . . . .	354
Client Information for Connections . . . . .	355
SQL Escape Sequences. . . . .	355
Isolation Levels. . . . .	355
Using Scrollable Cursors . . . . .	356
JTA Support . . . . .	356
Batch Inserts and Updates. . . . .	356

Parameter Metadata Support .....	357
Insert and Update Statements.....	357
Select Statements.....	358
Stored Procedures .....	359
ResultSet MetaData Support .....	359
Executing Insert/Update/Delete Statements with a RETURNING Clause .....	361
Rowset Support .....	364
Auto-Generated Keys Support .....	364
Configuring Failover .....	365
Specifying Primary and Alternate Servers.....	366
Specifying Connection Retry .....	369
Failover Properties .....	369
Bulk Load .....	371
<b>7 The Microsoft SQL Server Driver .....</b>	<b>373</b>
Data Source and Driver Classes .....	373
Connection URL .....	374
J2EE Connector Architecture Resource Adapter Class .....	375
Connection Properties .....	375
Performance Considerations .....	417
Connecting to Named Instances .....	420
Data Types.....	421
Returning and Inserting/Updating XML Data .....	424
Returning XML Data .....	424
Inserting/Updating XML Data .....	427
Authentication .....	428
Using the AuthenticationMethod Property .....	429
Configuring SQL Server Authentication .....	430
Configuring Kerberos Authentication .....	431

Specifying User Credentials for Kerberos Authentication (Delegation of Credentials) . . . . .	434
Obtaining a Kerberos Ticket Granting Ticket . . . . .	436
Configuring NTLM Authentication. . . . .	437
Data Encryption. . . . .	439
Using SSL with Microsoft SQL Server . . . . .	440
Configuring SSL Encryption. . . . .	441
DML with Results (Microsoft SQL Server 2005 and Higher) . . . . .	442
Reauthentication. . . . .	444
Client Information for Connections . . . . .	446
SQL Escape Sequences. . . . .	446
Isolation Levels. . . . .	447
Using the Snapshot Isolation Level (Microsoft SQL Server 2005 and Higher) . . . . .	447
Using Scrollable Cursors . . . . .	448
Server-Side Updatable Cursors . . . . .	448
JTA Support: Installing Stored Procedures. . . . .	449
Distributed Transaction Cleanup . . . . .	452
Transaction Timeout . . . . .	453
Explicit Transaction Cleanup . . . . .	454
Large Object (LOB) Support . . . . .	455
Batch Inserts and Updates. . . . .	455
Parameter Metadata Support. . . . .	456
Insert and Update Statements . . . . .	456
Select Statements. . . . .	456
Stored Procedures . . . . .	458
ResultSet MetaData Support . . . . .	458
Rowset Support . . . . .	460
Auto-Generated Keys Support . . . . .	460

Null Values .....	462
Configuring Failover .....	463
Specifying Primary and Alternate Servers.....	464
Specifying Connection Retry .....	467
Failover Properties .....	468
Bulk Load .....	470
<b>8 The Sybase Driver .....</b>	<b>471</b>
Data Source and Driver Classes .....	471
Connection URL .....	472
J2EE Connector Architecture Resource Adapter Class .....	472
Connection Properties .....	473
Performance Considerations .....	507
Data Types.....	510
Authentication .....	512
Using the AuthenticationMethod Property .....	513
Configuring User ID/Password Authentication.....	513
Configuring Kerberos Authentication .....	514
Specifying User Credentials for Kerberos .....	514
Authentication (Delegation of Credentials).....	517
Obtaining a Kerberos Ticket Granting Ticket.....	519
Data Encryption .....	520
Client Information for Connections .....	521
SQL Escape Sequences .....	521
Isolation Levels .....	522
Using Scrollable Cursors.....	522
Large Object (LOB) Support.....	522
Batch Inserts and Updates .....	523
Parameter Metadata Support .....	524

ResultSet MetaData Support .....	524
Rowset Support .....	526
Auto-Generated Keys Support .....	526
Null Values .....	527
Configuring Failover .....	529
Specifying Primary and Alternate Servers .....	530
Specifying Connection Retry.....	532
Failover Properties.....	533
Bulk Load .....	534
<b>A JDBC Support.....</b>	<b>535</b>
JDBC/JVM Compatibility .....	535
Supported Functionality .....	536
<b>B JDBC Extensions.....</b>	<b>601</b>
Using JDBC Wrapper Methods to Access the JDBC Extensions.....	602
DDBulkLoad Interface .....	603
ExtConnection Interface .....	613
ExtDatabaseMetaData Interface .....	619
ExtLogControl Class.....	620
<b>C Client Information for Connections .....</b>	<b>621</b>
How Databases Store Client Information .....	622
Storing Client Information .....	623
Returning Client Information.....	626
Returning MetaData About Client Information Locations .....	627

DB2 Workload Manager (WLM) Attributes . . . . .	628
DB2 V9.5 and V9.7 for Linux/UNIX/Windows . . . . .	628
DB2 for z/OS . . . . .	629
<b>D <code>getTypeInfo</code> . . . . .</b>	<b>631</b>
DB2 Driver . . . . .	631
Informix Driver . . . . .	642
MySQL Driver . . . . .	650
Oracle Driver . . . . .	662
SQL Server Driver . . . . .	670
Sybase Driver . . . . .	684
<b>E <b>Designing JDBC Applications for Performance Optimization</b> . . . . .</b>	<b>695</b>
Using Database Metadata Methods . . . . .	696
Minimizing the Use of Database Metadata Methods . . . . .	696
Avoiding Search Patterns . . . . .	697
Using a Dummy Query to Determine Table Characteristics . . . . .	698
Returning Data . . . . .	700
Returning Long Data . . . . .	700
Reducing the Size of Returned Data . . . . .	701
Choosing the Right Data Type . . . . .	702
Retrieving Result Sets . . . . .	702
Selecting JDBC Objects and Methods . . . . .	703
Using Parameter Markers as Arguments to Stored Procedures . . . . .	703
Using the Statement Object Instead of the PreparedStatement Object . . . . .	704
Using Batches Instead of Prepared Statements . . . . .	705
Choosing the Right Cursor . . . . .	707

Using get Methods Effectively . . . . .	708
Retrieving Auto-Generated Keys . . . . .	709
Managing Connections and Updates . . . . .	711
Managing Connections . . . . .	711
Managing Commits in Transactions . . . . .	713
Choosing the Right Transaction Model . . . . .	714
Using updateXXX Methods . . . . .	714
Using getBestRowIdentifier . . . . .	715
<b>F SQL Escape Sequences for JDBC . . . . .</b>	<b>717</b>
Date, Time, and Timestamp Escape Sequences . . . . .	718
Scalar Functions . . . . .	718
Outer Join Escape Sequences . . . . .	725
LIKE Escape Character Sequence for Wildcards . . . . .	727
Procedure Call Escape Sequences . . . . .	727
<b>G Using DataDirect Test™ . . . . .</b>	<b>729</b>
DataDirect Test™ Tutorial . . . . .	729
Configuring DataDirect Test™ . . . . .	730
Starting DataDirect Test™ . . . . .	731
Connecting Using DataDirect Test™ . . . . .	734
Executing a Simple Select Statement . . . . .	739
Executing a Prepared Statement . . . . .	741
Retrieving Database Metadata . . . . .	745
Scrolling Through a Result Set . . . . .	748
Batch Execution on a Prepared Statement . . . . .	751
Returning ParameterMetaData . . . . .	755
Establishing Savepoints . . . . .	757
Updatable Result Sets . . . . .	764
Large Object (LOB) Support . . . . .	776

<b>H Tracking JDBC Calls with DataDirect Spy™ . . . . .</b>	<b>783</b>
Enabling DataDirect Spy™ . . . . .	784
Using the JDBC Driver Manager . . . . .	784
Using JDBC Data Sources . . . . .	785
DataDirect Spy™ Attributes . . . . .	787
<b>I Connection Pool Manager . . . . .</b>	<b>789</b>
Configuring a Connection Pool . . . . .	789
Using Reauthentication with the Pool Manager . . . . .	790
Checking the Pool Manager Version . . . . .	793
Enabling Pool Manager Tracing . . . . .	793
Creating a Connection Pool . . . . .	794
Creating a Driver DataSource Object . . . . .	794
Creating the Connection Pool . . . . .	796
Connecting Using a Connection Pool . . . . .	799
Closing the Connection Pool . . . . .	802
DataDirect Connection Pool Manager Classes . . . . .	803
PooledConnectionDataSourceFactory . . . . .	803
PooledConnectionDataSource . . . . .	804
ConnectionPoolMonitor Class . . . . .	809
<b>J Statement Pool Monitor . . . . .</b>	<b>811</b>
Enabling Statement Pooling . . . . .	811
Accessing the Statement Pool Monitor . . . . .	812
Using JMX . . . . .	812
Using DataDirect-Specific Methods . . . . .	815
Importing Statements into a Statement Pool . . . . .	820
Clearing All Statements in a Statement Pool . . . . .	821
Freezing and Unfreezing the Statement Pool . . . . .	822
Generating a Statement Pool Export File . . . . .	823

DataDirect Statement Pool Monitor Classes .....	824
ExtStatementPoolMonitor Class .....	824
ExtStatementPoolMonitorMBean Interface .....	824
<b>K Using DataDirect Bulk Load .....</b>	<b>829</b>
Using a DDBulkLoad Object .....	830
Exporting Data to a CSV File .....	834
Loading Data .....	835
Logging .....	836
Using Bulk Load for Batch Inserts .....	837
Using CSV Files .....	838
Bulk Load Configuration File .....	838
Character Set Conversions .....	840
External Overflow Files .....	842
Discard File .....	843
<b>L Troubleshooting .....</b>	<b>845</b>
Troubleshooting Your Application .....	845
Turning On and Off DataDirect Spy™ Logging .....	845
DataDirect Spy™ Log Example .....	846
Troubleshooting Connection Pooling .....	851
Enabling Pool Manager Tracing .....	851
Pool Manager Trace File Example .....	851
Troubleshooting Statement Pooling .....	858
Generating a Statement Pool Export File .....	859
Statement Pool Export File Example .....	859
<b>Glossary .....</b>	<b>861</b>
<b>Index .....</b>	<b>865</b>



# Preface

This book is your guide to customizing and using Progress® DataDirect Connect® for JDBC™ from DataDirect Technologies.

---

## What Is DataDirect Connect® for JDBC?

DataDirect Connect for JDBC consists of Type 4 JDBC drivers that enable your applications to connect to a variety of databases from the Java™ platform running on operating systems that support Java. It supports advanced functionality such as distributed transactions, connection pooling, failover, authentication, and data encryption.

It also provides the following components:

- Performance Tuning Wizard for fine-tuning driver performance based on your application design and environment.
- DataDirect Test™ for JDBC for testing and debugging applications.
- DataDirect Spy™ for JDBC for tracing JDBC activity. This feature is built into the drivers.
- DataDirect Connection Pool Manager for creating your own connection pooling implementations.

---

# Using This Book

This book assumes that you are familiar with your operating system and its commands, the definition of directories, and accessing a database through an end-user application.

This book contains the following information:

- [Chapter 1 “Quick Start Connect” on page 29](#) provides information about connecting with and testing your DataDirect Connect *for JDBC* drivers.
- [Chapter 2 “Using DataDirect Connect® for JDBC Drivers” on page 35](#) provides information about using JDBC applications with the DataDirect Connect *for JDBC* drivers.
- Subsequent chapters for each database driver provide detailed information specific to the driver.
- [Appendix A “JDBC Support” on page 535](#) provides information about the JDBC interfaces and methods supported for DataDirect Connect *for JDBC*.
- [Appendix B “JDBC Extensions” on page 601](#) describes the JDBC extensions provided by the com.ddtek.jdbc.extensions package.
- [Appendix C “Client Information for Connections” on page 621](#) describes how you can use DataDirect Connect *for JDBC* to store and return client information for a connection.
- [Appendix D “getTypeInfo” on page 631](#) provides results returned from the DataBaseMetaData.getTypeinfo() method for all DataDirect Connect *for JDBC* drivers.
- [Appendix E “Designing JDBC Applications for Performance Optimization” on page 695](#) explains how you optimize your application code to improve performance.

- [Appendix F “SQL Escape Sequences for JDBC” on page 717](#) describes the scalar functions supported by the DataDirect Connect for JDBC drivers. Your data store may not support all these functions.
- [Appendix G “Using DataDirect Test™” on page 729](#) contains a tutorial that takes you through a step-by-step example of how to use DataDirect Test, a tool that allows you to test and debug your JDBC applications during development.
- [Appendix H “Tracking JDBC Calls with DataDirect Spy™” on page 783](#) describes how to use DataDirect Spy for tracking JDBC calls in running applications.
- [Appendix I “Connection Pool Manager” on page 789](#) describes how to use the DataDirect Connection Pool Manager to create your own connection pooling mechanism.
- [Appendix J “Statement Pool Monitor” on page 811](#) describes how to use the DataDirect Statement Pool Monitor to import statements to and remove statements from the statement pool as well as generate information to help you troubleshoot statement pooling performance.
- [Appendix K “Using DataDirect Bulk Load” on page 829](#) describes how to use DataDirect Bulk Load, a feature that allows your application to send large numbers of rows of data to the database.
- [Appendix L “Troubleshooting” on page 845](#) provides information that can help you troubleshoot driver problems.

In addition, a [“Glossary” on page 861](#) helps you with terminology referenced in this book.

NOTE: This book refers the reader to Web pages using URLs for more information about specific topics, including Web URLs not maintained by DataDirect Technologies. Because it is the nature of Web content to change frequently, DataDirect Technologies can guarantee only that the URLs referenced in this book were correct at the time of publishing.

# About the Product Documentation

The DataDirect Connect *for JDBC* library consists of the following books:

- *DataDirect Connect for JDBC Installation Guide* details requirements and procedures for installing DataDirect Connect *for JDBC*.
- *DataDirect Connect for JDBC User's Guide and Reference* provides information about customizing and using DataDirect Connect *for JDBC* drivers.

## HTML Version

The *DataDirect Connect for JDBC User's Guide and Reference* is placed on your system as HTML online help during a normal installation of the product. The help system is located in the help subdirectory of the product installation directory. To use the help, you must have one of the following browsers installed:

- Internet Explorer 5.x, 6.x, and 7.x
- Mozilla Firefox 1.x, 2.x, and 3.x
- Netscape 4x, 7.x and 8.x
- Safari 1.x, 2.x, and 3x
- Opera 7.54u2, 8.x, and 9x

You can access the help system by navigating to the help subdirectory of the product installation directory and opening the following file from within your browser:

`install_dir/help/help.htm`

where `install_dir` is the path to your product installation directory.

After the browser opens, the left pane displays the Table of Contents, Index, and Search tabs for the entire documentation library. When you open the main screen of the help system in your browser, you can bookmark it in the browser for quick access later.

NOTE: Security features set in your browser can prevent the Help system from launching. A security warning message is displayed. Often, the warning message provides instructions for unblocking the Help system for the current session. To allow the Help system to launch without encountering a security warning message, the security settings in your browser can be modified. Check with your system administrator before disabling any security features.

## PDF Version

DataDirect product documentation is also provided in PDF format, which allows you to view it, perform text searches, and print it. You can view the PDF documentation using Adobe Acrobat Reader. The PDF documentation is available on the product DVD and also on the DataDirect Technologies Web site:

<http://www.datadirect.com/techres/jdbcproddoc/index.ssp>

You can download the entire library as a compressed file. When you uncompress the file, the library will appear in the correct directory structure.

If you copy the documentation library from the product DVD, you must maintain the same directory structure that is on the DVD.

- **To copy all product books**, copy the entire \books directory to your local or network drive.
- **To copy a specific set of books**, copy that book's directory structure (beneath the \books directory) to your local or network drive. For example, in the case of:

\books\product\_name

you would copy the entire \product\_name directory.

Maintaining the correct directory structure allows cross-book text searches and cross-references. If you download or copy the books individually outside of their normal directory structure, their cross-book search indexes and hyperlinked cross-references to other books will not work. You can view a book individually, but it will not automatically open other books to which it has cross-references.

To help you navigate through the library, a file named books.pdf is provided. This file lists each online book provided for the product. We recommend that you open this file first and, from this file, open the book you want to view.

---

# Typographical Conventions

This book uses the following typographical conventions:

Convention	Explanation
<i>italics</i>	Introduces new terms with which you may not be familiar, and is used occasionally for emphasis.
<b>bold</b>	Emphasizes important information. Also indicates button, menu, and icon names on which you can act. For example, click <b>Next</b> .
UPPERCASE	Indicates keys or key combinations that you can use. For example, press the ENTER key.
	Also used for SQL reserved words.
monospace	Indicates syntax examples, values that you specify, or results that you receive.
<i>monospaced</i>	Indicates names that are placeholders for values that you specify. For example, <i>filename</i> .
<i>italics</i>	
forward slash /	Separates menus and their associated commands. For example, Select File / Copy means that you should select Copy from the File menu.
	The slash also separates directory levels when specifying locations under UNIX.
vertical rule	Indicates an "OR" separator used to delineate items.
brackets [ ]	Indicates optional items. For example, in the following statement: SELECT [DISTINCT], DISTINCT is an optional keyword.
	Also indicates sections of the Windows Registry.

Convention	Explanation
braces { }	Indicates that you must select one item. For example, {yes   no} means that you must specify either yes or no.
ellipsis . . .	Indicates that the immediately preceding item can be repeated any number of times in succession. An ellipsis following a closing bracket indicates that all information in that unit can be repeated.

---

## Contacting Technical Support

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

<http://support.datadirect.com>

The DataDirect Technologies 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.

To obtain technical support for an evaluation copy of the product, go to:

[http://www.datadirect.com/support/eval\\_help/index.ssp](http://www.datadirect.com/support/eval_help/index.ssp)

or contact your sales representative.

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

- 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 customer information, including location.
- The DataDirect product and the version that you are using.
- The type and version of the operating system where you have installed your DataDirect 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 recreated.
- 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.



# 1 Quick Start Connect

This chapter provides basic information for connecting, testing, and using your DataDirect Connect *for JDBC* drivers immediately after installation. To take full advantage of the many driver features, we recommend that you read [Chapter 2 “Using DataDirect Connect® for JDBC Drivers”](#) and the chapters specific to the drivers you are using.

---

## Connecting to a Database

Once DataDirect Connect *for JDBC* is installed and the drivers are configured the way you want them, you can connect from your application to your database in either of the following ways: with a connection URL through the JDBC Driver Manager or with a Java Naming Directory Interface (JNDI) data source. This quick start explains how to test your database connection using a connection URL. See [Chapter 2 “Using DataDirect Connect® for JDBC Drivers”](#) for information about using data sources.

You can connect through the JDBC Driver Manager with the `DriverManager.getConnection` method, which uses a string containing a connection URL.

Use the following steps to load the drivers from your JDBC application.

### 1. Setting the Classpath

The DataDirect Connect *for JDBC* drivers must be defined in your CLASSPATH variable. The CLASSPATH is the search string your

Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. If the drivers are not defined on your CLASSPATH, you will receive a "class not found" error when trying to load the driver. Set your system CLASSPATH to include the driver jar file as shown, where *driver.jar* is the driver jar file (for example, *sqlserver.jar*) and *install\_dir* is the path to your DataDirect Connect for JDBC installation directory:

```
install_dir/lib/driver.jar
```

### Windows Example

```
CLASSPATH=. ;c:\connect\jdbc\lib\sqlserver.jar
```

### UNIX Example

```
CLASSPATH=. :/home/user1/connect/jdbc/lib/sqlserver.jar
```

## 2. Registering the Driver

**IMPORTANT:** If using Java SE 6, you do not need to register the driver and can skip this step. Java SE 6 automatically registers the driver with the JDBC Driver Manager.

Registering the driver tells the JDBC Driver Manager which driver to load. The driver is registered by using the `Class.forName` method and specifying the driver class name as the argument. The names of the DataDirect Connect for JDBC drivers are:

- `com.ddtek.jdbc.db2.DB2Driver`
- `com.ddtek.jdbc.informix.InformixDriver`
- `com.ddtek.jdbc.mysql.MySQLDriver`
- `com.ddtek.jdbc.oracle.OracleDriver`
- `com.ddtek.jdbc.sqlserver.SQLServerDriver`
- `com.ddtek.jdbc.sybase.SybaseDriver`

For example:

```
Class.forName("com.ddtek.jdbc.sqlserver.SQLServerDriver");
```

## 3. Passing the Connection URL

After registering the driver, pass your database connection information in the form of a connection URL. Use the following URL formats as templates to create your own connection URLs, substituting the appropriate values specific to your database.

### DB2 for Linux, UNIX, and Windows<sup>1</sup>

```
jdbc:datadirect:db2://server_name:50000;DatabaseName=your_database
```

### DB2 for z/OS and iSeries<sup>1</sup>

```
jdbc:datadirect:db2://server_name:446;LocationName=db2_location
```

### Informix

```
jdbc:datadirect:informix://server_name:2003;InformixServer=your_server;  
DatabaseName=your_database
```

### MySQL

```
jdbc:datadirect:mysql://server_name:3306
```

### Oracle<sup>2</sup>

```
jdbc:datadirect:oracle://server_name:1521;ServiceName=your_servicename
```

### Microsoft SQL Server<sup>3</sup>

```
jdbc:datadirect:sqlserver://server_name:1433
```

### Sybase

```
jdbc:datadirect:sybase://server_name:5000
```

---

1. See “[DB2 Packages](#)” on page 134 for more information about creating DB2 packages.

2. See “[Using tnsnames.ora Files](#)” on page 319 for instructions on retrieving connection information from an Oracle tnsnames.ora file.

3. See “[Connecting to Named Instances](#)” on page 420 for instructions on connecting to named instances.

For example, the following connection URL establishes a connection to a Microsoft SQL Server database and includes user ID and password information for the connection:

```
Connection conn = DriverManager.getConnection  
("jdbc:datadirect:sqlserver://server1:1433;User=test;Password=secret");
```

#### NOTES:

- *server\_name* is an IP address or a host name, if your network resolves host names to IP addresses. To test this feature, use the ping command to access the host name. For example:

```
ping server1
```

Then, verify that the correct IP address was returned in the reply from the command.

- The integer value after the server name is the port number on which the database is listening.

See the appropriate driver chapter for a complete list of connection properties for each driver.

---

## Testing the Connection

You can use DataDirect Test to test your connections. DataDirect Test is a pure Java™ software component developed by DataDirect Technologies to test JDBC applications. It can also be used as a tool to help developers learn the JDBC API.

The DataDirect Test menu items correspond to specific JDBC functions (for example, connecting to a database or passing a SQL statement) or encapsulate multiple JDBC function calls as a shortcut to perform common tasks (for example, displaying the contents of a result set). DataDirect Test displays the results of all JDBC function calls and provides fully commented sample JDBC code in the same window.

See the tutorial in [Appendix G “Using DataDirect Test™” on page 729](#) for information about testing your connection using DataDirect Test.

---

## Using the Performance Tuning Wizard

The Performance Tuning Wizard leads you step-by-step through a series of questions about your application. Based on your answers, the Wizard provides the optimal settings for performance-related connection properties.

The Wizard runs as an applet within a browser window. To use the Wizard, your browser must be configured to run applets. Refer to your browser’s documentation for information on configuring your browser.

### Starting the Wizard

You can start the Wizard by launching the following file from your browser window, where *install\_dir* is your DataDirect Connect for JDBC installation directory:

*install\_dir/wizards/index.html*

NOTE: Security features set in your browser can prevent the Performance Wizard from launching. A security warning message is displayed. Often, the warning message provides instructions for unblocking the Performance Wizard for the current session. To allow the Performance Wizard to launch without encountering a security warning message, the security settings in your browser can be modified. Check with your system administrator before disabling any security features.

## Tuning Performance Using the Wizard

After you start the Wizard, a Welcome window appears. Click **Start** to start the process and select a driver. When you have answered all questions for a driver, the Wizard displays the optimal settings for connection properties that affect performance. You can copy the settings to use in your application or in a data source configuration tool. In addition, you can copy and paste the settings into a text file for reference.

See the appropriate driver chapter for information about configuring connection properties for the drivers.

## 2 Using DataDirect Connect® for JDBC Drivers

DataDirect Connect *for JDBC* Type 4 drivers provide JDBC access through any Java-enabled applet, application, or application server. They deliver high-performance point-to-point and *n*-tier access to industry-leading data stores across the Internet and intranets. Our drivers are optimized for the Java environment, allowing you to incorporate Java technology and extend the functionality and performance of your existing system.

---

## About the Product

In addition to DataDirect Connect *for JDBC* drivers, the following components are shipped with DataDirect Connect *for JDBC*:

- DataDirect Test
- DataDirect Spy<sup>1</sup>
- DataDirect Connection Pool Manager
- J2EE Connector Architecture resource adapters

---

1.This feature is built into the drivers.

## The Drivers

DataDirect Connect *for JDBC* drivers are compliant with JDBC 4.0 and earlier specifications.

NOTE: JDBC 4.0 functionality requires Java SE 6.

The DataDirect Connect *for JDBC* drivers provide connection properties that allow you to customize each driver for your environment. See “[Specifying Connection Properties](#)” on page 43 for information about using driver connection properties.

## DataDirect Test™

DataDirect Test is a menu-driven software component that helps you debug your JDBC applications and learn how to use the DataDirect Connect *for JDBC* drivers. DataDirect Test contains menu items that:

- Correspond to specific JDBC functions—for example, connecting to a database or passing a SQL statement.
- Encapsulate multiple JDBC function calls as a shortcut to perform some common tasks, such as displaying the contents of a result set.

DataDirect Test displays the results of all JDBC function calls in one window, while displaying fully commented, Java JDBC code in an alternate window. DataDirect Test works with all DataDirect Connect *for JDBC* drivers.

See [Appendix G “Using DataDirect Test™”](#) on page 729 for more information.

## DataDirect Spy™

DataDirect Spy is a feature for tracking and logging detailed information about JDBC calls at runtime. DataDirect Spy provides the following advantages:

- Logging is JDBC 4.0-compliant.
- Logging is consistent, regardless of the JDBC driver used.
- All parameters and function results for JDBC calls can be logged.
- Logging works with all DataDirect Connect for JDBC drivers.
- Logging can be enabled without changing the application.

See [Appendix H “Tracking JDBC Calls with DataDirect Spy™” on page 783](#) for more information.

## DataDirect Connection Pool Manager

Database access performance can be improved significantly when connection pooling is used. *Connection pooling* means that connections are reused rather than created each time a connection is requested.

Your applications can use connection pooling through the DataDirect Connection Pool Manager. See [Appendix I “Connection Pool Manager” on page 789](#) for more information.

## J2EE Connector Architecture Resource Adapters

The J2EE Connector Architecture defines a standard structure for connecting the J2EE platform to Enterprise Information Systems (EIS). Examples of EIS include mainframe transaction processing,

database systems, and legacy applications that are not written in the Java programming language. The J2EE Connector Architecture allows you to integrate EIS with application servers and enterprise applications. The J2EE Connector Architecture defines a standard set of system-level contracts between an application server and the EIS to ensure compatibility between them. The resource adapter implements the EIS part of these system-level contracts.

The J2EE Connector Architecture also defines a standard Service Provider Interface (SPI) for integrating the transaction, security, and connection management facilities of an application server with those of a transactional resource manager. The JDBC specification provides more information about the relationship of JDBC to the SPI specified in the J2EE Connector Architecture.

DataDirect Connect *for JDBC* supports JDBC functionality through the J2EE Connector Architecture SPI through resource adapters. A *resource adapter* is a system-level software driver used by an application server to connect to an EIS. The resource adapter communicates with the server to provide the underlying transaction, security, and connection pooling mechanisms.

DataDirect Connect *for JDBC* resource adapters conform to the J2EE Connector Architecture 1.0 specification. The resource adapters are provided in resource archive (RAR) files (see [Table 2-1](#)).

---

**Table 2-1. DataDirect Connect for JDBC Resource Adapters**

---

<b>DataDirect Connect <i>for JDBC</i> Driver</b>	<b>RAR File</b>
DataDirect Connect <i>for JDBC</i> DB2 driver	db2.rar
DataDirect Connect <i>for JDBC</i> Informix driver	informix.rar
DataDirect Connect <i>for JDBC</i> MySQL driver	mysql.rar
DataDirect Connect <i>for JDBC</i> Oracle driver	oracle.rar

---

**Table 2-1. DataDirect Connect for JDBC Resource Adapters (cont.)**

---

DataDirect Connect for JDBC Driver	RAR File
DataDirect Connect for JDBC SQL Server driver	sqlserver.rar
DataDirect Connect for JDBC Sybase driver	sybase.rar

---

## ***Using Resource Adapters with an Application Server***

DataDirect Connect for JDBC resource adapters can be used with any J2EE 1.3 or higher application server. With J2EE 1.4 and higher application servers, they must be used in conjunction with the Sun Microsystems JDBC Connector. Refer to the Sun Microsystems JDBC Connector documentation for more information or go to the following Web site:

<http://java.sun.com/developer/earlyAccess/jdbc/index.html>

In an application server environment, resource adapters are deployed using a deployment tool. Typically, when deploying a resource adapter, you place the resource archive file (RAR file) in the application server's /lib directory. Each RAR file includes a *deployment descriptor*, which instructs the application server on how to use the resource adapter in an application server environment. The deployment descriptor contains information about the resource adapter, including security and transactional capabilities, as well as the ManagedConnectionFactory class name.

Refer to your application server documentation for details about deploying components using the deployment tool. See “[Security Permissions for Resource Adapters](#)” on page 42 for more information about security and resource adapters.

For an example that shows how to set up the DataDirect Connect for JDBC resource adapters with a J2EE application server, refer to the document named connectorsample.pdf in the /examples/connector subdirectory of your DataDirect Connect for JDBC installation directory.

## ***Using Resource Adapters from an Application***

Your application also can use resource adapters directly instead of using them through a container-managed, application server environment. The following code example shows how your application can access an Oracle database directly using a resource adapter:

```
import java.util.Hashtable;
import java.sql.Connection;
import javax.sql.DataSource;
import javax.naming.*;

import com.ddtek.resource.jdbc.JCAConnectionFactory;
import com.ddtek.resource.jdbc.spi.*;

public class RAExample {

    static public void main(String[] args) {
        try {

            // Create a connection factory instance
            OracleManagedConnectionFactory managedFactory =
                new OracleManagedConnectionFactory();

            managedFactory.setServerName( "MyOracleServer" );
            managedFactory.setPortNumber( "1521" );
            managedFactory.setSID( "TEST" );

            JCAConnectionFactory factory = (JCAConnectionFactory)
                managedFactory.createConnectionFactory();
        }
    }
}
```

```
// Get an InitialContext.  
// Using File System JNDI Service Provider as an example  
Hashtable env = new Hashtable();  
  
env.put(Context.INITIAL_CONTEXT_FACTORY,  
"com.sun.jndi.fscontext.RefFSContextFactory");  
env.put(Context.PROVIDER_URL, "file:c:/ConnectionFactories");  
  
Context connectorContext = new InitialContext(env);  
  
// Bind the connection factory  
try {  
    connectorContext.bind("OracleConnectionFactory", factory);  
}  
catch (NameAlreadyBoundException except) {  
    connectorContext.rebind("OracleConnectionFactory", factory);  
}  
}  
catch (Exception except) {  
    System.out.println("Error creating DataSource");  
    System.exit(0);  
}  
  
// Connect via the DataSource  
try {  
  
    // Get an InitialContext.  
    // Using File System JNDI Service Provider as an example  
    Hashtable env = new Hashtable();  
  
    env.put(Context.INITIAL_CONTEXT_FACTORY,  
            "com.sun.jndi.fscontext.RefFSContextFactory");  
    env.put(Context.PROVIDER_URL, "file:c:/ConnectionFactories");  
  
    Context connectorContext = new InitialContext(env);  
  
    // Lookup the connection factory  
    DataSource dataSource = (DataSource)  
    connectorContext.lookup("OracleConnectionFactory");  
    Connection connection = dataSource.getConnection("scott", "tiger");  
}
```

```
        catch (Exception except) {
            System.out.println("Looking up connection factory");
        }
    }
}
```

## ***Security Permissions for Resource Adapters***

Each DataDirect Connect *for JDBC* RAR file includes a deployment descriptor, which contains information about the resource adapter, including a list of permissions that must be granted to the resource adapters when they run in the presence of a Security Manager. The Security Manager is a class defined by the Java Security Architecture (part of the J2SE API).

Typically, when a Java program is run from the command line, the program is run without a Security Manager, meaning that the application has full access to the resources on the computer on which it is running. Optionally, you can specify a Security Manager using a command-line argument to the JVM. If a Security Manager is specified, the JVM checks with the Security Manager prior to allowing application access to system resources. If the application has permission to access the resource, the operation is allowed to proceed; otherwise, an exception is generated indicating that the application does not have the required permissions to access the resource.

Permissions are granted to an application using a set of policy files. Typically, Security Managers are only used when running within an application server or when running an applet, but you can specify that a Security Manager be used when running any Java application.

The deployment descriptors for the DataDirect Connect *for JDBC* resource adapters contain only one permission, the `modifyThreadGroup` permission. This permission must be granted to the resource adapter; it is required to create new threads.

Check with your system administrator to make sure that the resource adapter is granted the modifyThreadGroup permission.

For more information about the Java Security Architecture, refer to the following document on the Sun Microsystems Web site:

<http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc.html>.

---

## Specifying Connection Properties

DataDirect Connect *for JDBC* provides many connection options that allow you to customize the drivers for your environment. You can specify connection properties using either of the following methods:

- JDBC Driver Manager. See “[Connecting Through the JDBC Driver Manager](#)” on page 44 for more information.
- JDBC data sources. See “[Connecting Through Data Sources](#)” on page 48 for information about example code that you can use as a template for creating and using your own data sources.

You can use connection options to accomplish a number of different tasks, such as provide specific details for a connection or optimize performance in certain scenarios.

## Providing Connection Information

DataDirect Connect *for JDBC* provides connection options that allow you to specify details for a connection. For example, you may want to enable failover and client load balancing to make sure that your data is always available, even when server failures

occur. See “[Using Failover](#)” on page 58 and “[Using Client Load Balancing](#)” on page 65 for details about these features.

## Optimizing Performance

DataDirect Connect for JDBC provides connection properties that allow you to optimize performance for your application in certain scenarios. For example, if your application accesses an Oracle database and can accept not receiving update count information when executing batches, you may want to set the BatchPerformanceWorkaround connection property to true. When set to true, the native Oracle batch mechanism is used, which can significantly improve the performance of batch inserts and updates. This mechanism does not return individual update counts for each statement or parameter in the batch.

The Performance Tuning Wizard helps you determine the optimal settings for performance-related connection properties. See “[Using the Performance Tuning Wizard](#)” on page 33 for more information about using the Wizard.

---

## Connecting Through the JDBC Driver Manager

One way to connect to a database is through the JDBC Driver Manager by using the `DriverManager.getConnection` method. This method uses a string containing a connection URL. The following code fragment shows an example of using the JDBC Driver Manager to connect to Microsoft SQL Server.

```
Connection conn = DriverManager.getConnection  
("jdbc:datadirect:sqlserver://server1:1433;User=test;Password=secret");
```

## Registering the Drivers

**IMPORTANT:** If using Java SE 6, you do not need to register the drivers and can skip this step. Java SE 6 automatically registers the drivers with the JDBC Driver Manager.

Registering the DataDirect Connect *for* JDBC drivers with the JDBC Driver Manager allows the JDBC Driver Manager to load them. The names of the DataDirect Connect *for* JDBC drivers are:

- com.ddtek.jdbc.db2.DB2Driver
- com.ddtek.jdbc.informix.InformixDriver
- com.ddtek.jdbc.mysql.MySQLDriver
- com.ddtek.jdbc.oracle.OracleDriver
- com.ddtek.jdbc.sqlserver.SQLServerDriver
- com.ddtek.jdbc.sybase.SybaseDriver

You can register the DataDirect Connect *for* JDBC drivers with the JDBC Driver Manager using any of the following methods:

- **Method 1:** Set the Java system property `jdbc.drivers` using the Java -D option. The `jdbc.drivers` property is defined as a colon-separated list of driver class names. This example registers the DataDirect Connect *for* JDBC DB2 driver and the DataDirect Connect *for* JDBC SQL Server driver.

```
java -Djdbc.drivers=com.ddtek.jdbc.db2.DB2Driver:  
com.ddtek.jdbc.sqlserver.SQLServerDriver
```

- **Method 2:** Set the Java property `jdbc.drivers` from within your Java application or applet. Include the following code fragment in your Java application or applet, and call `DriverManager.getConnection`. This example registers the DataDirect Connect *for* JDBC SQL Server driver.

```
Properties p = System.getProperties();  
p.put ("jdbc.drivers",  
"com.ddtek.jdbc.sqlserver.SQLServerDriver");  
System.setProperties (p);
```

- **Method 3:** Explicitly load the driver class using the standard Class.forName method. Include the following code fragment in your application or applet and call DriverManager.getConnection. This example registers the DataDirect Connect for JDBC DB2 driver.

```
Class.forName( "com.ddtek.jdbc.db2.DB2Driver" );
```

## Specifying Connection URLs

The connection URL format used with the Driver Manager is:

`jdbc:datadirect:drivername://hostname:port[;property=value[;...]]`

where:

*drivername* is the name of the DataDirect Connect for JDBC driver, for example, `sybase`.

*hostname* is the IP address or TCP/IP host name of the server to which you are connecting. See “[Using IP Addresses](#)” on page 50 for details on using IP addresses.

NOTE: Untrusted applets cannot open a socket to a machine other than the originating host.

*port* is the number of the TCP/IP port.

*property=value* specifies connection properties. For a list of connection properties supported for each DataDirect Connect for JDBC driver and their valid values, see the appropriate driver chapter.

The following examples show some typical DataDirect Connect for JDBC driver connection URLs:

### DB2 for Linux, UNIX, and Windows<sup>1</sup>

```
jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc;User=test;  
Password=secret
```

### DB2 for z/OS and iSeries<sup>1</sup>

```
jdbc:datadirect:db2://server1:446;LocationName=Sample;User=test;  
Password=secret
```

### Informix

```
jdbc:datadirect:informix://server4:1526;InformixServer=ol_test;  
DatabaseName=your_database;User=test;Password=secret
```

### MySQL

```
jdbc:datadirect:mysql://server1:3306;User=test;Password=secret
```

### Oracle<sup>2</sup>

```
jdbc:datadirect:oracle://server3:1521;ServiceName=ORCL;User=test;  
Password=secret
```

### Microsoft SQL Server<sup>3</sup>

```
jdbc:datadirect:sqlserver://server1:1433;User=test;Password=secret
```

### Sybase

```
jdbc:datadirect:sybase://server2:5000;User=test;Password=secret
```

- 
- 1.See “[DB2 Packages](#)” on page 134 for more information about creating DB2 packages.
  - 2.See “[Using tnsnames.ora Files](#)” on page 319 for instructions on retrieving connection information from an Oracle tnsnames.ora file.
  - 3.See “[Connecting to Named Instances](#)” on page 420 for instructions on connecting to named instances.

# Connecting Through Data Sources

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

The main advantage of using a data source is that it works with the Java Naming Directory Interface (JNDI) naming service, and it is created and managed apart from applications that use it. Because the connection information is defined outside of the application, the effort to reconfigure your infrastructure when a change is made is minimal. For example, if the database is moved to another database server and uses another port number, the administrator needs only to change the relevant properties of the data source (`DataSource` object). The applications using the database do not need to change because they only refer to the logical name of the data source.

## How Data Sources Are Implemented

DataDirect Technologies ships a data source class for each DataDirect Connect *for JDBC* driver. See the appropriate driver chapter for the name of the data source class associated with each driver.

Each DataDirect Connect *for JDBC* data source class implements the following JDBC interfaces:

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

- javax.sql.XADataSource allows applications to use distributed transactions through the Java Transaction API (JTA).

## Creating Data Sources

Your DataDirect Connect *for JDBC* installation contains the following examples that show how to create and use DataDirect Connect *for JDBC* data sources:

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

NOTE: You must include the javax.sql.\* and javax.naming.\* classes to create and use DataDirect Connect *for JDBC* data sources. The DataDirect Connect *for JDBC* drivers provide all the necessary JAR files, which contain the required classes and interfaces.

## Calling a Data Source in an Application

Applications can call a DataDirect Connect *for JDBC* 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 example:

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

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

See “[Creating Data Sources](#)” on page 49 for information about example data sources shipped with DataDirect Connect *for JDBC* that you can use as a template for creating your own data sources.

---

## Using IP Addresses

The DataDirect Connect *for JDBC* drivers support Internet Protocol (IP) addresses in IPv4 and IPv6 format. IPv6 addresses are only supported when connecting to certain database versions (as shown in [Table 2-2](#)). In addition, to connect to IPv6 addresses, the driver machine requires J2SE 5.0 or higher on Windows and J2SE 1.4 or higher on UNIX/Linux.

---

**Table 2-2. IP Address Formats Supported by the DataDirect Connect for JDBC Drivers**

---

Driver	IPv4	IPv6
DB2	All supported versions	DB2 V9.1 and higher for Linux/UNIX/Windows DB2 v9.1 for z/OS DB2 V5R2 and higher for iSeries
Informix	All supported versions	Informix 10 and higher
MySQL	All supported versions	Not supported
Oracle	All supported versions	Not supported

---

**Table 2-2. IP Address Formats Supported by the DataDirect Connect for JDBC Drivers (cont.)**

---

Driver	IPv4	IPv6
Microsoft SQL Server	All supported versions	Microsoft SQL Server 2005 and higher
Sybase	All supported versions	Sybase 12.5.2 and higher

---

If your network supports named servers, the server name specified in the connection URL or data source can resolve to an IPv4 or IPv6 address. For example, the server name DB2Server in the following URL can resolve to either type of address:

```
jdbc:datadirect:db2://DB2Server:50000;DatabaseName=jdbc;User=test;
Password=secret
```

Alternatively, you can specify addresses using IPv4 or IPv6 format in the server name portion of the connection URL. For example, the following connection URL specifies the server using IPv4 format:

```
jdbc:datadirect:db2://123.456.78.90:50000;DatabaseName=jdbc;User=test;
Password=secret
```

You also can specify addresses in either format using the `ServerName` data source property. The following example shows a data source definition that specifies the server name using IPv6 format:

```
DB2DataSource mds = new DB2DataSource();
mds.setDescription("My DB2DataSource");
mds.setServerName("[ABCD:EF01:2345:6789:ABCD:EF01:2345:6789]");
mds.setPortNumber(50000);
...
```

**NOTE:** When specifying IPV6 addresses in a connection URL or data source property, the address must be enclosed by brackets.

In addition to the normal IPv6 format, the DataDirect Connect for JDBC drivers support IPv6 alternative formats for compressed and IPv4/IPv6 combination addresses. For example, the following connection URL specifies the server using IPv6 format, but uses the compressed syntax for strings of zero bits:

```
jdbc:datadirect:db2://[2001:DB8:0:0:8:800:200C:417A]:50000;DatabaseName=jdbc;  
User=test;Password=secret
```

Similarly, the following connection URL specifies the server using a combination of IPv4 and IPv6:

```
jdbc:datadirect:db2://[0000:0000:0000:0000:0000:FFFF:123.456.78.90]:50000;  
DatabaseName=jdbc;User=test;Password=secret
```

For complete information about IPv6, go to the following URL:

<http://tools.ietf.org/html/rfc4291#section-2.2>

---

## Using Connection Pooling

Typically, creating a connection is the most performance-expensive operation that an application performs. Connection pooling allows you to reuse connections rather than create a new one every time a DataDirect Connect for JDBC driver needs to establish a connection to the database. Connection pooling manages connection sharing across different user requests to maintain performance and reduce the number of new connections that must be created. For example, compare the following transaction sequences.

### Example A: Without Connection Pooling

- 1 The client application creates a connection.
- 2 The client application sends a data access query.
- 3 The client application obtains the result set of the query.

- 4 The client application displays the result set to the end user.
- 5 The client application ends the connection.

#### **Example B: With Connection Pooling**

- 1 The client application requests a connection from the connection pool.
- 2 If an unused connection exists, it is returned by the pool; otherwise, the pool creates a new connection.
- 3 The client application sends a data access query.
- 4 The client application obtains the result set of the query.
- 5 The client application displays the result set to the end user.
- 6 The client application closes the connection, which returns the connection to the pool.

NOTE: The client application calls the close() method, but the connection remains open and the pool is notified of the close request.

## **Understanding Connection Pooling**

The connection pool implementation (the DataDirect Connection Pool Manager in the case of DataDirect Connect for JDBC) creates "real" database connections, referred to as PooledConnections, using the getPooledConnection() method of the ConnectionPoolDataSource interface. Then, the Pool Manager registers itself as a listener to the PooledConnection. When a client application requests a connection, the Pool Manager assigns an available connection. If a connection is unavailable, the Pool Manager establishes a new connection and assigns it to that application.

When the client application closes the connection, the Pool Manager is notified by the driver through the

ConnectionEventListener interface that the connection is free and available for reuse. The Pool Manager is also notified by the ConnectionEventListener interface if the client somehow corrupts the database connection so the Pool Manager can remove that connection from the pool.

Connection pool implementations, such as the DataDirect Connection Pool Manager, use objects that implement the javax.sql.ConnectionPoolDataSource interface to create the physical connections managed in the pool. All DataDirect Connect *for JDBC* DataSource objects implement the ConnectionPoolDataSource interface.

Once a data source has been created and registered with JNDI, it can be used by your JDBC application as shown in the following example, typically through a third-party connection pool tool:

```
Context ctx = new InitialContext();
ConnectionPoolDataSource ds =
(ConnectionPoolDataSource)ctx.lookup("EmployeeDB");
Connection conn = ds.getConnection("scott", "tiger");
```

In this example, first, the JNDI environment is initialized. Next, the initial naming context is used to find the logical name of the JDBC data source (EmployeeDB). The Context.lookup method returns a reference to a Java object, which is narrowed to a javax.sql.ConnectionPoolDataSource object. Next, the ConnectionPoolDataSource.getPooledConnection() method is called to establish a connection with the underlying database. Finally, the application obtains a connection from the ConnectionPoolDataSource.

The DataDirect Connection Pool Manager is shipped with DataDirect Connect *for JDBC*. See [Appendix I “Connection Pool Manager” on page 789](#) for more information about managing connection pools with the DataDirect Connection Pool Manager.

## Using Reauthentication

Typically, you can configure a connection pool to provide scalability for connections. In addition, to help minimize the number of connections required in a connection pool, you can switch the user associated with a connection to another user, a process known as *reauthentication*.

For example, suppose you are using Kerberos authentication to authenticate users using their operating system user name and password. To reduce the number of connections that must be created and managed, you can use reauthentication to switch the user associated with a connection to multiple users. For example, suppose your connection pool contains a connection, Conn, which was established by the user ALLUSERS. That connection can service multiple users (User A, B, and C) by switching the user associated with the connection Conn to User A, B, and C.

Not all databases support reauthentication. For those that do, the user performing the switch must have been granted specific database permissions.

[Table 2-3](#) shows DataDirect Connect for JDBC support for reauthentication and lists the required database permissions. Refer to your database documentation for information about granting permissions.

---

**Table 2-3. DataDirect Connect for JDBC Support for Reauthentication**

---

Driver	Database	Required Database Permissions
DB2 driver	DB2 V9.1 and higher for Linux/UNIX/Windows	SETSESSIONUSER
	DB2 v8.1.4 and higher for Linux/UNIX/Windows	SYSADM

**Table 2-3. DataDirect Connect for JDBC Support for Reauthentication (cont.)**

<b>Driver</b>	<b>Database</b>	<b>Required Database Permissions</b>
Oracle driver	Oracle 8.1.6 and higher <sup>1</sup>	CONNECT THROUGH
SQL Server driver	Microsoft SQL Server 2005 and higher <sup>2</sup>	IMPERSONATE

1. Oracle refers to reauthentication as *proxy authentication*.  
 2. Microsoft SQL Server refers to reauthentication as *impersonation*.

**NOTE:** Before performing reauthentication, applications must ensure that any statements or result sets created as one user are closed before switching the connection to another user.

To perform reauthentication on a connection explicitly, call the `setCurrentUser()` method in the `ExtConnection` interface located in the `com.ddtek.jdbc.extensions` package. For example, the following code switches the user on the connection from the user that created the connection (TEST) to a new user (SMITH). In addition, it sets options that switch the current schema and current database path to the schema and database path of the new user.

```
import com.ddtek.jdbc.extensions.*
// Get Database Connection
Connection con = DriverManager.getConnection
  ("jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc", "TEST", "secret");
ExtConnection extCon = (ExtConnection)con
Properties props = new Properties();

props.put("CURRENT_SCHEMA", "SCHEMA");
props.put("CURRENT_PATH", "PATH");
ExtCon.setCurrentUser("SMITH", props);
...
```

See the individual driver chapters for information on which options are supported for each driver.

If you are using the DataDirect Connection Pool Manager for connection pooling, you can enable reauthentication in the Pool Manager (see [Appendix I “Connection Pool Manager” on page 789](#) for details).

---

## Using 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 it prevents the overhead of repeatedly parsing and creating cursors for the same statement, along with the associated network round trips to the database server.

You can use the DataDirect 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 DataDirect Connect for JDBC drivers. It is implemented as a Java Management Extensions (JMX) MBean. Applications can access the Statement Pool Monitor using the standard JMX API or using methods in the DataDirect-specific ExtConnection interface. Because the Statement Pool Monitor is implemented as a JMX MBean, it can easily be used by any JMX-compliant tool, such as JConsole.

See [Appendix J “Statement Pool Monitor” on page 811](#) for more information about managing statement pools with the DataDirect Statement Pool Monitor.

---

## Using Failover

To ensure continuous, uninterrupted access to data, DataDirect Connect for JDBC provides the following levels of failover protection, listed from basic to more comprehensive:

- *Connection failover* provides failover protection for new connections only. The driver fails over new connections to an alternate, or backup, database server if the primary database server is unavailable, for example, because of a hardware failure or traffic overload. If a connection to the database is lost, or dropped, the driver does not fail over the connection. This failover method is the default.
- *Extended connection failover* provides failover protection for new connections and lost database connections. If a connection to the database is lost, the driver fails over the connection to an alternate server, preserving the state of the connection at the time it was lost, but not any work in progress.
- *Select failover* provides failover protection for new connections and lost database connections. In addition, it provides protection for Select statements that have work in progress. If a connection to the database is lost, the driver fails over the connection to an alternate server, preserving the state of the connection at the time it was lost and preserving the state of any work being performed by Select statements.

The method you choose depends on how failure tolerant your application is. For example, if a communication failure occurs

while processing, can your application handle the recovery of transactions and restart them? If so, your application may only need the failover protection provided by the connection failover or extended connection failover methods. If your application has no built-in recovery of transactions, the driver can provide this failover protection using the Select failover method.

You can specify which failover method you want to use by setting the FailoverMode connection property. Read the following sections for details on each failover method:

- “[Connection Failover](#)” on page 59
- “[Extended Connection Failover](#)” on page 61
- “[Select Failover](#)” on page 62

Regardless of the failover method you choose, you must configure one or multiple alternate servers using the AlternateServers connection property. See “[Guidelines for Primary and Alternate Servers](#)” on page 63 for information about primary and alternate servers.

## Connection Failover

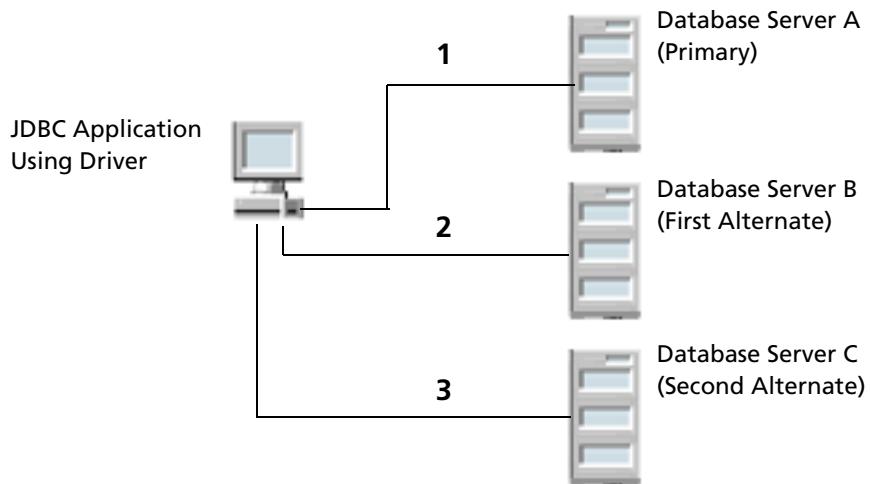
Connection failover provides failover protection for new connections only and does not provide protection for lost connections to the database. If the primary database server is unavailable when the driver makes a new connection, the driver fails over the connection to an alternate database server. Alternate servers are tried until a connection is successfully established or until all database servers (primary and alternate) have been tried a specified number of times.

For example, as shown in [Figure 2-1](#), Database Server A is the primary database server, Database Server B is the first alternate server, and Database Server C is the second alternate server.

---

**Figure 2-1. Connection Failover**

---



First, the driver attempts to connect to the primary database server, Database Server A (1). If connection failover is enabled and Database Server A fails to accept the connection, the driver attempts to connect to Database Server B (2). If that connection attempt also fails, the driver attempts to connect to Database Server C (3).

In this scenario, it is probable that at least one connection attempt would succeed, but if no connection attempt succeeds, the driver retries each database server (primary and alternate) for a specified number of attempts. You can customize the *connection retry* feature to specify the number of attempts that are made. For more information about connection retry, see ["Using Connection Retry" on page 66](#).

The driver fails over to the next alternate database server only if a successful connection cannot be established with the current alternate server. If the driver successfully establishes communication with a database server and the connection request is rejected by the database server because login

credentials are invalid, for example, the driver does not try to connect to the next database server in the list and generates an exception.

## Extended Connection Failover

Extended connection failover provides failover protection for the following types of connections:

- New connections, in the same way as described in [“Connection Failover” on page 59](#)
- Lost connections

When a connection to the database is lost, the driver fails over the connection to an alternate server, restoring the same state of the connection at the time it was lost. For example, when reestablishing a lost connection on the alternate database server, the driver performs the following actions:

- Restores the connection using the same connection properties specified by the lost connection
- Reallocates statement objects and attributes
- Logs in the user to the database with the same user credentials
- Restores any prepared statements associated with the connection and repopulates the statement pool
- Restores manual commit mode if autocommit mode was turned off

The driver does not preserve work in progress. For example, if the database server experienced a hardware failure while processing a query, partial rows processed by the database and returned to the client would be lost.

If an exception occurs while the driver is reestablishing a lost connection, the driver can fail the entire failover process or proceed with the process as far as it can. For example, suppose an exception occurred while reestablishing the connection because the driver was unable to log the user into the database. In this case, you may want the driver to notify your application of the exception and proceed with the failover process. You can choose how you want the driver to behave if exceptions occur during failover by setting the `FailoverGranularity` connection property.

During the failover process, your application may experience a short pause while the driver establishes a new connection or reestablishes a lost connection on an alternate server. If your application is time-sensitive (a real-time customer order application, for example) and cannot absorb this wait, you can set the `FailoverPreconnect` property to true. Setting the `FailoverPreconnect` property to true instructs the driver to establish connections to the primary server and an alternate server at the same time. Your application uses the first connection that is successfully established. As a bonus, if this connection to the database is lost at a later time, the driver saves time in reestablishing the connection on the server it fails over to because it can use the spare connection in its failover process.

## Select Failover

Extended connection failover provides failover protection for the following types of connections:

- New connections, in the same way as described in [“Connection Failover” on page 59](#)
- Lost connections, in the same way as described in [“Extended Connection Failover” on page 61](#)

In addition, the driver can recover work in progress because it keeps track of the last `Select` statement the application executed on each `Statement` object, including how many rows were

fetched to the client. For example, if the database had only processed 500 of 1,000 rows requested by a Select statement when the connection was lost, the driver would reestablish the connection to an alternate server, reexecute the Select statement, and position the cursor on the next row so that the driver can continue fetching the balance of rows as if nothing had happened.

**NOTE:** The driver only recovers work requested by Select statements. You must explicitly restart the following types of statements after a failover occurs:

- Insert, Update, or Delete statements
- Statements that modify the connection state, for example, SET or ALTER SESSION statements
- Objects stored in a temporary tablespace or global temporary table
- Partially executed stored procedures and batch statements

By default, the driver verifies that the rows that are restored match the rows that were originally fetched and, if they do not match, generates an exception warning your application that the Select statement must be reissued. By setting the FailoverGranularity connection property, you can customize the driver to ignore this check altogether or fail the entire failover process if the rows do not match.

## Guidelines for Primary and Alternate Servers

Many databases provide advanced database replication technologies such as DB2 High Availability Disaster Recovery (HADR), Oracle Real Application Clusters (RAC), and Microsoft Cluster Server (MSCS). The failover functionality provided by the drivers does not require any of these technologies, but can work

with them to provide comprehensive failover protection. Use the following guidelines for primary and alternate servers to ensure that failover works correctly in your environment:

- Alternate servers should mirror data on the primary server or be part of a configuration where multiple database nodes share the same physical data.
- If using failover with DB2 HADR, the primary server must be the primary server configured in your HADR system and any alternate server must be a standby server configured in your HADR system.
- If using failover with an Oracle RAC, the primary server must be a primary node and any alternate server must be a secondary node.
- If using failover with MSCS, which determines the alternate server for failover instead of the driver, any alternate server specified must be the same as the primary server. For example:

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=TEST;User=test;Password=secret;AlternateServers=(server1:1433;DatabaseName=TEST)
```

---

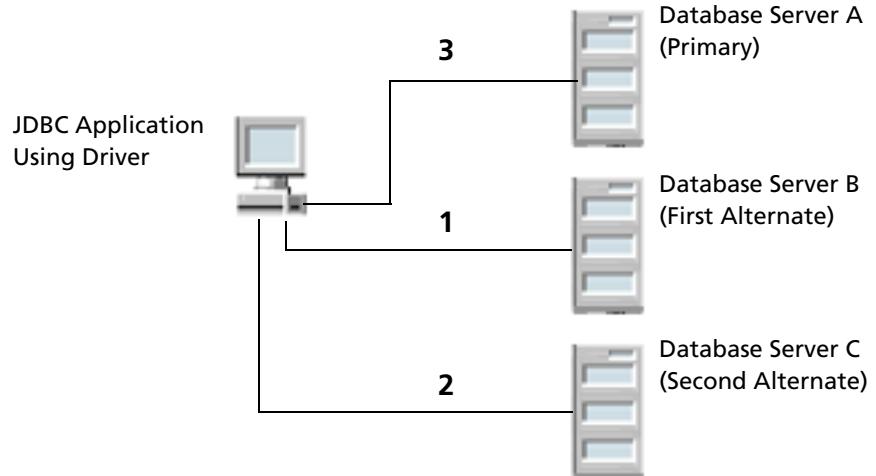
# Using Client Load Balancing

*Client load balancing* helps distribute new connections in your environment so that no one server is overwhelmed with connection requests. When client load balancing is enabled, the order in which primary and alternate database servers are tried is random. For example, let us suppose that client load balancing is enabled as shown in [Figure 2-2](#).

---

**Figure 2-2. Client Load Balancing**

---



First, Database Server B is tried (1). Then, Database Server C may be tried (2), followed by a connection attempt to Database Server A (3). In contrast, if client load balancing was not enabled in this scenario, each database server would be tried in sequential order, primary server first, and then, each alternate server based on their entry order in the alternate servers list.

For details on configuring client load balancing, see the appropriate driver chapter.

## Using Connection Retry

*Connection retry* defines the number of times that the driver attempts to connect to the primary, and, if configured, alternate database servers after the first unsuccessful connection attempt. Connection retry can be an important strategy for system recovery. For example, suppose you have a power failure scenario in which both the client and the server fail. When the power is restored and all computers are restarted, the client may be ready to attempt a connection before the server has completed its startup routines. If connection retry is enabled, the driver can continue to retry the connection until a connection is successfully accepted by the server.

Connection retry can be used in environments that only have one server or can be used as a complementary feature to connection failover in environments with multiple servers.

Using the `ConnectionRetryCount` and `ConnectionRetryDelay` properties, you can specify the number of times the driver attempts to connect and the time in seconds between connection attempts. For details on configuring connection retry, see the appropriate driver chapter.

---

# Using Security

The DataDirect Connect *for JDBC* drivers support the following security features: authentication and data encryption. For the most current information, refer to the security matrix on the DataDirect Technologies Web site:

<http://www.datadirect.com/products/security/documentation/securitymatrix.htm>

## Authentication

On most computer systems, a password is used to prove a user's identity. This password often is transmitted over the network and can possibly be intercepted by malicious hackers. Because this password is the one secret piece of information that identifies a user, anyone knowing a user's password can effectively *be* that user. Authentication methods protect the identity of the user. DataDirect Connect *for JDBC* drivers support the following authentication methods:

- User ID/password authentication authenticates the user to the database using a database user name and password.
- Kerberos is a trusted third-party authentication service. The drivers support both Windows Active Directory Kerberos and MIT Kerberos implementations for DB2, Oracle, and Sybase. For Microsoft SQL Server, the driver supports Windows Active Directory Kerberos only.
- Client authentication uses the user ID of the user logged onto the system on which the driver is running to authenticate the user to the database. The database server relies on the client to authenticate the user and does not provide additional authentication.

- NTLM authentication is a single sign-on authentication method for Windows environments. This method provides authentication from Windows clients only.

[Table 2-4](#) shows the authentication methods supported by the DataDirect Connect for JDBC drivers.

---

**Table 2-4. Authentication Methods Supported by the DataDirect Connect for JDBC Drivers**

---

Driver	User ID/ Password	Kerberos <sup>1</sup>	Client	NTLM
DB2 for Linux/UNIX/Windows	X	X	X	
DB2 for z/OS	X	X	X	
DB2 for iSeries	X		X	
Informix	X			
MySQL	X			
Oracle	X	X	X	X
Microsoft SQL Server	X	X <sup>2</sup>		X
Sybase	X	X		

- 
1. For DB2, Oracle, and Sybase, the drivers support the Windows Active Directory KDC and MIT Kerberos KDC. For Microsoft SQL Server, the driver supports the Windows Active Directory KDC only.
  2. Supported for Microsoft SQL Server 2000 and higher.
- 

See the individual driver chapter for details about configuring authentication:

- For DB2, see “[Authentication](#)” on page 146.
- For Oracle, see “[Authentication](#)” on page 340
- For Microsoft SQL Server, see “[Authentication](#)” on page 428
- For Sybase, see “[Authentication](#)” on page 512

# Data Encryption Across the Network

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. 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.

DataDirect Connect *for JDBC* drivers support the following encryption methods:

- Database-specific encryption (DB2 for Linux/UNIX/Windows and DB2 for z/OS only). DB2 defines its own encryption protocol. See “[Data Encryption](#)” on page 155 for information about configuring DB2 encryption.
- Secure Sockets Layer (SSL). 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.

[Table 2-5](#) shows the data encryption methods supported by the DataDirect Connect for JDBC drivers.

**Table 2-5. Data Encryption Methods Supported by the DataDirect Connect for JDBC Drivers**

Driver	Database-Specific	SSL
DB2 for Linux/UNIX/Windows	X	X <sup>1</sup>
DB2 for z/OS	X	X <sup>2</sup>
DB2 for iSeries		X <sup>3</sup>
Informix		
MySQL		X
Oracle		X
Microsoft SQL Server		X <sup>4</sup>
Sybase		X

1. Supported for V9.1 Fixpack 2 and higher for Linux/UNIX/Windows.
2. Supported for DB2 v9.1 for z/OS.
3. Supported for DB2 V5R3 and higher for iSeries.
4. Supported for Microsoft SQL Server 2000 and higher.

## SSL Encryption

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

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

NOTE: DB2, MySQL, and Oracle are the only databases supported by DataDirect Connect for JDBC that support SSL client authentication.

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

See the individual driver chapters for details about configuring SSL:

- For DB2, see “[Data Encryption](#)” on page 155.
- For MySQL, see “[Data Encryption](#)” on page 250
- For Oracle, see “[Data Encryption](#)” on page 352
- For Microsoft SQL Server, see “[Data Encryption](#)” on page 439
- For Sybase, see “[Data Encryption](#)” on page 520

## ***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 SSL sockets created in the JVM.

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

```
TrustStore=C:\Certificates\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 DataDirect Connect for JDBC 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.

## ***SSL Client Authentication (DB2, MySQL, and Oracle)***

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

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

The DataDirect Connect *for JDBC* 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 keystore contains only one certificate. To gain access to the certificate and its private key, the driver must provide only 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 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:\Certificates\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.

---

## Storing and Returning Client Information for Connections

Many databases allow applications to store client information associated with a connection. This information can be useful for database administration and monitoring purposes. For DB2 V9.5 and V9.7 for Linux/UNIX/Windows and DB2 for z/OS, this information can feed directly into the Workload Manager (WLM) for workload management and monitoring purposes. See “[Workload Manager \(WLM\)](#)” on page 160 for more information about using the WLM.

The DataDirect Connect *for JDBC* drivers allow your application to store and return the following types of client information:

- Name of the application
- User ID
- Host name of the client
- Additional accounting information, such as an accounting ID
- Product name and version of the driver

See [Appendix C “Client Information for Connections” on page 621](#) for more information.

---

## Using DataDirect Bulk Load

The DB2, Oracle, SQL Server, and Sybase drivers support DataDirect Bulk Load, a feature that allows your application to send large numbers of rows of data to the database in a continuous stream instead of in numerous smaller database protocol packets. Similar to a batch operation, performance improves because far fewer network round trips are required. Bulk load bypasses the data parsing that is usually done by the

database, providing an additional performance gain over batch operations.

NOTES:

- The Oracle driver supports bulk load only for Oracle 9i R2 and higher.
- Because a bulk load operation may bypass data integrity checks, your application must ensure that the data it is transferring does not violate integrity constraints in the database. For example, suppose you are bulk loading data into a database table and some of that data duplicates data stored as a primary key, which must be unique. The driver will not throw an exception to alert you to the error; your application must provide its own data integrity checks.

**IMPORTANT:** DataDirect Bulk Load requires a licensed installation of the drivers. If the drivers are installed with an evaluation license, the bulk load feature is available for prototyping with your applications, but with limited scope. Contact your sales representative or DataDirect SupportLink for further information.

See [Appendix K “Using DataDirect Bulk Load” on page 829](#) for more information about using DataDirect Bulk Load.

---

## Testing Connections

See [Appendix G “Using DataDirect Test™” on page 729](#) for instructions on testing JDBC driver connections using DataDirect Test.

---

# Required Permissions for the Java 2 Platform

Using the drivers on a Java 2 Platform with the standard Security Manager enabled requires certain permissions to be set in the security policy file of the Java 2 Platform. This security policy file can be found in the jre/lib/security subdirectory of the Java 2 Platform installation directory.

NOTE: Web browser applets running in the Java 2 plug-in always run in a JVM with the standard Security Manager enabled.

To run an application on a Java 2 Platform with the standard Security Manager, use the following command:

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

where *application\_class\_name* is the class name of the application.

Refer to your Java 2 Platform documentation for more information about setting permissions in the security policy file.

## Permissions for Establishing Connections

To establish a connection to the database server, the DataDirect Connect for JDBC drivers must be granted the permissions as shown in the following example:

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

where *install\_dir* is the DataDirect Connect for JDBC installation directory.

In addition, if Microsoft SQL Server named instances are used, permission must be granted for the listen and accept actions as shown in the following example:

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

where *install\_dir* is the DataDirect Connect *for JDBC* installation directory.

## Granting Access to Java Properties

To allow the DataDirect Connect *for JDBC* drivers to read the value of various Java properties to perform certain operations, permissions must be granted as shown in the following example:

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

where *install\_dir* is the DataDirect Connect *for JDBC* installation directory.

## Granting Access to Temporary Files

Access to the temporary directory specified by the JVM configuration must be granted in the security policy file of the Java 2 Platform to use insensitive scrollable cursors or to perform client-side sorting of DatabaseMetaData result sets. The following example shows permissions that have been granted for the C:\TEMP directory:

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

where *install\_dir* is the DataDirect Connect *for JDBC* installation directory.

## Granting Access to Oracle tnsnames.ora Files

If you are using an Oracle tnsnames.ora file to connect with the DataDirect Connect *for JDBC* Oracle driver, read access to the tnsnames.ora file must be granted to the driver in the security policy file of the Java 2 Platform.

```
grant codeBase "file:/install_dir/lib/-" {  
    permission java.io.FilePermission "C:\\oracle\\ora92\\network\\admin\\  
    tnsnames.ora", "read";  
};
```

where *install\_dir* is the DataDirect Connect *for JDBC* installation directory.

See “[Using tnsnames.ora Files](#)” on page 319 for more information about using tnsnames.ora files to connect to Oracle databases.

## Permissions for Kerberos Authentication

To use Kerberos authentication with the DataDirect Connect *for JDBC* drivers that support it, the application and driver code bases must be granted security permissions in the security policy file of the Java 2 Platform as shown in the following examples.

For more information about using Kerberos authentication with the DataDirect Connect *for JDBC* drivers, see the appropriate driver chapter.

## **DB2**

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

where:

*install\_dir* is the DataDirect Connect for JDBC installation directory.

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

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

*db\_hostname* is the host name of the machine running the database.

## **Microsoft SQL Server**

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

where:

*install\_dir* is the DataDirect Connect for JDBC installation directory.

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

*db\_hostname* is the host name of the machine running the database.

*SQLServer\_port* is the TCP/IP port on which the Microsoft SQL Server instance is listening.

## Sybase

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

where:

*install\_dir* is the DataDirect Connect for JDBC installation directory.

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

*principal\_name* is the service principal name registered with the KDC that identifies the database service.

*db\_hostname* is the host name of the machine running the database.

## Unicode Support

Multi-lingual applications can be developed on any operating system platform with JDBC using the DataDirect Connect *for JDBC* drivers to access both Unicode and non-Unicode enabled databases. Internally, Java applications use UTF-16 Unicode encoding for string data. When fetching data, the drivers automatically perform the conversion from the character encoding used by the database to UTF-16. Similarly, when inserting or updating data in the database, the drivers automatically convert UTF-16 encoding to the character encoding used by the database.

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

---

## Error Handling

The DataDirect Connect *for JDBC* drivers report errors to the application by throwing SQLExceptions. Each SQLException contains the following information:

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

## Driver Errors

An error generated by the DataDirect Connect for JDBC drivers has the following format:

[DataDirect][Connect for JDBC Driver]message

For example:

[DataDirect][Sybase JDBC Driver]Timeout expired.

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

## Database Errors

An error generated by the database has the following format:

[DataDirect][Connect for JDBC Driver][Database] message

For example:

[DataDirect][SQL Server JDBC Driver][SQL Server]  
Invalid Object Name.

Use the native error code to look up details about the possible cause of the error. For these details, refer to your database documentation.



# 3 The DB2 Driver

The DataDirect Connect *for JDBC* DB2 driver (the "DB2 driver") supports:

- DB2 V9.1, V9.5, and V9.7 for Linux, UNIX, and Windows via DRDA
- DB2 Universal Database (UDB) v7.x and v8.x for Linux, UNIX, and Windows via DRDA
- DB2 UDB v7.x, v8.1, and v9.1 for z/OS via DRDA
- DB2 UDB V5R1, V5R2, V5R3, V5R4, and V6R1 for iSeries via DRDA

NOTE: This documentation uses the following terms to describe the different DB2 versions:

- "DB2 for Linux/UNIX/Windows" refers to all versions of DB2 on Linux, UNIX, and Windows platforms.
- "DB2 for z/OS" refers to all versions of DB2 on z/OS platforms.
- "DB2 for iSeries" refers to all versions of DB2 on iSeries platforms.

## Data Source and Driver Classes

The data source class for the DB2 driver is:

`com.ddtek.jdbcx.db2.DB2DataSource`

See [“Connecting Through Data Sources” on page 48](#) for information about DataDirect Connect *for JDBC* data sources.

The driver class for the DB2 driver is:

`com.ddtek.jdbc.db2.DB2Driver`

---

## Connection URL

The connection URL format for the DB2 driver is:

`jdbc:datadirect:db2://hostname:port[;property=value[;...]]`

where:

*hostname* is the IP address or TCP/IP host name of the server to which you are connecting. See [“Using IP Addresses” on page 50](#) for details on using IP addresses.

NOTE: Untrusted applets cannot open a socket to a machine other than the originating host.

*port* is the number of the TCP/IP port.

*property=value* specifies connection properties. For a list of connection properties and their valid values, see [“Connection Properties” on page 88](#).

For example:

#### **DB2 for Linux, UNIX, and Windows<sup>1</sup>**

```
jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc;User=test;  
Password=secret
```

#### **DB2 for z/OS and iSeries<sup>1</sup>**

```
jdbc:datadirect:db2://server1:446;LocationName=Sample;User=test;  
Password=secret
```

---

## **J2EE Connector Architecture Resource Adapter Class**

The ManagedConnectionFactory class for the DB2 resource adapter is:

`com.ddtek.resource.spi.DB2ManagedConnectionFactory`

See “[J2EE Connector Architecture Resource Adapters](#)” on [page 37](#) for information about using DataDirect Connect *for JDBC* drivers as J2EE Connector Architecture resource adapters.

---

1. See “[DB2 Packages](#)” on [page 134](#) for information about creating DB2 packages

---

# Connection Properties

This section lists the JDBC connection properties supported by the DB2 driver and describes each property. The properties have the form:

*property=value*

You can use these connection properties with either the JDBC Driver Manager or DataDirect Connect *for JDBC* data sources unless otherwise noted.

NOTES:

- All connection property names are case-insensitive. For example, Password is the same as password. Required properties are noted as such.
- The data type listed for each connection property refers to the Java data type used for the property value in a JDBC data source.

## AccountingInfo

Description Accounting information to be stored in the database. This value sets the CURRENT CLIENT\_ACCTNG register (DB2 for Linux/UNIX/Windows) or the CLIENT\_ACCTNG register (DB2 for z/OS and DB2 for iSeries) in the database. This value is used for database administration/monitoring purposes.

Valid Values *string*

where *string* is the accounting information.

NOTE: Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. See “[Returning MetaData About Client Information Locations](#)” on page 627.

Default	empty string
Data Type	String
See Also	<a href="#">"Client Information for Connections" on page 160</a>

## AddToCreateTable

Description	A string that is appended to the end of all CREATE statements. This property can be used to add an "in database" clause.
Valid Values	<i>string</i> where <i>string</i> is the set of characters appended to all CREATE statements.
Default	None
Data Type	String

## AllowImplicitResultSetCloseForXA

Description	Specifies whether result sets in distributed transactions are automatically closed when all rows of the result sets have been returned.
Valid Values	true   false  If set to true, the driver automatically closes result sets in distributed transactions when all rows of the result sets have been returned.  If set to false, the driver does not automatically close result sets in distributed transactions when all rows of the result sets have been returned. Because automatically closing results sets in distributed transactions does not work correctly with DB2 v8.1 for Linux/UNIX/Windows servers prior to Fix Pack 5, use this value if your application accesses DB2 v8.1 for Linux/UNIX/Windows prior to Fix Pack 5.

Default	true
Data Type	boolean

## AlternateID

Description	The name of the schema to be used to qualify unqualified database objects in dynamically prepared SQL statements.  For DB2 for Linux/UNIX/Windows and DB2 for iSeries, this property sets the value in the DB2 CURRENT SCHEMA special register.  For DB2 for z/OS, this property sets the value in the DB2 CURRENT SQLID special register.
Valid Values	For DB2 for Linux/UNIX/Windows and DB2 for iSeries, a valid DB2 schema name. This value is not validated by the database server.  For DB2 for z/OS, this value is validated by the database server. Refer to your IBM documentation for valid values for the CURRENT SQLID register.
Default	None
Data type	String

## AlternateServers

Description	A list of alternate database servers that is used to failover new or lost connections, depending on the failover method selected. See the <a href="#">FailoverMode</a> property for information about choosing a failover method.
Valid Values	( <i>servername1[:port1][;property=value[;...]]</i> [ , <i>servername2[:port2][;property=value[;...]]</i> ...]...)

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 50000 is used.

Optional connection properties are **DatabaseName** (DB2 for Linux/UNIX/Windows) and **LocationName** (DB2 for z/OS and DB2 for iSeries).

**NOTE:** If using failover with DB2 High Availability Disaster Recovery (HADR), the primary server and any alternate server must be the primary server and a standby server, respectively, that is configured in your HADR system.

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

```
jdbc:datadirect:db2://server1:50000;DatabaseName=TEST;
User=test;Password=secret;AlternateServers=(server2:50000;
DatabaseName=TEST2,server3:50000;DatabaseName=TEST3)
```

**Default** None

**Data type** String

**See Also** [“Configuring Failover” on page 171](#)

## ApplicationName

**Description** The name of the application to be stored in the database. This value sets the CURRENT CLIENT\_APPLNAME register (DB2 for Linux/UNIX/Windows) or CLIENT\_APPLNAME register (DB2 for z/OS and DB2 for iSeries) in the database. For DB2 V9.1 and higher for Linux/UNIX/Windows, this value also sets the APPL\_NAME value of the SYSIBMADM.APPLICATIONS table. These values are used for database administration/monitoring purposes.

Valid Values *string*

where *string* is the name of the application.

NOTE: Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. See “[Returning MetaData About Client Information Locations](#)” on page 627.

Default empty string

Data Type String

See Also [“Client Information for Connections” on page 160](#)

## AuthenticationMethod

Description Determines which authentication method the driver uses when it establishes 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 kerberos | encryptedUIDPassword | encryptedPassword | clearText | client

If set to kerberos, the driver uses Kerberos authentication. The driver ignores any user ID or password specified.

If set to encryptedUIDPassword, the driver uses user ID/password authentication. The driver sends an encrypted user ID and password to the DB2 server for authentication. If a user ID and password are not specified, the driver throws an exception. If this value is set, the driver can also use data encryption.

If set to encryptedPassword, the driver uses user ID/password authentication. The driver sends a user ID in clear text and an encrypted password to the DB2 server for authentication. If a user ID and password are not specified, the driver throws an exception. If this value is set, the driver can also use data encryption.

If set to clearText, the driver uses user ID/password authentication. The driver sends the user ID and password in clear text to the DB2 server for authentication. If a user ID and password are not specified, the driver throws an exception. If this value is set, the driver can also use data encryption.

If set to client, the driver uses client authentication. The DB2 server relies on the client to authenticate the user and does not provide additional authentication. The driver ignores any user ID or password specified.

**NOTE:** The [User](#) property provides the user ID. The [Password](#) property provides the password.

**Default** clearText

**Data Type** String

**See Also** [“Authentication” on page 146](#)

## BatchPerformanceWorkaround

**Description** The driver uses the native DB2 batch mechanism. This property determines whether restrictions are enforced to facilitate data conversions.

**Valid Values** true | false

If set to true, restrictions are removed; however, parameter sets may not be executed in the order that they were specified.

If set to false, the methods that are used to set the parameter values of a batch operation that is performed using a `PreparedStatement` must match the database data type of the column with which the parameter is associated. This is because DB2 servers do not perform implicit data conversions.

**Default** false

Data Type boolean

See Also ["Batch Inserts and Updates" on page 163](#)

## BulkLoadBatchSize

Description 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.

### 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 set by this property. See ["DDBulkLoad Interface" on page 603](#) for a description of the method.

Valid Values  $x$

where  $x$  is a positive integer.

Default 2048

Data Type long

See Also ["Using DataDirect Bulk Load" on page 829](#)

## CatalogIncludesSynonyms

Description	Determines whether synonyms are included in the result sets that are returned from the DatabaseMetaData.getColumns() method.  This property is ignored for DB2 v8.x and higher for Linux/UNIX/Windows. By default, the driver returns synonyms for the DatabaseMetaData.getColumns() method when connected to DB2 v8.x and higher for Linux/UNIX/Windows.
Valid Values	true   false  If set to true, synonyms are included in the result sets that are returned from the DatabaseMetaData.getColumns() method.  If set to false, synonyms are omitted from result sets that are returned from the DatabaseMetaData.getColumns() method.
Default	true
Data Type	boolean
See Also	<a href="#">“Performance Considerations” on page 131</a>

## CatalogOptions

Description	Determines which type of metadata information is included in result sets when an application calls DatabaseMetaData methods.
Valid Values	0   2   6  If set to 0, result sets do not contain synonyms.  If set to 2, result sets contain synonyms that are returned from the following DatabaseMetaData methods: getColumns(), getExportedKeys(), getFunctionColumns(), getFunctions(), getImportedKeys(), getIndexInfo(), getPrimaryKeys(), getProcedureColumns(), and getProcedures().

If set to 6, a hint is provided to the driver to emulate `getColumns()` calls using the `ResultSetMetaData` object instead of querying database catalogs for column information. Result sets contain synonyms. 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.

Default 2

Data Type int

See Also [“Performance Considerations” on page 131](#)

## CatalogSchema

Description The DB2 schema to use for catalog methods.

To improve performance, views of system catalog tables can be created in a schema other than the default catalog schema. Setting this property to a schema that contains views of the catalog tables allows the driver to use those views. To ensure that catalog methods function correctly, views for specific catalog tables must exist in the specified schema. The views that are required depend on your DB2 database. See [“Non-Default Schemas for Catalog Methods” on page 157](#) for views for catalog tables that must exist in the specified schema.

Valid Values *string*

where *string* is the name of a valid DB2 schema.

Default SYSCAT (DB2 for Linux/UNIX/Windows) | SYSIBM (DB2 for z/OS) | QSYS2 (DB2 for iSeries)

Data Type String

See Also [“Performance Considerations” on page 131](#)

## CharsetFor65535

Description	The code page to be used by the driver to convert character data that is stored as bit data in character columns (Char, Varchar, Longvarchar, Char for Bit Data, Varchar for Bit Data, Longvarchar for Bit Data) defined with CCSID 65535. All character data that is stored as bit data and returned from the database using columns defined with CCSID 65535 is converted using the specified code page. This property has no effect when writing data to character columns that are defined with CCSID 65535.
Valid Values	<i>string</i> where <i>string</i> is the name of a valid code page that is supported by your JVM.
Example	CP950
Default	None
Data Type	String

## ClientHostName

Description	The host name of the client machine to be stored in the database. This value sets the CURRENT CLIENT_WRKSTNNNAME register (DB2 for Linux/UNIX/Windows) or CLIENT_WRKSTNNNAME register (DB2 for z/OS and DB2 for iSeries) in the database. This value is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is the host name of the client machine.  NOTE: Your database may impose character length restrictions on the value that is set by this property. If the value exceeds a restriction, the driver truncates it. See " <a href="#">Returning MetaData About Client Information Locations</a> " on page 627.

Default	empty string
Data Type	String
See Also	<a href="#">"Client Information for Connections" on page 160</a>

## ClientUser

Description	The user ID to be stored in the database. This property sets the CURRENT CLIENT_USERID register (DB2 for Linux/UNIX/Windows) and CLIENT USERID register (DB2 for z/OS and DB2 for iSeries) in the database. This value is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is a valid user ID.
	NOTE: Your database may impose character length restrictions on the value that is set by this property. If the value exceeds a restriction, the driver truncates it. See <a href="#">"Returning MetaData About Client Information Locations" on page 627</a> .
Default	empty string
Data Type	String
See Also	<a href="#">"Client Information for Connections" on page 160</a>

## CodePageOverride

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

**Valid Values** *string*

where *string* is the name of a valid code page that is supported by your JVM.

**Example** CP950

**Default** None

**Data Type** String

## CollectionId

**Description** DEPRECATED. This property is recognized for backward compatibility. Use the [PackageCollection](#) property to specify the name of the collection or library (group of packages) to which DB2 packages are bound.

## ConnectionRetryCount

**Description** The number of times the driver retries connection attempts to the primary database server, and if specified, alternate servers until a successful connection is established.

NOTE: 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.

**Valid Values** 0 | *x*

where *x* is a positive integer.

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.

NOTE: The [ConnectionRetryDelay](#) property specifies the wait interval, in seconds, to occur between retry attempts.

Example	If this property is set to 2 and alternate servers are specified using the <a href="#">AlternateServers</a> property, the driver retries the list of servers (primary and alternate) twice after the initial retry attempt.
Default	5 (seconds)
Data Type	int
See Also	<a href="#">"Configuring Failover" on page 171</a>

## ConnectionRetryDelay

Description	The number of seconds the driver waits between connection retry attempts when <a href="#">ConnectionRetryCount</a> is set to a positive integer.
Valid Values	0   $x$ where $x$ is a number of seconds.  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 <a href="#">ConnectionRetryCount</a> is set to 2, this property is set to 3, and alternate servers are specified using the <a href="#">AlternateServers</a> 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.
Default	1 (second)
Data Type	int
See Also	<a href="#">"Configuring Failover" on page 171</a>

## ConvertNull

Description	Controls how data conversions are handled for null values.
Valid Values	0   1
	If set to 0, the driver does not perform the data type check if the value of the column is null. This allows null values to be returned even though a conversion between the requested type and the column type is undefined.
	If set to 1, the driver checks the data type being requested against the data type of the table column that stores the data. If a conversion between the requested type and column type is not defined, the driver generates an "unsupported data conversion" exception regardless of the data type of the column value.
Default	1
Data Type	int

## CreateDefaultPackage

Description	Determines whether the driver automatically creates required DB2 packages.  For DB2 for Linux/UNIX/Windows, this property must be used in conjunction with the <a href="#">ReplacePackage</a> property.  For DB2 for z/OS and DB2 for iSeries, DB2 packages are created in the collection or library that is specified by the <a href="#">PackageCollection</a> property.
Valid Values	true   false  If set to true, the driver automatically creates required DB2 packages, even if they already exist. Existing DB2 packages are replaced by the new packages.

If set to false, the driver determines if the required DB2 packages exist. If they do not, the driver automatically creates them.

Default false

Data Type boolean

See Also ["DB2 Packages" on page 134](#)

## CurrentFunctionPath

Description A list of DB2 schema names that are used to resolve unqualified function names and data type references in dynamically prepared SQL statements. It also is used to resolve unqualified stored procedure names that are specified in CALL statements. This property sets the CURRENT PATH register in the database.

Valid Values *schema\_name* [ [ ,*schema\_name*] . . . ]

where *schema\_name* is a valid DB2 schema name.

Default null

Data Type String

## Database

Description An alias for the [DatabaseName](#) property.

## DatabaseName

Description REQUIRED. The name of the database to which you want to connect. This property is supported only for DB2 for Linux/UNIX/Windows.

NOTE: This property is an alias for [LocationName](#) when connecting to DB2 for z/OS or iSeries.

Valid Values	<i>string</i>
where <i>string</i> is the name of a DB2 database.	
Default	None
Data Type	String
Alias	<b>Database</b> property. If both the Database and DatabaseName properties are specified in a connection URL, the last property that is positioned in the connection URL is used. For example, if your application specifies the following connection URL, the value of the Database connection property would be used instead of the value of the DatabaseName connection property.
<pre>jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc; Database=acct;User=test;Password=secret</pre>	

## DynamicSections

Description	The maximum number of prepared statements that the driver can have open at any one time.
Valid Values	<i>x</i>
	where <i>x</i> is a positive integer.
Default	200
Data Type	int

## EnableBulkLoad

Description	Specifies whether the driver uses the native bulk load protocols in the database instead of the batch mechanism for batch inserts. Bulk load bypasses the data parsing that is usually done by the database, providing an additional performance gain over batch operations. This property allows existing applications with batch inserts to take advantage of bulk load without requiring changes to the application code.
-------------	--

Valid Values true | false

If set to true, the driver uses the native bulk load protocols for batch inserts.

NOTE: If set to true, any value set for [BatchPerformanceWorkaround](#) is ignored.

If set to false, the driver uses the batch mechanism for batch inserts.

Default false

Data Type boolean

See Also ["Using Bulk Load for Batch Inserts" on page 837](#)

["Performance Considerations" on page 131](#)

## EnableCancelTimeout

Description 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.

Default	false
Data Type	boolean

## EncryptionMethod

Description	Determines whether data is encrypted and decrypted when transmitted over the network between the driver and database server.  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 <a href="#">LoginTimeout</a> property to avoid problems when connecting to a server that does not support SSL.
Valid Values	<p>noEncryption   DBEncryption   requestDBEncryption   SSL</p> <p>If set to noEncryption, data is not encrypted or decrypted.</p> <p>If set to DBEncryption, data is encrypted using DES encryption if the database server supports it. If the database server does not support DES encryption, the connection fails and the driver throws an exception. The <a href="#">AuthenticationMethod</a> property must be set to a value of clearText, encryptedPassword, or encryptedUIDPassword. This value is not supported for DB2 for iSeries.</p> <p>If set to requestDBEncryption, data is encrypted using DES encryption if the database server supports it. If the database server does not support DES encryption, the driver attempts to establish an unencrypted connection. The <a href="#">AuthenticationMethod</a> property must be set to a value of clearText, encryptedPassword, or encryptedUIDPassword. This value is not supported for DB2 for iSeries.</p> <p>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.</p>

When SSL is enabled, the following properties also apply:

[HostNameInCertificate](#)

[KeyStore](#) (for SSL client authentication)

[KeyStorePassword](#) (for SSL client authentication)

[KeyPassword](#) (for SSL client authentication)

[TrustStore](#)

[TrustStorePassword](#)

[ValidateServerCertificate](#)

Default noEncryption

Data Type String

See Also [“Configuring SSL Encryption” on page 156](#)

[“Performance Considerations” on page 131](#)

## FailoverGranularity

Description Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. This property is ignored if [FailoverMode=connect](#).

Valid Values nonAtomic | atomic | atomicWithRepositioning | disableIntegrityCheck

If set to nonAtomic, the driver continues with the failover process and posts any exceptions on the statement on which they occur.

If set to atomic, the driver fails the entire failover process if an exception is generated as the result of restoring the state of the connection. If an exception is generated as a result of restoring the state of work in progress, the driver continues with the failover process, but generates an exception warning that the Select statement must be reissued.

If set to atomicWithRepositioning, the driver fails the entire failover process if any exception is generated as the result of

restoring the state of the connection or the state of work in progress.

If set to disableIntegrityCheck, the driver does not verify that the rows restored during the failover process match the original rows. This value is applicable only when FailoverMode=select.

**Default** nonAtomic

**Data Type** String

**See Also** ["Configuring Failover" on page 171](#)

## FailoverMode

**Description** Specifies the type of failover method the driver uses.

**Valid Values** connect | extended | select

If set to connect, the driver provides failover protection for new connections only.

If set to extended, the driver provides failover protection for new and lost connections, but not any work in progress.

If set to select, the driver provides failover protection for new and lost connections. In addition, it preserves the state of work performed by the last Select statement executed on the Statement object.

### NOTES:

- The [AlternateServers](#) property specifies one or multiple alternate servers for failover and is required for all failover methods.
- The [FailoverGranularity](#) property determines which action the driver takes if exceptions occur during the failover process.

- The [FailoverPreconnect](#) property specifies whether the driver tries to connect to multiple database servers (primary and alternate) at the same time.

Default connect

Data Type String

See Also [“Configuring Failover” on page 171](#)

## FailoverPreconnect

Description Specifies whether the driver tries to connect to the primary and an alternate server at the same time. This property is ignored if [FailoverMode=connect](#).

Valid Values true | false

If set to true, the driver tries to connect to the primary and an alternate server at the same time. This can be useful if your application is time-sensitive and cannot absorb the wait for the failover connection to succeed.

If set to false, the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection. This value provides the best performance, but your application typically experiences a short wait while the failover connection is attempted.

NOTE: The [AlternateServers](#) property specifies one or multiple alternate servers for failover.

Default false

Data Type boolean

See Also [“Configuring Failover” on page 171](#)

## Grantee

Description	The name of the schema to which you want to grant EXECUTE privileges for DB2 packages. This property is ignored if the <a href="#">GrantExecute</a> property is set to false.
Valid Values	<i>string</i> where <i>string</i> is a valid DB2 schema.  IMPORTANT: Using a value other than PUBLIC restricts access to use the driver. For example, if you set this property to TSMITH, only the user TSMITH would be allowed access to use the driver against the server.
Default	PUBLIC
Data Type	String
See Also	<a href="#">"DB2 Packages" on page 134</a>

## GrantExecute

Description	Determines which DB2 schema is granted EXECUTE privileges for DB2 packages.
Valid Values	true   false  If set to true, EXECUTE privileges are granted to the schema that is specified by the <a href="#">Grantee</a> property.  If set to false, EXECUTE privileges are granted to the schema that created the DB2 packages.
Default	true
Data Type	boolean
See Also	<a href="#">"DB2 Packages" on page 134</a>

## HostNameInCertificate

**Description** 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.

**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.

**Valid Values** *host\_name* | #SERVERNAME#

where *host\_name* is a valid host name.

If *host\_name* is specified, the driver compares the specified host name to the DNSName value of the SubjectAlternativeName in the certificate. If a DNSName value does not exist in the SubjectAlternativeName or if the certificate does not have a SubjectAlternativeName, the driver compares the host name with the Common Name (CN) part of the certificate's Subject name. 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 a DNSName value does not exist in the SubjectAlternativeName or 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.

Default	empty string
Data Type	String

## ImportStatementPool

Description	<p>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.</p> <p>If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver throws an exception.</p>
Valid Values	<p><i>string</i></p> <p>where <i>string</i> is the path and file name of the file to be used to load the contents of the statement pool.</p>

Default	empty string
Data Type	String
See Also	<a href="#">"Importing Statements into a Statement Pool" on page 820</a> <a href="#">"Performance Considerations" on page 131</a>

## InitializationString

Description	<p>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.</p>
-------------	--

**Valid Values** *command[[;command]...]*

where *command* is a SQL command.

**NOTE:** 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** The following connection URL adds USER2 to the CURRENT PATH special register and sets the CURRENT PRECISION special register to DEC31.

```
jdbc:datadirect:db2://server1:50000;
InitializationString=(SET CURRENT PATH=current_path,
USER2;SET CURRENT PRECISION='DEC31')
```

**NOTE:** Setting the CURRENT PRECISION special register is only valid for DB2 for z/OS.

**Default** None

**Data Type** String

## InInsensitiveResultSetBufferSize

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

**Valid Values** -1 | 0 | *x*

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 $x$ , where $x$ is a positive integer, 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.
Default	2048
Data Type	int
See Also	<a href="#">"Performance Considerations" on page 131</a>

## JavaDoubleToString

Description	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
	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.
Default	false
Data Type	boolean

## JDBCBehavior

Description	Determines how the driver describes database data types that map to the following JDBC 4.0 data types: NCHAR, NVARCHAR, NLONGVARCHAR, NCLOB, and SQLXML.
	This property is applicable only when the application is using Java SE 6.
Valid Values	0   1
	If set to 0, the driver describes the data types as JDBC 4.0 data types when using Java SE 6.
	If set to 1, the driver describes the data types using JDBC 3.0-equivalent data types, regardless of JVM. This allows your application to continue using JDBC 3.0 types in a Java SE 6 environment. In addition, the JDBC 4.0 method <code>ResultSet.getHoldability()</code> returns the value of the JDBC 3.0 method <code>Connection.getHoldability()</code> .
Default	1
Data Type	int

## KeyPassword

Description	Specifies the password that is used to access the individual keys in the keystore file when SSL is enabled ( <a href="#">EncryptionMethod=SSL</a> ) 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	<i>string</i>
	where <i>string</i> is a valid password.
Default	None
Data Type	String

## KeyStore

Description	<p>Specifies the directory of the keystore file to be used when SSL is enabled (<a href="#">EncryptionMethod=SSL</a>) 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.</p> <p>This value overrides the directory of the keystore file that is specified by the <code>javax.net.ssl.keyStore</code> Java system property. If this property is not specified, the keystore directory is specified by the <code>javax.net.ssl.keyStore</code> Java system property.</p> <p><b>NOTE:</b> The keystore and truststore files can be the same file.</p>
Valid Values	<i>string</i> where <i>string</i> is a valid directory of a keystore file.
Default	None
Data Type	String

## KeyStorePassword

Description	<p>Specifies the password that is used to access the keystore file when SSL is enabled (<a href="#">EncryptionMethod=SSL</a>) 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.</p> <p>This value overrides the password of the keystore file that is specified by the <code>javax.net.ssl.keyStorePassword</code> Java system property. If this property is not specified, the keystore password is specified by the <code>javax.net.ssl.keyStorePassword</code> Java system property.</p> <p><b>NOTE:</b> The keystore and truststore files can be the same file.</p>
-------------	---

Valid Values *string*

where *string* is a valid password.

Default None

Data Type String

## LoadBalancing

Description 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

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).

Default false

Data Type boolean

See Also [“Configuring Failover” on page 171](#)

## LocationName

Description	<p>REQUIRED. The name of the DB2 location that you want to access.</p> <p>For DB2 for z/OS, your system administrator can determine the name of your DB2 location using the following command:</p> <pre>DISPLAY DDF</pre> <p>For DB2 for iSeries, your system administrator can determine the name of your DB2 location using the following command. The name of the database that is listed as *LOCAL is the value you should use for this property.</p> <pre>WRKRDBDIR</pre> <p>This property is not supported for DB2 for Linux/UNIX/Windows.</p>
Valid Values	<p><i>string</i></p> <p>where <i>string</i> is the DB2 location.</p>
Default	None
Data Type	String

Alias [DatabaseName](#) property. If both the DatabaseName and LocationName connection properties are specified in a connection URL, the last property positioned in the connection URL is used. For example, if your application specifies the following connection URL, the value of the LocationName property would be used instead of the value of the DatabaseName property.

```
jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc;
LocationName=acct;User=test;Password=secret
```

## LoginTimeout

Description	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.  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.
Default	0
Data Type	int

## MaxPooledStatements

Description	The maximum number of pooled prepared statements for this connection. Setting MaxPooledStatements to an integer greater than zero (0) enables the driver's internal prepared statement pooling, which is useful when the driver is not running from within an application server or another application that provides its own prepared statement pooling.
Valid Values	0   $x$ where $x$ is a positive integer.  If set to 0, the driver's internal prepared statement pooling is not enabled.  If set to $x$ , the driver enables the DataDirect Statement Pool Monitor and uses the specified value to cache a certain number of prepared statements created by an application. If the value set for this property is greater than the number of prepared

statements that are used by the application, all prepared statements that are created by the application are cached. Because CallableStatement is a sub-class of PreparedStatement, CallableStatements also are cached.

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

**Default** 0

**Data Type** int

**Alias** [MaxStatements](#) property

**See Also** [“Performance Considerations” on page 131](#)

[Appendix J “Statement Pool Monitor” on page 811](#)

## MaxStatements

**Description** An alias for the [MaxPooledStatements](#) property.

## PackageCollection

**Description** The name of the collection or library (group of packages) to which DB2 packages are bound.

**NOTES:**

- This property is ignored for DB2 for Linux/UNIX/Windows.
- This property replaces the CollectionId property, which has been deprecated; however, the CollectionId property is still recognized for backward compatibility. If both the PackageCollection and CollectionId properties are specified, the CollectionId property is ignored.

**Default** NULLID

Data Type String

See Also ["DB2 Packages" on page 134](#)

## PackageOwner

Description The owner to be used for any DB2 packages that are created.

Valid Values *string*

where *string* is the owner of the DB2 packages.

Default NULL

Data Type String

See Also ["DB2 Packages" on page 134](#)

## Password

Description A password that is used to connect to your DB2 database. A password is required if security is enabled on your database. Contact your system administrator to obtain your password.

Valid Values *string*

where *string* is a valid password. The password is case-sensitive.

Default None

Data Type String

## PortNumber

Description The TCP port of the primary database server that is listening for connections to the DB2 database.

This property is supported only for data source connections.

Valid Values *port*

where *port* is the port number.

Default 50000

Data Type int

## ProgramID

Description The product and version information of the driver on the client to be stored in the database. This value sets the CLIENT\_PRDID value in the database. For DB2 V9.1 and higher for Linux/UNIX/Windows, this value is located in the SYSIBMADM.APPLICATIONS table.

Valid Values DDJVVRM

where:

- *DDJ* is an identifier for the DataDirect Connect *for JDBC* driver.
- *VV* identifies a 2-digit version number (with high-order 0 in the case of a 1-digit version).
- *RR* identifies a 2-digit release number (with high-order 0 in the case of a 1-digit release).
- *M* identifies a 1-character modification level (0-9 or A-Z).

Example DDJ04100

Default empty string

Data Type String

See Also “[Client Information for Connections](#)” on page 160

## QueryTimeout

Description	Sets the default query timeout (in seconds) for all statements created by a connection.
Valid Values	-1   0   $x$ where $x$ is a number of seconds.
	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 created by the connection. To override the default timeout value that is set by this property, call the Statement.setQueryTimeout() method to set a timeout value for a particular statement.
Default	0
Data Type	int

## ReplacePackage

Description	Determines whether the current bind process will replace the existing DB2 packages used by the driver.  For DB2 for Linux/UNIX/Windows, this property must be used in conjunction with the <a href="#">CreateDefaultPackage</a> property.
Valid Values	true   false  If set to true, the current bind process will replace the existing DB2 packages that are used by the driver.  If set to false, the current bind process will not replace the existing DB2 packages.

Default	false
Data Type	boolean
See Also	<a href="#">"DB2 Packages" on page 134</a>

## ResultSetMetaDataOptions

Description	Determines whether the driver returns table name information in the ResultSet metadata for Select statements.
Valid Values	0   1
	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.
Default	0
Data Type	int
See Also	<a href="#">"Performance Considerations" on page 131</a>

## SecurityMechanism

Description	DEPRECATED. This property is recognized for backward compatibility. Use the <a href="#">AuthenticationMethod</a> property to set the authentication method used by the driver.
-------------	--

## SendStreamAsBlob

Description	Determines whether binary stream data that is less than 32 KB is sent to the database as DB2 Long Varchar for Bit Data or Blob data. Binary streams that are larger than 32 KB can only be inserted into a Blob column. The driver always sends binary stream data larger than 32 KB to the database as Blob data.
Valid Values	true   false
	If set to true, the driver sends binary stream data that is less than 32 KB to the database as DB2 Blob data. If the target column is a Long Varchar for Bit Data column and not a Blob column, the Insert or Update statement fails. The driver automatically retries the Insert or Update statement, sending the data as Long Varchar for Bit Data, if the pointer in the stream can be reset to the beginning of the stream. If you know that you are sending the binary stream data to a Blob column, setting this value improves performance.
	If set to false, the driver sends binary stream data that is less than 32 KB to the database as Long Varchar for Bit Data data. If the target column is a Blob column and not a Long Varchar for Bit Data column, the Insert or Update statement fails. The driver retries the Insert or Update statement, sending the data as Blob data, if the pointer in the stream can be reset to the beginning of the stream.
Default	false
Data Type	boolean
See Also	<a href="#">"Performance Considerations" on page 131</a>

## ServerName

Description	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	<i>string</i>
	where <i>string</i> is a valid IP address or server name.
Example	122.23.15.12 or DB2Server
Default	None
Data Type	String

## SpyAttributes

Description	Enables DataDirect Spy to log detailed information about calls that are issued by the driver on behalf of the application. DataDirect Spy is not enabled by default.
Valid Values	( <i>spy_attribute</i> [; <i>spy_attribute</i> ]...)
where <i>spy_attribute</i> is any valid DataDirect Spy attribute. See <a href="#">Appendix H "Tracking JDBC Calls with DataDirect Spy™" on page 783</a> for a list of supported attributes.	
	NOTE: If coding a path on Windows to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: log=(file)C:\\temp\\\\spy.log.
Example	The following value instructs the driver to log all JDBC activity to a file using a maximum of 80 characters for each line.  (log=(file)/tmp/spy.log;linelimit=80)

Default None  
Data Type String

## StripNewlines

Description	Specifies whether newline characters in a SQL statement are sent to the DB2 server. If you know that SQL statements in your application do not contain newline characters, the driver can skip the removal of newline characters, improving performance by eliminating the parsing required by the DB2 server to remove them.
Valid Values	true   false
	If set to true, the driver removes all newline characters from SQL statements.
	If set to false, the driver does not remove any newline characters from SQL statements.
Default	false
Data Type	boolean
See Also	<a href="#">“Performance Considerations” on page 131</a>

## TrustStore

Description	Specifies the directory of the truststore file to be used when SSL is enabled using the <a href="#">EncryptionMethod</a> property and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.  This value overrides the directory of the truststore file that is specified by the javax.net.ssl.trustStore Java system property. If this property is not specified, the truststore directory is specified by the javax.net.ssl.trustStore Java system property.
-------------	---

This property is ignored if [ValidateServerCertificate=false](#).

**Valid Values** *string*

where *string* is the directory of the truststore file.

**Default** None

**Data Type** String

## TrustStorePassword

**Description** Specifies the password that is used to access the truststore file when SSL is enabled using the [EncryptionMethod](#) property 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.

**Default** None

**Data Type** String

## UseCurrentSchema

**Description** Specifies whether results are restricted to the tables and views in the current schema if a `DatabaseMetaData.getTables()` or `DatabaseMetaData.getColumns()` method is called without specifying a schema or if the schema is specified as the wildcard character %. Restricting results to the tables and views in the

current schema improves the performance of calls for `getTables()` methods that do not specify a schema.

**Valid Values** true | false

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

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

**Default** false

**Data Type** boolean

**See Also** [“Performance Considerations” on page 131](#)

## User

**Description** The user name that is used to connect to the DB2 database.

**Valid Values** *string*

where *string* is a valid user name. The user name is case-sensitive.

**Default** None

**Data Type** String

## ValidateServerCertificate

**Description** 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
	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 <a href="#">HostNameInCertificate</a> property is specified, the driver also validates the certificate using a host name. The <a href="#">HostNameInCertificate</a> 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 <a href="#">TrustStore</a> and <a href="#">TrustStorePassword</a> properties or Java system properties.
	NOTE: Truststore information is specified using the <a href="#">TrustStore</a> and <a href="#">TrustStorePassword</a> properties or by using Java system properties.
Default	true
Data Type	boolean

## WithHoldCursors

Description	Determines whether the cursor stays open on commit—either DB2 leaves all cursors open (Preserve cursors) or closes all open cursors (Delete cursors) after a commit. Rolling back a transaction closes all cursors regardless of how this property is specified.
-------------	--

Valid Values true | false

If set to true, the cursor behavior is Preserve.

If set to false, the cursor behavior is Delete.

Default true

Data Type boolean

## XMLDescribeType

Description Determines whether the driver maps XML data to the CLOB or BLOB data type.

Valid Values clob | blob

If set to clob, the driver maps XML data to the CLOB data type.

If set to blob, the driver maps XML data to the BLOB data type.

Default None

Data Type String

See Also ["Returning and Inserting/Updating XML Data" on page 142](#)

---

# Performance Considerations

You can optimize your application's performance if you set the DB2 driver connection properties as described in this section:

**CatalogIncludesSynonyms:** The `DatabaseMetaData.getColumns()` method is often used to determine characteristics about a table, including the synonym, or alias, associated with a table. If your application accesses DB2 v7.x for Linux/UNIX/Windows, DB2 for z/OS, or DB2 for iSeries and your application does not use database table synonyms, the driver can improve performance by ignoring this information. By default, the driver returns synonyms for the `DatabaseMetaData.getColumns()` method when accessing DB2 v8.x and higher for Linux/UNIX/Windows.

**CatalogOptions:** Retrieving synonym information is expensive. If your application does not need to return this information, the driver can improve performance. Default driver behavior is to include synonyms in the result set of calls to the following `DatabaseMetaData` methods: `getColumns()`, `getExportedKeys()`, `getFunctionColumns()`, `getFunctions()`, `getImportedKeys()`, `getIndexInfo()`, `getPrimaryKeys()`, `getProcedureColumns()`, and `getProcedures()`. If your application needs to return synonyms for `getColumns()` calls, the driver can emulate `getColumns()` calls using the `ResultSetMetaData` object instead of querying database catalogs for the column information. Using emulation can improve performance because the SQL statement formulated by the emulation is less complex than the SQL statement formulated using `getColumns()`.

**CatalogSchema:** To improve performance, views of system catalog tables can be created in a catalog schema other than the default. The DB2 driver can access the views of catalog tables if this property is set to the name of the schema containing the views. The default catalog schema is SYSCAT for DB2 for

Linux/UNIX/Windows, SYSIBM for DB2 for z/OS, and QSYS2 for DB2 for iSeries.

To ensure that catalog methods function correctly, views for specific catalog tables must exist in the specified schema. The views that are required depend on your DB2 database. See ["Non-Default Schemas for Catalog Methods" on page 157](#) for views for catalog tables that must exist in the specified schema.

**EnableBulkLoad:** For batch inserts, the driver can use native bulk load protocols instead of the batch mechanism. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations. Set this property to true to allow existing applications with batch inserts to take advantage of bulk load without requiring changes to the code.

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

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

**MaxPooledStatements:** To improve performance, the driver's own internal prepared statement pooling should be enabled when the driver does not run from within an application server or from within another application that 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.

See “[Managing Connections](#)” on page 711 for more information about using prepared statement pooling to optimize performance.

**SendStreamAsBlob:** If the large binary objects you insert or update are stored as Blobs, performance can be improved by sending the binary stream as Blob data. In this case, this property should be set to true.

**StripNewLines:** If you know that the SQL statements used in your application do not contain newline characters, the driver can improve performance by omitting the parsing required to remove them.

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

---

## DB2 Packages

The DB2 driver automatically creates all required DB2 packages the first time it connects to the database. By default, the DB2 packages contain 200 dynamic sections and are created in the NULLID collection (or library). You can override the default number of dynamic sections by setting the `DynamicSections` property. Similarly, you can override the collection in which the packages are created by setting the `PackageCollection` property.

NOTES: The initial connection may take a few minutes because of the number and size of the packages that must be created for the connection. Subsequent connections do not incur this delay. When the driver has completed creating packages, it writes the following message to the standard output: DB2 packages created.

In most cases, you do not need to create DB2 packages because the DB2 driver automatically creates them the first time it connects to the database. If, for some reason, you need to explicitly create them, you can create them using any of the following methods:

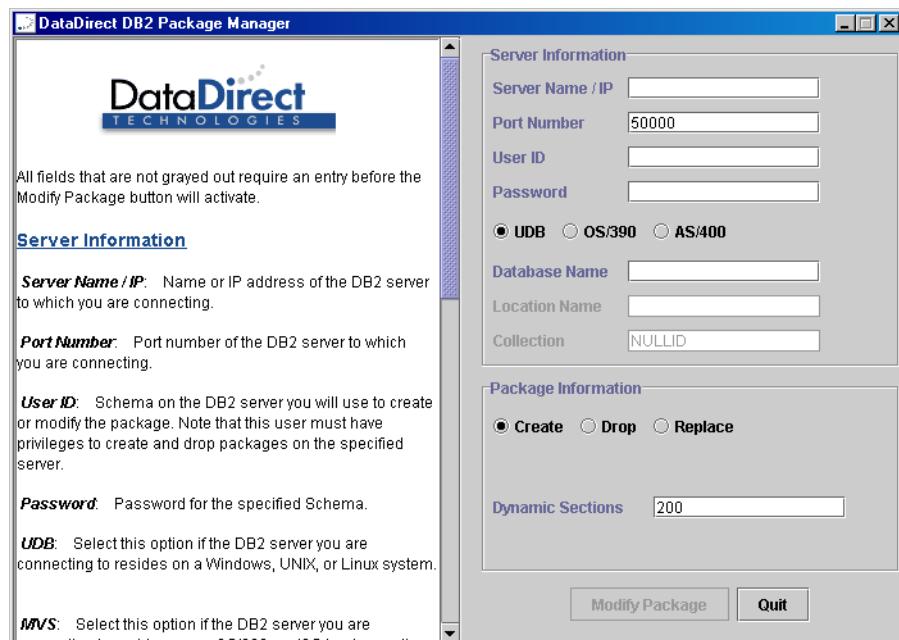
- Using the DataDirect DB2 Package Manager, a Java graphical tool that makes it easy for you to create, drop, and replace DB2 packages. See “[Creating DB2 Packages Using the Package Manager](#)” on page 135 for instructions.
- Using DB2 driver connection properties specified in a connection URL or data source. See “[Creating DB2 Packages Using Connection Properties](#)” on page 136 for instructions.
- Running the appropriate DB2 package creation list file for your operating system on the database server. See “[Creating DB2 Packages Using Package Creation List Files](#)” on page 137 for instructions.

- On z/OS and iSeries only, copy the packages that are installed with the driver as described in “[Copying the DB2 Packages \(z/OS and iSeries\)](#)” on page 139.

## Creating DB2 Packages Using the Package Manager

**NOTE:** The user ID creating the DB2 packages must have BINDADD privileges on the database. Consult with your database administrator to ensure that you have the correct privileges.

To start the DataDirect DB2 Package Manager, run the DB2PackageManager.bat file (on Windows) or the DB2PackageManager.sh script (on UNIX) located in the lib subdirectory in the DataDirect Connect *for JDBC* installation directory. The DataDirect DB2 Package Manager dialog box appears.



Complete the fields in the right pane. Refer to the left pane of the DataDirect DB2 Package Manager for instructions on completing these fields. When you are satisfied with your choices, click the **Modify Package** button to create the package on the DB2 server.

## Creating DB2 Packages Using Connection Properties

NOTE: The user ID creating the DB2 packages must have BINDADD privileges on the database. Consult with your database administrator to ensure that you have the correct privileges.

[Table 3-1](#) lists the connection properties you can use to explicitly create DB2 packages.

---

**Table 3-1. Connection Properties for Creating DB2 Packages**

---

Property	Database
PackageCollection= <i>collection_name</i> (where <i>collection_name</i> is the name of the collection or library to which DB2 packages are bound)	DB2 for z/OS and iSeries
CreateDefaultPackage=true	DB2 for Linux/UNIX/Windows, z/OS, and iSeries
ReplacePackage=true	DB2 for Linux/UNIX/Windows
DynamicSections=x (where x is a positive integer)	DB2 for Linux/UNIX/Windows, z/OS, and iSeries

---

NOTE: To create new DB2 packages on DB2 for Linux/UNIX/Windows, you must use ReplacePackage=true in conjunction with CreateDefaultPackage=true. If a DB2 package already exists, it will be replaced when ReplacePackage=true.

### **Example for DB2 for Linux/UNIX/Windows:**

The following URL creates DB2 packages with 400 dynamic sections. If any DB2 packages already exist, they will be replaced by the new ones being created.

```
jdbc:datadirect:db2://server1:50000;DatabaseName=SAMPLE;
CreateDefaultPackage=TRUE;ReplacePackage=TRUE;DynamicSections=400
```

### **Example for DB2 for z/OS and iSeries:**

The following URL creates DB2 packages with 400 dynamic sections.

```
jdbc:datadirect:db2://server1:446;LocationName=SAMPLE;
CreateDefaultPackage=TRUE;DynamicSections=400
```

## **Creating DB2 Packages Using Package Creation List Files**

**NOTE:** The user ID creating the DB2 packages must have BINDADD privileges on the database. Consult with your database administrator to ensure that you have the correct privileges.

To explicitly create DB2 packages, bind the package creation list files for your operating system on your database server:

<b>File</b>	<b>Location</b>
DDJDBC_LUW.lst	<i>installdir</i> /DB2/bind/LUW
DDJDBC_400.lst	<i>installdir</i> /DB2/bind/iSeries
DDJDBC_MVS.lst	<i>installdir</i> /DB2/bind/zOS

When you bind the list files, if any DB2 packages exist, they will be replaced by the new packages. The list files create DB2 packages that contain 200 dynamic sections and are created in the NULLID collection.

**To explicitly create DB2 packages:**

- 1 Copy the appropriate list (\*.lst) and bind (\*.bnd) files to a directory on the database server.
- 2 From the directory on the database server where you placed the list and bind files, start the DB2 command-line utility. Use the utility to connect to the database where you want to bind the packages. Connect using the following command:

`connect to database_name user authorization_name using  
password`

where:

*database\_name* is the name of the database to which you are connecting.

*authorization\_name* is the name of the user you are authenticating to the server.

*password* is the user's password.

- 3 Execute the DB2 bind command:

`bind @list_file grant public`

where *list\_file* is the name of the list file you want to bind.

## Copying the DB2 Packages (z/OS and iSeries)

NOTE: The user ID creating the DB2 packages must have BINDADD privileges on the database. Consult with your database administrator to ensure that you have the correct privileges.

To copy the DB2 packages on z/OS and iSeries operating systems, refer to the instruction file for your operating system:

File	Location
CfJDBC zOS Manual Package Creation Instructions.txt	<i>installdir\DB2\bind\zOS</i>
CfJDBC AS400 Manual Package Creation Instructions.txt	<i>installdir\DB2\bind\iSeries</i>

The DB2 packages contain 200 dynamic sections.

## Data Types

[Table 3-2](#) lists the data types supported by the DB2 driver and describes how they are mapped to JDBC data types.

**Table 3-2. DB2 Data Types**

DB2 Data Type	JDBC Data Type
Bigint <sup>1</sup>	BIGINT
Binary <sup>1</sup>	BINARY
Blob <sup>2</sup>	BLOB
Char	CHAR
Char for Bit Data	BINARY

**Table 3-2. DB2 Data Types (cont.)**

<b>DB2 Data Type</b>	<b>JDBC Data Type</b>
Clob	CLOB
Date	DATE
DBClob <sup>2</sup>	CLOB  NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NCLOB (if using Java SE 6) or CLOB (if using another JVM).
Decfloat <sup>3</sup>	DECIMAL
Decimal	DECIMAL
Double	DOUBLE
Float	FLOAT
Graphic	CHAR  NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NCHAR (if using Java SE 6) or CHAR (if using another JVM).
Integer	INTEGER
Long Varchar	LONGVARCHAR
Long Varchar for Bit Data	LONGVARBINARY
Long Vargraphic	LONGVARCHAR  NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: LONGNVARCHAR (if using Java SE 6) or LONGVARCHAR (if using another JVM).
Numeric	NUMERIC
Real	REAL
Rowid <sup>4</sup>	VARBINARY
Smallint	SMALLINT
Time	TIME

**Table 3-2. DB2 Data Types (cont.)**

<b>DB2 Data Type</b>	<b>JDBC Data Type</b>
Timestamp	TIMESTAMP
Varbinary <sup>1</sup>	VARBINARY
Varchar	VARCHAR
Varchar for Bit Data	VARBINARY
Vargraphic	VARCHAR NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NVARCHAR (if using Java SE 6) or VARCHAR (if using another JVM).
XML <sup>5 6</sup>	CLOB NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: SQLXML (if using Java SE 6) or CLOB (if using another JVM).

1. Supported only for DB2 v9.1 for z/OS.  
 2. Supported only for DB2 v8.1 and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  
 3. Supported only for DB2 V9.5 and V9.7 for Linux/UNIX/Windows, DB2 V9.1 for z/OS and DB2 V6R1 for iSeries.  
 4. Supported only for DB2 for z/OS and DB2 V5R2 and higher for iSeries.  
 5. Supported only for DB2 V9.1 and higher for Linux/UNIX/Windows.  
 6. The XMLDescribeType property overrides the mappings for XML data.

See “[Large Object \(LOB\) Support](#)” on page 163 for more information about the Blob, Clob, and DBClob data types. See “[Returning and Inserting/Updating XML Data](#)” on page 142 for more information about the XML data type. See [Appendix D “getTypeInfo” on page 631](#) for a description of the data types returned by the `getTypeInfo()` method.

---

## Returning and Inserting/Updating XML Data

For DB2 v9.1 and higher for Linux/UNIX/Windows, the DB2 driver supports the XML data type. Which JDBC data type the XML data type is mapped to depends on whether the JDBCBehavior and XMLDescribeType properties are set:

- If XMLDescribeType=clob or XMLDescribeType=blob, the driver maps the XML data type to the JDBC CLOB or BLOB data type, respectively, regardless of the setting of the JDBCBehavior property.
- If JDBCBehavior=1 (the default) and the XMLDescribeType property is not set, the driver maps XML data to the JDBC CLOB data type.
- If JDBCBehavior=0 and the XMLDescribeType property is not set, XML data is mapped to SQLXML or CLOB, depending on which JVM your application is using. The driver maps the XML data type to the JDBC SQLXML data type if your application is using Java SE 6. If your application is using another JVM, the driver maps the XML data type to the JDBC CLOB data type.

## Returning XML Data

You can specify whether XML data is returned as character or binary data by setting the XMLDescribeType property. For example, consider a database table defined as:

```
CREATE TABLE xmlTable (id int, xmlCol xml NOT NULL)
```

and the following code:

```
String sql="SELECT xmlCol FROM xmlTable";
ResultSet rs=stmt.executeQuery(sql);
```

If your application uses the following connection URL, which specifies that the XML data type be mapped to the BLOB data type, the driver would return XML data as binary data:

```
jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc;User=test;  
Password=secret;XMLDescribeType=blob
```

## Character Data

When XMLDescribeType=clob, XML data is returned as character data. The result set column is described with a column type of CLOB and the column type name is xml.

When XMLDescribeType=clob, your application can use the following methods to return data stored in XML columns as character data:

```
ResultSet.getString()  
ResultSet.getCharacterStream()  
ResultSet.getBlob()  
CallableStatement.getString()  
CallableStatement.getBlob()
```

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.

When `XMLDescribeType=blob`, your application should not use any of the methods for returning character data described in this section. In this case, the driver applies the standard JDBC character-to-binary conversion to the data, which returns the hexadecimal representation of the character data.

## ***Binary Data***

When `XMLDescribeType=blob`, the driver returns XML data as binary data. The result set column is described with a column type of `BLOB` and the column type name is `xml`.

When `XMLDescribeType=blob`, your application can use the following methods to return XML data as binary data:

```
ResultSet.getBytes()  
ResultSet.getBinaryStream()  
ResultSet.getBlob()  
ResultSet.getObject()  
CallableStatement.getBytes()  
CallableStatement.getBlob()  
CallableStatement.getObject()
```

The driver does not apply any data conversions to the XML data returned from the database server. These methods return a byte array or binary stream that contains the XML data encoded as UTF-8.

When `XMLDescribeType=clob`, your application should not use any of the methods for returning binary data described in this section. In this case, the driver applies the standard JDBC binary-to-character conversion to the data, which returns the hexadecimal representation of the binary data.

# Inserting/Updating XML Data

The driver can insert or update XML data as character or binary data regardless of the setting of the XMLDescribeType connection property.

## ***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()  
ResultSet.updateObject()
```

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

---

## **Authentication**

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See “[Authentication](#)” on page 67 for an overview.

The DB2 driver supports the following methods of authentication:

- User ID/password authentication authenticates the user to the database using a database user name and password. Depending on the method you specify, the driver passes one of the following sets of credentials to the DB2 database server for authentication:
  - Encrypted user ID and password
  - User ID in clear text and an encrypted password
  - Both user ID and password in clear text

- 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 and MIT Kerberos.

- Client authentication uses the user ID of the user logged onto the system on which the driver is running to authenticate the user to the database. The DB2 database server relies on the client to authenticate the user and does not provide additional authentication.

NOTE: Because the database server does not authenticate the user when client authentication is used, use this method of authentication if you can guarantee that only trusted clients can access the database server.

The driver's AuthenticationMethod connection property controls which authentication mechanism the driver uses when establishing connections. See "[Using the AuthenticationMethod Property](#)" on page 147 for information about setting the value for this property.

## Using the AuthenticationMethod Property

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

When AuthenticationMethod=kerberos, the driver uses Kerberos authentication when establishing a connection. The

driver ignores any values specified by the User property and Password properties.

When AuthenticationMethod=encryptedUIDPassword, AuthenticationMethod=encryptedPassword, or AuthenticationMethod=clearText (the default), the driver uses user ID/password authentication when establishing a connection. The User property provides the user ID. The Password property provides the password. The set of credentials that are passed to the DB2 server depend on the specified value:

- When AuthenticationMethod=encryptedUIDPassword, an encrypted user ID and encrypted password are sent to the DB2 server for authentication.
- When AuthenticationMethod=encryptedPassword, a user ID in clear text and an encrypted password are sent to the DB2 server for authentication.
- When AuthenticationMethod=clearText, both a user ID and a password are sent in clear text to the DB2 server for authentication.

If any of these values are set, the driver also can use data encryption by setting the EncryptionMethod property.

When AuthenticationMethod=client, the driver uses the user ID of the user logged onto the system on which the driver is running when establishing a connection. The DB2 database server relies on the client to authenticate the user and does not provide additional authentication. The driver ignores any values specified by the User property and Password properties.

## Configuring User ID/Password Authentication

- 1 Set the AuthenticationMethod property to encryptedUIDPassword, encryptedPassword, or clearText (the default). See “[Using the AuthenticationMethod Property](#)” on page 147 for more information about setting a value for this property.
- 2 Set the User property to provide the user ID.
- 3 Set the Password property to provide the password.

## Configuring Kerberos Authentication

This section provides requirements and instructions for configuring Kerberos authentication for the DB2 driver.

### ***Product Requirements***

Verify that your environment meets the requirements listed in [Table 3-3](#) before you configure the driver for Kerberos authentication.

---

***Table 3-3. Kerberos Authentication Requirements for the DB2 Driver***

---

Component	Requirements
Database server	The database server must be running one of the following database versions: <ul style="list-style-type: none"> <li>■ DB2 v8.1 or higher for Linux/UNIX/Windows</li> <li>■ DB2 v7.x or higher for z/OS</li> </ul>

**Table 3-3. Kerberos Authentication Requirements for the DB2 Driver (cont.)**

<b>Component</b>	<b>Requirements</b>
Kerberos server	<p>The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos KDC.</p> <p>Network authentication must be provided by one of the following methods:</p> <ul style="list-style-type: none"> <li>■ Windows Active Directory on one of the following operating systems:           <ul style="list-style-type: none"> <li>• Windows Server 2003</li> <li>• Windows 2000 Server Service Pack 3 or higher</li> </ul> </li> <li>■ MIT Kerberos 1.4.2 or higher</li> </ul>
Client	J2SE 1.4.2 or higher must be installed.

## ***Configuring the Driver***

During installation, DataDirect Connect for JDBC installs the following files required for Kerberos authentication in the /lib subdirectory of your DataDirect Connect for JDBC installation directory:

- krb5.conf is a Kerberos configuration file containing values for the Kerberos realm and the KDC name for that realm. DataDirect Connect for JDBC installs a generic file that you must modify for your environment.
- JDBCDriverLogin.conf file is a configuration file that specifies which Java Authentication and Authorization Service (JAAS) login module to use for Kerberos authentication. This file is configured to load automatically unless the java.security.auth.login.config system property is set to load another configuration file. You can modify this file, but the driver must be able to find the JDBC\_DRIVER\_01 entry in this file or another specified login configuration file to configure

the JAAS login module. Refer to your J2SE documentation for information about setting configuration options in this file

**To configure the driver:**

- 1 Set the AuthenticationMethod property to kerberos. See ["Using the AuthenticationMethod Property" on page 147](#) for more information about setting a value for this property.
- 2 Modify the krb5.conf file to contain your Kerberos realm name and the KDC name for that Kerberos realm by editing the file with a text editor or by specifying the system properties, java.security.krb5.realm and java.security.krb5.kdc.

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

For example, if your Kerberos realm name is XYZ.COM and your KDC name is kdc1, your krb5.conf file would look like this:

```
[libdefaults]
    default_realm = XYZ.COM

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

If the krb5.conf file does not contain a valid Kerberos realm and KDC name, the following exception is thrown:

```
Message:[DataDirect][DB2 JDBC Driver]Could not
establish a connection using integrated security: No
valid credentials provided
```

The krb5.conf file installed by DataDirect Connect for JDBC is configured to load automatically unless the

- java.security.krb5.conf system property is set to point to another Kerberos configuration file.
- 3 If using Kerberos authentication with a Security Manager on a Java 2 Platform, you must grant security permissions to the application and driver. See “[Permissions for Kerberos Authentication](#)” on page 79 for an example.

## Specifying User Credentials for Kerberos Authentication (Delegation of Credentials)

By default, when Kerberos authentication is used, the DB2 driver takes advantage of the user name and password maintained by the operating system to authenticate users to the database. By allowing the database to share the user name and password used for the operating system, users with a valid operating system account can log into the database without supplying a user name and password.

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

```
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import java.sql.*;

// The following code creates a javax.security.auth.Subject instance
// used for authentication. Refer to the Java Authentication
// and Authorization Service documentation for details on using a
// LoginContext to obtain a Subject.
```

```
LoginContext lc = null;
Subject subject = null;

try {

    lc = new LoginContext("JaasSample", new TextCallbackHandler());
    lc.login();
    subject = lc.getSubject();
}
catch (Exception le) {
    ... // display login error
}

// This application passes the javax.security.auth.Subject
// to the driver by executing the driver code as the subject

Connection con =
    (Connection) Subject.doAs(subject, new PrivilegedExceptionAction() {

        public Object run() {

            Connection con = null;
            try {

                Class.forName("com.ddtek.jdbc.db2.DB2Driver");
                String url = "jdbc:datadirect:db2://myServer:50000;
                    DatabaseName=jdbc";
                con = DriverManager.getConnection(url);
            }
            catch (Exception except) {

                ... //log the connection error
                Return null;
            }

            return con;
        }
    });
});
```

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

## Obtaining a Kerberos Ticket Granting Ticket

To use Kerberos authentication, the application user first must obtain a Kerberos Ticket Granting Ticket (TGT) from the Kerberos server. The Kerberos server verifies the identity of the user and controls access to services using the credentials contained in the TGT.

If the application uses Kerberos authentication from a Windows client, the application user does not need to explicitly obtain a TGT. Windows Active Directory automatically obtains a TGT for the user.

If the application uses Kerberos authentication from a UNIX or Linux client, the user must explicitly obtain a TGT. To explicitly obtain a TGT, the user must log onto the Kerberos server using the kinit command. For example, the following command requests a TGT from the server with a lifetime of 10 hours, which is renewable for 5 days:

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

where *user* is the application user.

Refer to your Kerberos documentation for more information about using the kinit command and obtaining TGTs for users.

## Configuring Client Authentication

Set the AuthenticationMethod property to client. See “[Using the AuthenticationMethod Property](#)” on page 147 for more information about setting a value for this property.

---

## Data Encryption

The DB2 driver supports database-specific encryption (DB2’s own encryption protocol) for the following databases:

- DB2 for Linux/UNIX/Windows
- DB2 for z/OS

The DB2 driver supports SSL encryption for the following databases:

- DB2 V9.1 Fixpack 2 and higher for Linux/UNIX/Windows
- DB2 v9.1 for z/OS
- DB2 V5R3 and higher for iSeries

See “[Data Encryption Across the Network](#)” on page 69 for more information.

## Configuring DB2-Specific Encryption

- 1 Set the AuthenticationMethod property to clearText, encryptedPassword, or encryptedUIDPassword.
- 2 Set the EncryptionMethod property to DBEncryption or RequestDBEncryption.

## Configuring 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:

- 1 Set the EncryptionMethod property to SSL.
- 2 Specify the location and password of the truststore file used for SSL server authentication. Either set the TrustStore and TrustStore properties or their corresponding Java system properties (javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword, respectively).
- 3 To validate certificates sent by the database server, set the ValidateServerCertificate property to true.
- 4 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.
- 5 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 KeyStore 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.

---

# Non-Default Schemas for Catalog Methods

To ensure that catalog methods function correctly when the CatalogSchema property is set to a schema other than the default schema, views for the catalog tables listed in [Table 3-4](#) must exist in the specified schema. The views that are required depend on your DB2 database.

---

**Table 3-4. Catalog Tables for DB2**

---

Database	Catalog Tables	
DB2 for Linux/UNIX/Windows	SYSCAT.TABLES SYSCAT.COLUMNS SYSCAT.PROCEDURES SYSCAT.PROCPARAMS SYSCAT.COLAUTH SYSCAT.TABAUTH SYSCAT.KEYCOLUSE	SYSCAT.INDEXES SYSCAT.INDEXCOLUSE SYSCAT.REFERENCES SYSCAT.SYSSCHEMATA SYSCAT.TYPEMAPPINGS SYSCAT.DBAUTH
DB2 for z/OS	SYSIBM.SYSTABCONST SYSIBM.SYSTABLES SYSIBM.SYSSYNONYMS SYSIBM.SYSCOLUMNS SYSIBM.SYSPROCEDURES SYSIBM.SYSROUTINES SYSIBM.SYSPARMS SYSIBM.SYSCOLAUTH	SYSIBM.SYSTABAUTH SYSIBM.SYSKEYS SYSIBM.SYSINDEXES SYSIBM.SYSRELS SYSIBM.SYSFOREIGNKEYS SYSIBM.SYSSCHEMAAUTH SYSIBM.SYSDBAUTH
DB2 for iSeries	QSYS2.SYSCST QSYS2.SYSKEYCST QSYS2.SYSPROCS QSYS2.SYSPARMS QSYS2.SYSTABLES QSYS2.SYSSYNONYMS	QSYS2.SYSCOLUMNS QSYS2.SQLTABLEPRIVILEGES QSYS2.SYSKEYS QSYS2.SYSINDEXES QSYS2.SYSREFCST

---

---

## Reauthentication

The DB2 driver supports reauthentication for the following databases:

- DB2 V9.1 and higher for Linux/UNIX/Windows. The user performing the switch must have been granted the database SETSESSIONUSER permission.
- DB2 v8.1.4 and higher for Linux/UNIX/Windows. The user performing the switch must have been granted the SYSADM permission.

See ["Using Reauthentication" on page 55](#) for an introduction to reauthentication. See [Appendix I "Connection Pool Manager" on page 789](#) for information about using reauthentication with the DataDirect Connection Pool Manager.

**NOTE:** Before performing reauthentication, applications must ensure that any statements or result sets created as one user are closed before switching the connection to another user.

Your application can use the setCurrentUser() method in the ExtConnection interface to switch a user on a connection.

The `setCurrentUser()` method accepts driver-specific reauthentication options. The options supported for the DB2 driver are:

<code>CURRENT_SCHEMA</code>	Specifies the name of the current schema. The value must be a valid DB2 schema name. If the <code>setCurrentUser()</code> method is called and this option is not specified or the value is set to <code>#USER#</code> , the schema is switched to the schema of the current user. If the <code>setCurrentUser()</code> method is called and this option is specified as an empty string, only the user is switched; the schema is not switched.
<code>CURRENT_PATH</code>	Specifies the current path for the database to use when locating stored procedures and functions. The value must be a valid path name for the DB2 CURRENT PATH special register. If the <code>setCurrentUser()</code> method is called and this option is not specified or the value is set to <code>#USER#</code> , the path is switched to the path of the current user. If the <code>setCurrentUser()</code> method is called and this option is specified as an empty string, only the user is switched; the path is not switched.

See “[ExtConnection Interface](#)” on page 613 for more information about the `setCurrentUser()` method.

---

## Client Information for Connections

The DB2 driver allows applications to store and return the following types of client information associated with a particular connection:

- Name of the application
- User ID
- Host name of the client
- Additional accounting information, such as an accounting ID
- Product name and version of the DB2 driver

This information can feed directly into the Workload Manager (WLM) for workload management and monitoring purposes. See ["Workload Manager \(WLM\)"](#) for more information.

See [Appendix C "Client Information for Connections" on page 621](#) for information about storing and returning client information.

---

## Workload Manager (WLM)

The Workload Manager (WLM) is a priority and resource manager within DB2 V9.5 and V9.7 for Linux/UNIX/Windows. On z/OS, the WLM is part of the operating system. WLM prioritizes and matches DB2 workloads with available resources. *DB2 workloads* allow you to categorize similar types of work. For example, a database administrator can create a DB2 workload named Sales to service all connections that come from Sales applications.

The WLM automatically adjusts server resources, such as CPU and memory, based on the service class associated with a DB2 workload. Therefore, an application's performance is tied to the DB2 workload it is assigned to and, ultimately, to the service class

associated with that workload. For example, an application that performs batch work nightly when resource usage is low can use the default workload. In contrast, sales updates that need to be processed quickly twice a day need to use a workload that is governed by a high priority service class.

It is important to understand that, unless specified otherwise, all work will run in the default workload that is governed by the default service class. To ensure the best performance, consult with your database administrator to verify that your application is associated with the appropriate DB2 workload and service class.

In addition to workload management, WLM also provides monitoring functionality that is useful for troubleshooting. For example, the database administrator can set threshold limits to detect long-running queries and gather information about those queries.

The DB2 driver allows your application to set client information in the DB2 database that can be used by the WLM to classify work. If you know that your database environment uses WLM, coordinate with your database administrator to determine which attributes you need to set.

See [Appendix C “Client Information for Connections” on page 621](#) for more information about using the WLM with the DB2 driver.

---

## SQL Escape Sequences

See [Appendix F “SQL Escape Sequences for JDBC” on page 717](#) for information about SQL escape sequences supported by the DB2 driver.

---

## Isolation Levels

The DB2 driver supports the isolation levels listed in [Table 3-5](#). JDBC isolation levels are mapped to the appropriate DB2 transaction isolation levels as shown. The default isolation level is Read Committed.

---

***Table 3-5. Supported Isolation Levels***

---

JDBC Isolation Level	DB2 Isolation Level
None	No Commit <sup>1</sup>
Read Committed (default)	Cursor Stability
Read UnCommitted	Uncommitted Read
Repeatable Read	Read Stability
Serializable	Repeatable Read

---

1. Supported for DB2 for iSeries versions that do not enable journaling.

---

## Using Scrollable Cursors

The DB2 driver supports scroll-insensitive result sets and updatable result sets.

NOTE: When the DB2 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

To use JDBC distributed transactions through JTA with the DB2 driver, you must use one of the following databases:

- DB2 v8.x and higher for Linux/UNIX/Windows
- DB2 v9.1 for z/OS
- DB2 V5R4 and higher for iSeries

---

## Large Object (LOB) Support

Retrieving and updating Blobs, Clobss, and DBClobss is supported by the DB2 driver with the following databases:

- DB2 v8.x and higher for Linux/UNIX/Windows
- DB2 for z/OS
- DB2 V5R2 and higher for iSeries

The DB2 driver supports Clobss up to a maximum of 2 GB with these databases. The DB2 driver supports retrieving and updating Clobss up to a maximum of 32 KB with all other supported DB2 databases.

---

## Batch Inserts and Updates

The DB2 driver uses the native DB2 batch mechanism. By default, the methods used to set the parameter values of a batch performed using a PreparedStatement must match the database data type of the column with which the parameter is associated.

DB2 servers do not perform implicit data conversions, so specifying parameter values that do not match the column data

type causes the DB2 server to generate an error. For example, to set the value of a Blob parameter using a stream or byte array when the length of the stream or array is less than 32 KB, you must use the `setObject()` method and specify the target JDBC type as `BLOB`; you cannot use the `setBinaryStream()` or `setBytes()` methods.

To remove the method-type restriction, set the `BatchPerformanceWorkaround` property to true. For example, you can use the `setBinaryStream()` or `setBytes()` methods to set the value of a Blob parameter regardless of the length of the stream or array; however, the parameter sets may not be executed in the order they were specified. Performance may be decreased because the driver must convert the parameter data to the correct data type and re-execute the statement.

---

## Parameter Metadata Support

The DB2 driver supports returning parameter metadata as described in this section.

## Insert and Update Statements

The DB2 driver supports returning parameter metadata for all types of SQL statements with the following databases:

- DB2 v8.x and higher for Linux/UNIX/Windows
- DB2 for z/OS
- DB2 V5R2 and higher for iSeries

For DB2 v7x for Linux/UNIX/Windows and DB2 V5R1 for iSeries, the DB2 driver supports returning parameter metadata for the following forms of Insert and Update statements:

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

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

## Select Statements

The DB2 driver supports returning parameter metadata for all types of SQL statements with the following DB2 databases:

- DB2 v8.x and higher for Linux/UNIX/Windows
- DB2 for z/OS
- DB2 V5R2 and higher for iSeries

For DB2 v7x for Linux/UNIX/Windows and DB2 V5R1 for iSeries, the DB2 driver supports returning parameter metadata for Select statements that contain parameters in ANSI SQL 92 entry-level predicates, for example, such as COMPARISON, BETWEEN, IN, LIKE, and EXISTS predicate constructs. Refer to the ANSI SQL reference for detailed syntax.

Parameter metadata can be returned for a Select statement if one of the following conditions is true:

- The statement contains a predicate value expression that can be targeted against the source tables in the associated FROM clause. For example:

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

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

- The statement contains a predicate value expression part that is a nested query. The nested query's metadata must describe a single column. For example:

```
SELECT * FROM foo WHERE (SELECT x FROM y  
WHERE z = 1) < ?
```

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

```
SELECT col1, col2 FROM foo WHERE col1 = ? and col2 > ?
```

```
SELECT ... WHERE colname = (SELECT col2 FROM t2  
WHERE col3 = ?)
```

```
SELECT ... WHERE colname LIKE ?
```

```
SELECT ... WHERE colname BETWEEN ? and ?
```

```
SELECT ... WHERE colname IN (?, ?, ?)
```

```
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE col1 < ?)
```

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

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

Joins are supported. For example:

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

Fully qualified names and aliases are supported. For example:

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

## Stored Procedures

The DB2 driver supports returning parameter metadata for stored procedure arguments.

---

## ResultSet Metadata Support

If your application requires table name information, the DB2 driver can return table name information in ResultSet metadata for Select statements. If you set the `ResultSetMetaDataOptions` property to 1, the DB2 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.

The table name information that is returned by the DB2 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 DB2 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 DB2 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 DB2 driver also can return schema name and catalog name information when the `ResultSetMetaData.getSchemaName()` and `ResultSetMetaData.getCatalogName()` methods are called if the driver can determine that information. For example, for the following statement, the DB2 driver returns "test" for the catalog name, "test1" for the schema name, and "foo" for the table name:

```
SELECT * FROM test.test1.foo
```

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

---

## Rowset Support

The DB2 driver supports any JSR 114 implementation of the RowSet interface, including:

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

See <http://www.jcp.org/en/jsr/detail?id=114> for more information about JSR 114.

---

## Auto-Generated Keys Support

The DB2 driver supports retrieving the values of auto-generated keys. An auto-generated key returned by the DB2 driver is the value of an auto-increment 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 that contains parameters:

- When using an Insert statement that does not contain any parameters, the DB2 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 that contains parameters, the DB2 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)`

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.

See ["Retrieving Auto-Generated Keys" on page 709](#) for information about how auto-generated keys can improve performance.

---

# Configuring Failover

1 Specify the primary and alternate servers:

- Specify your primary server using a connection URL or data source.
- Specify one or multiple alternate servers by setting the AlternateServers property.

See “[Specifying Primary and Alternate Servers](#)” on page 172.

NOTE: If using failover with DB2 High Availability Disaster Recovery (HADR), the primary server must be the primary server configured in your HADR system and any alternate server must be a standby server configured in your HADR system.

- 2 Choose a failover method by setting the FailoverMode connection property. The default method is connection failover (FailoverMode=connect). See “[Using Failover](#)” on page 58 for an overview of each failover method.
- 3 If FailoverMode=extended or FailoverMode=select, set the FailoverGranularity property to specify how you want the driver to behave if exceptions occur while trying to reestablish a lost connection. The default behavior of the driver is to continue with the failover process and post any exceptions on the statement on which they occur (FailoverGranularity=nonAtomic).
- 4 Optionally, configure the connection retry feature. See “[Specifying Connection Retry](#)” on page 174.
- 5 Optionally, set the FailoverPreconnect property if you want the driver to establish a connection with the primary and an alternate server at the same time. The default behavior is to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection (FailoverPreconnect=false).

## Specifying Primary and Alternate Servers

Connection information for primary and alternate servers can be specified using either one of the following methods:

- Connection URL through the JDBC Driver Manager
- JDBC data source

For example, the following connection URL for the DB2 driver specifies connection information for primary and alternate servers using a connection URL:

```
jdbc:datadirect:db2://server1:50000;DatabaseName=TEST;User=test;  
Password=secret;AlternateServers=(server2:50000;DatabaseName=TEST2,  
server3:50000;DatabaseName=TEST3)
```

In this example:

```
...server1:50000;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=(server2:50000;DatabaseName=TEST2,  
server3:50000;DatabaseName=TEST3)
```

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

```
DB2DataSource mds = new DB2DataSource();  
mds.setDescription("My DB2DataSource");  
mds.setServerName("server1");  
mds.setPortNumber(50000);  
mds.setDatabaseName("TEST");  
mds.setUser("test");  
mds.setPassword("secret");  
mds.setAlternateServers(server2:50000;DatabaseName=TEST2,  
server3:50000;DatabaseName=TEST3)
```

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 50000 is used.

*property=value* is one of the following connection properties: `DatabaseName` or `LocationName`. These connection properties are optional for each alternate server entry. For example:

```
jdbc:datadirect:db2://server1:50000;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:50000;DatabaseName=TEST2,
server3:50000;DatabaseName=TEST2)
```

or

```
jdbc:datadirect:db2://server1:50000;LocationName=TEST;User=test;
Password=secret;AlternateServers=(server2:50000;LocationName=TEST2,
server3:50000;LocationName=TEST3)
```

If you do not specify an optional 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 uses the database name specified for the primary server and tries to connect to the TEST database on the alternate server:

```
jdbc:datadirect:db2://server1:50000;DatabaseName=TEST;User=test;  
Password=secret;AlternateServers=(server2:50000,server3:50000)
```

## Specifying Connection Retry

Connection retry allows the DB2 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:db2://server1:50000;DatabaseName=TEST;User=test;  
Password=secret;AlternateServers=(server2:50000;DatabaseName=TEST2,  
server3:50000;DatabaseName=TEST3);ConnectionRetryCount=2;  
ConnectionRetryDelay=5
```

In this example, if a successful connection is not established on the DB2 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.

## Failover Properties

[Table 3-6](#) summarizes the connection properties that control how failover works with the DB2 driver. See [“Connection Properties” on page 88](#) for details about configuring each property.

**Table 3-6. Summary: Failover Properties for the DB2 Driver**

Property	Characteristic
AlternateServers	One or multiple alternate database servers. An IP address or server name identifying each server is required. Port number and supported connection properties (DatabaseName or LocationName) are optional. If the port number is unspecified, the port specified for the primary server is used. If a port number is not specified for the primary server, a default port number of 50000 is used.
ConnectionRetryCount	Number of times the driver retries the primary database server, and if specified, alternate servers until a successful connection is established. The default is 5.
ConnectionRetryDelay	Wait interval, in seconds, between connection retry attempts when the ConnectionRetryCount property is set to a positive integer. The default is 1.
DatabaseName	Name of the database to which you want to connect.
FailoverGranularity	Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. The default is nonAtomic (the driver continues with the failover process and posts any exceptions on the statement on which they occur).
FailoverMode	The failover method you want the driver to use. The default is connect (connection failover is used).
FailoverPreconnect	Specifies whether the driver tries to connect to the primary and an alternate server at the same time. The default is false (the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection).

---

**Table 3-6. Summary: Failover Properties for the DB2 Driver (cont.)**

---

Property	Characteristic
LoadBalancing	Sets whether the driver will use client load balancing in its attempts to connect to the database servers (primary and alternate). If client load balancing is enabled, the driver uses a random pattern instead of a sequential pattern in its attempts to connect. The default is false (client load balancing is not used).
PortNumber	Port listening for connections on the primary database server. This property is supported only for data source connections. The default port number is 50000.
ServerName	IP address or server name of the primary database server. This property is supported only for data source connections.

---

See “[Using Failover](#)” on page 58 and “[Using Client Load Balancing](#)” on page 65 for overviews of failover and client load balancing.

---

## Bulk Load

The driver supports DataDirect Bulk Load, a feature that allows your application to send large numbers of rows of data to the database in a continuous stream instead of in numerous smaller database protocol packets. Similar to batch operations, performance improves because far fewer network round trips are required. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations. See “[Using DataDirect Bulk Load](#)” on page 829 for more information.

# 4 The Informix Driver

The DataDirect Connect for JDBC Informix driver (the "Informix driver") supports:

- Informix Dynamic Server 9.2, 9.3, 9.4, 10, and 11

---

## Data Source and Driver Classes

The data source class for the Informix driver is:

`com.ddtek.jdbcx.informix.InformixDataSource`

See "[Connecting Through Data Sources](#)" on page 48 for information about DataDirect Connect for JDBC data sources.

The driver class for the Informix driver is:

`com.ddtek.jdbc.informix.InformixDriver`

---

## Connection URL

The connection URL format for the Informix driver is:

`jdbc:datadirect:informix://hostname:port[;property=value[;...]]`

where:

`hostname` is the TCP/IP address or TCP/IP host name of the server to which you are connecting. See "[Using IP Addresses](#)" on page 50 for details on using IP addresses.

NOTE: Untrusted applets cannot open a socket to a machine other than the originating host.

*port* is the number of the TCP/IP port.

*property=value* specifies connection properties. For a list of connection properties and their valid values, see “[Connection Properties](#)” on page 179.

For example:

```
jdbc:datadirect:informix://server4:1526;InformixServer=ol_test;  
DatabaseName=ACCT01;User=test;Password=secret
```

---

## J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the Informix resource adapter is:

`com.ddtek.resource.spi.InformixManagedConnectionFactory`

See “[J2EE Connector Architecture Resource Adapters](#)” on page 37 for information about using DataDirect Connect for JDBC drivers as J2EE Connector Architecture resource adapters.

---

# Connection Properties

This section lists the JDBC connection properties supported by the Informix driver and describes each property. The properties have the form:

*property=value*

You can use these connection properties with either the JDBC Driver Manager or DataDirect Connect for JDBC data sources unless otherwise noted.

## NOTES:

- All connection property names are case-insensitive. For example, Password is the same as password. Required properties are noted as such.
- The data type listed for each connection property is the Java data type used for the property value in a JDBC data source.

## AccountingInfo

Description	Accounting information to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is the accounting information.
Default	empty string
Data Type	String
See Also	<a href="#">"Client Information for Connections" on page 204</a>

## AlternateServers

**Description** A list of alternate database servers that is used to failover new or lost connections, depending on the failover method selected. See the [FailoverMode](#) property for information about choosing a failover method.

**Valid Values** `(servername1[:port1][;property=value[;...]]  
[,servername2[:port2][;property=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 2003 is used.

Optional connection properties are [DatabaseName](#) and [InformixServer](#).

**Example** The following URL contains alternate server entries for server2 and server3. The alternate server entries contain the optional [InformixServer](#) property.

```
jdbc:datadirect:informix://server1:2003;  
InformixServer=TestServer;DatabaseName=Test;  
AlternateServers=(server2:2003;InformixServer=  
TestServer2,server3:2003;InformixServer=TestServer3)
```

**Default** None

**Data type** String

**See Also** [“Configuring Failover” on page 211](#)

## ApplicationName

Description	The name of the application to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is the name of the application.
Default	empty string
Data Type	String
See Also	<a href="#">“Client Information for Connections” on page 204</a>

## ClientHostName

Description	The host name of the client machine to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is the host name of the client machine.
Default	empty string
Data Type	String
See Also	<a href="#">“Client Information for Connections” on page 204</a>

## ClientUser

Description	The user ID to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is a valid user ID.

Default	empty string
Data Type	String
See Also	<a href="#">"Client Information for Connections" on page 204</a>

## CodePageOverride

Description	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 that is returned from or written to the database is converted using the specified code page.  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.
Valid Values	<i>string</i>  where <i>string</i> is the name of a valid code page that is supported by your JVM.
Example	CP950
Default	None
Data Type	String

## ConnectionRetryCount

Description	The number of times the driver retries connection attempts to the primary database server, and if specified, alternate servers until a successful connection is established.  NOTE: If an application sets a login timeout value (for example, using <code>DataSource.loginTimeout</code> or <code>DriverManager.loginTimeout</code> ), and the login timeout expires, the driver ceases connection attempts.
-------------	---

Valid Values 0 | x

where x is a positive integer.

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.

**NOTE:** The [ConnectionRetryDelay](#) property specifies the wait interval, in seconds, to occur between retry attempts.

**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.

**Default** 5 (seconds)

**Data Type** int

**See Also** ["Configuring Failover" on page 211](#)

## ConnectionRetryDelay

**Description** The number of seconds the driver waits between connection retry attempts when [ConnectionRetryCount](#) is set to a positive integer.

Valid Values 0 | x

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.

Default 1 (second)

Data Type int

See Also ["Configuring Failover" on page 211](#)

## ConvertNull

Description Controls how data conversions are handled for null values.

Valid Values 0 | 1

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 the data type of the column value.

Default 1

Data Type int

## Database

Description An alias for the [DatabaseName](#) property.

## DatabaseName

Description	The name of the database to which you want to connect.  If this property is not specified, a connection is established to the specified server without connecting to a particular database. A connection that is established to the server without connecting to the database allows an application to use CREATE DATABASE and DROP DATABASE SQL statements. These statements require that the driver cannot be connected to a database. An application can connect to the database after the connection is established by executing the DATABASE SQL statement.  Refer to your IBM Informix documentation for details on using the CREATE DATABASE, DROP DATABASE, and DATABASE SQL statements.
Valid Values	<i>string</i>  where <i>string</i> is the name of a Informix database.
Default	None
Data Type	String
Alias	<a href="#">Database</a> property. If both the Database and DatabaseName properties are specified in a connection URL, the last property that is positioned in the connection URL is used. For example, if your application specifies the following connection URL, the value of the Database connection property would be used instead of the value of the DatabaseName connection property.  <code>jdbc:datadirect:informix://server1:2003; InformixServer=ol_test;DatabaseName=jdbc;Database=acct; User=test;Password=secret</code>

## DBDate

**Description** Sets the Informix DBDate server option for formatting literal date values when inserting, updating, and retrieving data in DATE columns. Using this property, you can customize the following items:

- Order in which the month, day, and year fields appear in a date string
- Year field to contain two or four digits
- Separator character used to separate the date fields

This property does not affect the format of the string in the date escape syntax. Dates specified using the date escape syntax always use the JDBC escape format: *yyyy-mm-dd*.

**Valid Values** DMY2 | DMY4 | MDY2 | MDY4 | Y4DM | Y4MD | Y2DM

where D is a 2-digit day field, M is a 2-digit month field, Y2 is a 2-digit year field, and Y4 is a 4-digit year field.

If unspecified, the format of literal date values conforms to the default server behavior.

Optionally, a separator character may be specified as the last character of the value. Valid separator characters are:

Hyphen (-)  
Period (.)  
Forward slash (/)

If a separator is not specified, a forward slash (/) is used to separate the fields.

**Example** A value of Y4MD- specifies a date format that has a 4-digit year, followed by the month and then by the day. The date fields are separated by a hyphen (-) (for example: 2004-02-15).

**Default** None

**Data Type** String

## FailoverGranularity

Description	Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. This property is ignored if <a href="#">FailoverMode=connect</a> .
Valid Values	<p>nonAtomic   atomic   atomicWithRepositioning   disableIntegrityCheck</p> <p>If set to nonAtomic, the driver continues with the failover process and posts any exceptions on the statement on which they occur.</p> <p>If set to atomic, the driver fails the entire failover process if an exception is generated as the result of restoring the state of the connection. If an exception is generated as a result of restoring the state of work in progress, the driver continues with the failover process, but generates an exception warning that the Select statement must be reissued.</p> <p>If set to atomicWithRepositioning, the driver fails the entire failover process if any exception is generated as the result of restoring the state of the connection or the state of work in progress.</p> <p>If set to disableIntegrityCheck, the driver does not verify that the rows that are restored during the failover process match the original rows. This value is applicable only when <a href="#">FailoverMode=select</a>.</p>
Default	nonAtomic
Data Type	String
See Also	<a href="#">"Configuring Failover" on page 211</a>

## FailoverMode

Description Specifies the type of failover method the driver uses.

Valid Values connect | extended | select

If set to connect, the driver provides failover protection for new connections only.

If set to extended, the driver provides failover protection for new and lost connections, but not any work in progress.

If set to select, the driver provides failover protection for new and lost connections. In addition, it preserves the state of work that is performed by the last Select statement that was executed on the Statement object.

### NOTES:

- The [AlternateServers](#) property specifies one or multiple alternate servers for failover and is required for all failover methods.
- The [FailoverGranularity](#) property determines which action the driver takes if exceptions occur during the failover process.
- The [FailoverPreconnect](#) property specifies whether the driver tries to connect to multiple database servers (primary and alternate) at the same time.

Default connect

Data Type String

See Also [“Configuring Failover” on page 211](#)

## FailoverPreconnect

Description	Specifies whether the driver tries to connect to the primary and an alternate server at the same time. This property is ignored if <a href="#">FailoverMode=connect</a> .
Valid Values	true   false
	If set to true, the driver tries to connect to the primary and an alternate server at the same time. This can be useful if your application is time-sensitive and cannot absorb the wait for the failover connection to succeed.
	If set to false, the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection. This value provides the best performance, but your application typically experiences a short wait while the failover connection is attempted.
	NOTE: The <a href="#">AlternateServers</a> property specifies one or multiple alternate servers for failover.
Default	false
Data Type	boolean
See Also	<a href="#">"Configuring Failover" on page 211</a>

## FetchBufferSize

Description	Specifies the size (in bytes) of the fetch buffer that the driver uses when retrieving data from the database.
	Decreasing the fetch buffer size reduces memory consumption, but means more network round trips, which decreases performance. Increasing the fetch buffer size improves performance because fewer network round trips are needed to return data from the database.

To determine the optimal value, use the following formula:

$$X = A * B * 50$$

where *A* is the number of rows that your application returns when executing Select statements, and *B* is the number of row columns that are typically returned when executing Select statements.

**Valid Values**

*x*

where *x* is a positive integer from 1 to 32767.

**Default**

32767

**Data Type**

int

**See Also** [“Performance Considerations” on page 201](#)

## ImportStatementPool

**Description**

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.

**Default**

empty string

**Data Type**

String

**See Also** [“Importing Statements into a Statement Pool” on page 820](#)

[“Performance Considerations” on page 201](#)

## InformixServer

Description	REQUIRED. The name of the Informix database server to which you want to connect.
Valid Values	<i>string</i> where <i>string</i> is the name of the Informix database server.
Default	None
Data Type	String

## InitializationString

Description	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	<i>command</i> [ [ ; <i>command</i> ] . . . ] where <i>command</i> is a SQL command.  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	<code>jdbc:datadirect:informix://server1:2003; InformixServer=TestServer;DatabaseName=Test; InitializationString=(command1;command2)</code>
Default	None
Data Type	String

## InInsensitiveResultSetBufferSize

Description      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.

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.

Default      2048

Data Type     int

See Also     [“Performance Considerations” on page 201](#)

## JavaDoubleToString

Description	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
	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.
Default	false
Data Type	boolean

## JDBCBehavior

Description	Determines how the driver describes database data types that map to the following JDBC 4.0 data types: NCHAR, NVARCHAR, NLONGVARCHAR, NCLOB, and SQLXML.  This property is applicable only when the application is using Java SE 6.
Valid Values	0   1

If set to 0, the driver describes the data types as JDBC 4.0 data types when using Java SE 6.

If set to 1, the driver describes the data types using JDBC 3.0-equivalent data types, regardless of JVM. This allows your application to continue using JDBC 3.0 types in a Java SE 6 environment.

Default 1

Data Type int

## LoadBalancing

Description 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

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).

Default false

Data Type boolean

See Also ["Configuring Failover" on page 211](#)

## LoginTimeout

Description	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.  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.
Default	0
Data Type	int

## MaxPooledStatements

Description	The maximum number of pooled prepared statements for this connection. Setting MaxPooledStatements to an integer greater than zero (0) enables the driver's internal prepared statement pooling, which is useful when the driver is not running from within an application server or another application that provides its own prepared statement pooling.
Valid Values	0   $x$ where $x$ is a positive integer.  If set to 0, the driver's internal prepared statement pooling is not enabled.  If set to $x$ , the driver enables the DataDirect Statement Pool Monitor and uses the specified value to cache a certain number of prepared statements that are created by an application. If the value set for this property is greater than the number of

prepared statements used by the application, all prepared statements that are created by the application are cached. Because CallableStatement is a sub-class of PreparedStatement, CallableStatements also are cached.

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

**Default** 0

**Data Type** int

**Alias** [MaxStatements](#) property

**See Also** [“Performance Considerations” on page 201](#)

[Appendix J “Statement Pool Monitor” on page 811](#)

## MaxStatements

**Description** An alias for the [MaxPooledStatements](#) property.

## Password

**Description** REQUIRED. A password that is used to connect to your Informix database. A password is required if security is enabled on your database. Contact your system administrator to obtain your password.

**Valid Values** *string*

where *string* is a valid password. The password is case-sensitive.

**Default** None

**Data Type** String

## PortNumber

Description	REQUIRED. The TCP port of the primary database server that is listening for connections to the Informix database.  This property is supported only for data source connections.
Valid Values	<i>port</i>  where <i>port</i> is the port number.
Default	Varies depending on operating system
Data Type	int

## ProgramID

Description	The product and version information of the driver on the client to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
Valid Values	DDJVVRM  where: <ul style="list-style-type: none"><li>■ <i>DDJ</i> is an identifier for the DataDirect Connect <i>for JDBC</i> driver.</li><li>■ <i>VV</i> identifies a 2-digit version number (with high-order 0 in the case of a 1-digit version).</li><li>■ <i>RR</i> identifies a 2-digit release number (with high-order 0 in the case of a 1-digit release).</li><li>■ <i>M</i> identifies a 1-character modification level (0-9 or A-Z).</li></ul>
Example	DDJ04100
Default	empty string
Data Type	String
See Also	<a href="#">"Client Information for Connections" on page 204</a>

## QueryTimeout

Description	Sets the default query timeout (in seconds) for all statements created by a connection.
Valid Values	-1   0   $x$ where $x$ is the number of seconds.  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 created by the connection. To override the default timeout value that is set by this connection option, call the Statement.setQueryTimeout() method to set a timeout value for a particular statement.
Default	0
Data Type	int

## ResultSetMetaDataOptions

Description	Determines whether the driver returns table name information in the ResultSet metadata for Select statements.
Valid Values	0   1  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.

Default 0

Data Type int

See Also ["Performance Considerations" on page 201](#)

## ServerName

Description 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 InformixServer

Default None

Data Type String

## SpyAttributes

Description 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. See [Appendix H “Tracking JDBC Calls with DataDirect Spy™” on page 783](#) for a list of supported attributes.

**NOTE:** If coding a path on Windows to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example:

```
log=(file)C:\\temp\\spy.log.
```

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

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

**Default** None

**Data Type** String

## UseDelimitedIdentifier

**Description** Controls how the Informix server interprets double quote ("") characters in SQL statements.

**NOTE:** If the DELIMIDENT environment variable is set on the server, the driver cannot change the setting. In this case, any value set by this property is ignored.

**Valid Values** true | false

If set to true, the driver sets the Informix DELIMIDENT server option, causing the Informix server to interpret strings enclosed in double quotes as identifiers, not as string literals.

If set to false, the driver does not set the Informix DELIMIDENT server option and the Informix server interprets strings enclosed in double quotes as string literals, not as identifiers.

**Default** true

**Data Type** boolean

## User

Description	REQUIRED. The user name that is used to connect to the Informix database.
Valid Values	<i>string</i> where <i>string</i> is a valid user name. The user name is case-insensitive.
Default	None
Data Type	String

---

## Performance Considerations

You can optimize your application's performance if you set the Informix driver connection properties as described in this section:

**FetchBufferSize:** Decreasing the fetch buffer size reduces memory consumption, but means more network round trips, which decreases performance. Increasing the fetch buffer size improves performance because fewer network round trips are needed to return data from the database.

**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.

See “[Managing Connections](#)” on page 711 for more information about using prepared statement pooling to optimize performance.

---

## Data Types

[Table 4-1](#) lists the data types supported by the Informix driver and how they are mapped to the JDBC data types.

---

***Table 4-1. Informix Data Types***

---

Informix Data Type	JDBC Data Type
BLOB	BLOB
BOOLEAN	BIT
BYTE	LONGVARBINARY
CHAR	CHAR
CLOB	CLOB
DATE	DATE
DATETIME HOUR TO SECOND	TIME
DATETIME YEAR TO DAY	DATE
DATETIME YEAR TO FRACTION(5)	TIMESTAMP
DATETIME YEAR TO SECOND	TIMESTAMP
DECIMAL	DECIMAL
FLOAT	FLOAT
INT8	BIGINT
INTEGER	INTEGER

**Table 4-1. Informix Data Types (cont.)**

<b>Informix Data Type</b>	<b>JDBC Data Type</b>
LVARCHAR	VARCHAR
MONEY	DECIMAL
NCHAR	CHAR  NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NCHAR (if using Java SE 6) or CHAR (if using another JVM).
NVARCHAR	VARCHAR  NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NVARCHAR (if using Java SE 6) or VARCHAR (if using another JVM).
SERIAL	INTEGER
SERIAL8	BIGINT
SMALLFLOAT	REAL
SMALLINT	SMALLINT
TEXT	LONGVARCHAR
VARCHAR	VARCHAR

See [Appendix D “getTypeInfo” on page 631](#) for more information about data types.

---

## Client Information for Connections

The Informix driver allows applications to store and return the following types of client information associated with a particular connection:

- Name of the application
- User ID
- Host name of the client
- Additional accounting information, such as an accounting ID
- Product name and version of the Informix driver

This information can be used for database administration and monitoring purposes. See [Appendix C “Client Information for Connections” on page 621](#) for details.

---

## SQL Escape Sequences

See [Appendix F “SQL Escape Sequences for JDBC” on page 717](#) for information about the SQL escape sequences supported by the Informix driver.

---

## Isolation Levels

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

---

# Using Scrollable Cursors

The Informix driver supports scroll-sensitive result sets, scroll-insensitive result sets, and updatable result sets.

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

---

## Parameter Metadata Support

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

### Insert and Update Statements

The Informix driver supports returning parameter metadata for Insert and Update statements.

### Select Statements

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

Parameter metadata can be returned for a Select statement if one of the following conditions is true:

- The statement contains a predicate value expression that can be targeted against the source tables in the associated FROM clause. For example:

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

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

- The statement contains a predicate value expression part that is a nested query. The nested query's metadata must describe a single column. For example:

```
SELECT * FROM foo WHERE (SELECT x FROM y  
WHERE z = 1) < ?
```

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

```
SELECT col1, col2 FROM foo WHERE col1 = ? and col2 > ?
```

```
SELECT ... WHERE colname = (SELECT col2 FROM t2  
WHERE col3 = ?)
```

```
SELECT ... WHERE colname LIKE ?
```

```
SELECT ... WHERE colname BETWEEN ? and ?
```

```
SELECT ... WHERE colname IN (?, ?, ?)
```

```
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE col1 < ?)
```

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

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

Joins are supported. For example:

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

Fully qualified names and aliases are supported. For example:

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

When parameter metadata is requested for a column defined as an approximate numeric data type, the driver returns a scale of 255, which indicates the column has an approximate numeric data type and has no scale. For example, suppose we create a table where col2 is an approximate numeric data type with a precision of 20:

```
CREATE table fooTest(col1 int, col2 decimal(20))
```

The driver returns parameter metadata that indicates that col2 has a data type of decimal, a precision of 20, and a scale of 255.

## Stored Procedures

The Informix driver does not support returning parameter metadata for stored procedure arguments.

---

# ResultSet MetaData Support

If your application requires table name information, the Informix driver can return table name information in ResultSet metadata for Select statements. By setting the ResultSetMetaDataOptions property to 1, the Informix 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 getTableNames() method may return an empty string for each column in the result set.

The table name information that is returned by the Informix 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 Informix 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 Informix 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 Informix driver also can return schema name and catalog name information when the ResultSetMetaData.getSchemaName() and ResultSetMetaData.getCatalogName() methods are called if the driver can determine that information. For example, for the

following statement, the Informix driver returns "test" for the catalog name, "test1" for the schema name, and "foo" for the table name:

```
SELECT * FROM test.test1.foo
```

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

---

## Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

See <http://www.jcp.org/en/jsr/detail?id=114> for more information about JSR 114.

---

## Blob and Clob Searches

When searching a Clob value for a string pattern using the `Clob.position` method, the search pattern must be less than or equal to a maximum value of 4096 bytes. Similarly, when searching a Blob value for a byte pattern using the `Blob.position`

method, the search pattern must be less than or equal to a maximum value of 4096 bytes.

---

## Auto-Generated Keys Support

The Informix driver supports retrieving the values of auto-generated keys. An auto-generated key returned by the Informix driver is the value of a SERIAL column or a SERIAL8 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 that contains parameters:

- When using an Insert statement that contains no parameters, the Informix 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 that contains parameters, the Informix 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)`

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.

See “[Retrieving Auto-Generated Keys](#)” on page 709 for information about how auto-generated keys can improve performance.

---

## Configuring Failover

- 1 Specify the primary and alternate servers:
  - Specify your primary server using a connection URL or data source.
  - Specify one or multiple alternate servers by setting the `AlternateServers` property.See “[Specifying Primary and Alternate Servers](#)” on page 212.
- 2 Choose a failover method by setting the `FailoverMode` connection property. The default method is connection failover (`FailoverMode=connect`). See “[Using Failover](#)” on page 58 for an overview of each failover method.
- 3 If `FailoverMode=extended` or `FailoverMode=select`, set the `FailoverGranularity` property to specify how you want the driver to behave if exceptions occur while trying to reestablish a lost connection. The default behavior of the driver is to continue with the failover process and post any exceptions on the statement on which they occur (`FailoverGranularity=nonAtomic`).
- 4 Optionally, configure the connection retry feature. See “[Specifying Connection Retry](#)” on page 214.

- 5 Optionally, set the FailoverPreconnect property if you want the driver to establish a connection with the primary and an alternate server at the same time. The default behavior is to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection (FailoverPreconnect=false).

## Specifying Primary and Alternate Servers

Connection information for primary and alternate servers can be specified using either one of the following methods:

- Connection URL through the JDBC Driver Manager
- JDBC data source

For example, the following connection URL for the Informix driver specifies connection information for the primary and alternate servers using a connection URL:

```
jdbc:datadirect:informix://server1:2003;InformixServer=TestServer;  
DatabaseName=TestServer;User=test;Password=secret;  
AlternateServers=(server2:2003;InformixServer=TestServer2,server3:2003)
```

In this example:

```
...server1:2003;InformixServer=TestServer;  
DatabaseName=TestServer...
```

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=(server2:2003;  
InformixServer=TestServer2,server3:2003)
```

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

```
InformixDataSource mds = new InformixDataSource();
mds.setDescription("My InformixDataSource");
mds.setServerName("server1");
mds.setPortNumber(2003);
mds.setInformixServer("TestServer");
mds.setDatabaseName("TestServer");
mds.setUser("test");
mds.setPassword("secret");
mds.setAlternateServers=(server2:2003;InformixServer=
TestServer2,server3:2003)
```

In this example, connection information for the primary server is specified using the ServerName, PortNumber, InformixServer, and DatabaseName properties. Connection information for 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. The port number is optional for each alternate server entry. If unspecified, the port number specified for the primary server is used.

*property=value* is either of the following connection properties: **DatabaseName** or **InformixServer**. These connection properties are optional for each alternate server entry. For example:

```
jdbc:datadirect:informix://server1:2003;InformixServer=TestServer;  
DatabaseName=TestServer;User=test;Password=secret;  
AlternateServers=(server2:2003;InformixServer=TestServer2;  
DatabaseName=TestServer,server3:2003)
```

If you do not specify an optional connection property in an alternate server entry, the connection to that alternate server uses the property specified in the URL. For example, if you specify **InformixServer=TestServer** and **DatabaseName=TestServer** for the primary server, but do not specify the **InformixServer** and **DatabaseName** properties in the alternate server entry as shown in the following URL, the driver uses the **InformixServer** and **DatabaseName** specified for the primary server and tries to connect to the **TestServer** database on the Informix server **TestServer**:

```
jdbc:datadirect:informix://server1:2003;InformixServer=TestServer;  
DatabaseName=TestServer;AlternateServers=(server2:2003,server3:2003)
```

## Specifying Connection Retry

Connection retry allows the Informix 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:informix://server1:2003;InformixServer=TestServer;  
DatabaseName=TestServer;User=test;Password=secret;  
AlternateServers=(server2:2003;DatabaseName=TEST2,server3:2003;  
DatabaseName=TEST3);ConnectionRetryCount=2;ConnectionRetryDelay=5
```

In this example, if a successful connection is not established on the Informix 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.

## Failover Properties

[Table 4-2](#) summarizes the connection properties that control how failover works with the Informix driver. See “[Connection Properties](#)” on page 179 for details about configuring each property.

---

**Table 4-2. Summary: Failover Properties for the Informix Driver**

---

Property	Characteristic
AlternateServers	One or multiple alternate database servers. An IP address or server name identifying each server is required. Port number and supported connection properties ( <code>DatabaseName</code> and <code>InformixServer</code> ) are optional. If the port number is unspecified, the port specified for the primary server is used.
ConnectionRetryCount	Number of times the driver retries the primary database server, and if specified, alternate servers until a successful connection is established. The default is 5.
ConnectionRetryDelay	Wait interval, in seconds, between connection retry attempts when the <code>ConnectionRetryCount</code> property is set to a positive integer. The default is 1.
DatabaseName	Name of the Informix database to which you want to connect.
FailoverGranularity	Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. The default is <code>nonAtomic</code> (the driver continues with the failover process and posts any exceptions on the statement on which they occur).
FailoverMode	The failover method you want the driver to use. The default is <code>connect</code> (connection failover is used).

**Table 4-2. Summary: Failover Properties for the Informix Driver (cont.)**

<b>Property</b>	<b>Characteristic</b>
FailoverPreconnect	Specifies whether the driver tries to connect to the primary and an alternate server at the same time. The default is false (the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection).
InformixServer	Name of the Informix database server to which you want to connect.
LoadBalancing	Sets whether the driver will use client load balancing in its attempts to connect to database servers (primary and alternate). If client load balancing is enabled, the driver uses a random pattern instead of a sequential pattern in its attempts to connect. The default is false (client load balancing is disabled).
PortNumber	Port listening for connections on the primary database server. This property is supported only for data source connections.
ServerName	IP address or server name of the primary database server. This property is supported only for data source connections.

See “[Using Failover](#)” on page 58 and “[Using Client Load Balancing](#)” on page 65 for overviews of failover and client load balancing.

# 5 The MySQL Driver

The DataDirect Connect *for JDBC* MySQL driver ("MySQL driver") supports MySQL 5.0.x and 5.1 with the following storage engines:

- InnoDB – Transactional
- MyISAM – Non-Transactional
- Memory (formerly HEAP) – Non-Transactional

NOTE: You must purchase commercially licensed MySQL database software or a MySQL Enterprise subscription to use the DataDirect Connect *for JDBC* for MySQL driver with MySQL software.

---

## Data Source and Driver Classes

The data source class for the MySQL driver is:

`com.ddtek.jdbc.mysql.MySQLDataSource`

See "[Connecting Through Data Sources](#)" for information about DataDirect Connect *for JDBC* data sources.

The driver class for the MySQL driver is:

`com.ddtek.jdbc.mysql.MySQLDriver`

## Connection URL

The connection URL format for the MySQL driver is:

```
jdbc:datadirect:mysql://hostname:[port][;property=value[;...]]
```

where:

*hostname* is the TCP/IP address or TCP/IP host name of the server to which you are connecting. See “[Using IP Addresses](#)” for details on using IP addresses.

NOTE: Untrusted applets cannot open a socket to a machine other than the originating host.

*port* is the number of the TCP/IP port.

*property=value* specifies connection properties. For a list of connection properties and their valid values, see “[Connection Properties](#)” on page 219.

For example:

```
jdbc:datadirect:mysql://server1:3306;User=test;Password=secret
```

---

## J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the MySQL resource adapter is:

com.ddtek.resource.spi.MySQLManagedConnectionFactory

See “[J2EE Connector Architecture Resource Adapters](#)” for information about using DataDirect Connect for JDBC drivers as J2EE Connector Architecture resource adapters.

---

# Connection Properties

This section lists the JDBC connection properties supported by the MySQL driver and describes each property. The properties have the form:

*property=value*

You can use these connection properties with either the JDBC Driver Manager or DataDirect Connect *for JDBC* data sources unless otherwise noted.

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

## AccountingInfo

Description	Accounting information to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is the accounting information.
Default	empty string
Data Type	String
See Also	<a href="#">“Client Information for Connections” on page 251</a>

## AlternateServers

Description	A list of alternate database servers that is used to failover new or lost connections, depending on the failover method selected. See the <a href="#">FailoverMode</a> property for information about choosing a failover method.
-------------	---

<b>Valid Values</b>	( <i>servername1[:port1][;DatabaseName=value]</i> [ , <i>servername2[:port2][;DatabaseName=value]</i> ]...)
	The server name ( <i>servername1</i> , <i>servername2</i> , and so on) is required for each alternate server entry. Port number ( <i>port1</i> , <i>port2</i> , and so on) and connection properties ( <i>property=value</i> ) 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 3306 is used.
	<b>DatabaseName</b> is an optional connection property.
<b>Example</b>	The following URL contains alternate server entries for server2 and server3. The alternate server entries contain the optional <b>DatabaseName</b> property.
	<pre>jdbc:datadirect:mysql://server1:3306;DatabaseName=TEST; User=test;Password=secretAlternateServers=(server2:3306; DatabaseName=TEST2,server3:3306;DatabaseName=TEST3)</pre>
<b>Default</b>	None
<b>Data type</b>	String
<b>See Also</b>	<a href="#">“Configuring Failover” on page 258</a>

## ApplicationName

<b>Description</b>	The name of the application to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
<b>Valid Values</b>	<i>string</i>
	where <i>string</i> is the name of the application.
<b>Default</b>	empty string
<b>Data Type</b>	String
<b>See Also</b>	<a href="#">“Client Information for Connections” on page 251</a>

## ClientHostName

Description	The host name of the client machine to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is the host name of the client machine.
Default	empty string
Data Type	String
See Also	<a href="#">“Client Information for Connections” on page 251</a>

## ClientUser

Description	The user ID to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is a valid user ID.
Default	empty string
Data Type	String
See Also	<a href="#">“Client Information for Connections” on page 251</a>

## CodePageOverride

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

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.

Valid Values    UTF8

If set to UTF8, the UTF-8 code page overrides the default database code page or column collation.

Default    None

Data Type    String

## ConnectionRetryCount

Description    The number of times the driver retries connection attempts to the primary database server, and if specified, alternate servers until a successful connection is established.

NOTE: 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.

Valid Values    0 |  $x$

where  $x$  is a positive integer.

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.

NOTE: The [ConnectionRetryDelay](#) property specifies the wait interval, in seconds, to occur between retry attempts.

Example	If this property is set to 2 and alternate servers are specified using the <a href="#">AlternateServers</a> property, the driver retries the list of servers (primary and alternate) twice after the initial retry attempt.
Default	5 (seconds)
Data Type	int
See Also	<a href="#">"Configuring Failover" on page 258</a>

## ConnectionRetryDelay

Description	The number of seconds the driver waits between connection retry attempts when <a href="#">ConnectionRetryCount</a> is set to a positive integer.
Valid Values	0   $x$ where $x$ is a number of seconds.  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 <a href="#">ConnectionRetryCount</a> is set to 2, this property is set to 3, and alternate servers are specified using the <a href="#">AlternateServers</a> 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.
Default	1 (second)
Data Type	int
See Also	<a href="#">"Configuring Failover" on page 258</a>

## ConvertNull

Description      Controls how data conversions are handled for null values.

Valid Values    0 | 1

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 the data type of the column value.

Default        1

Data Type     int

## Database

Description      An alias for the [DatabaseName](#) property.

## DatabaseName

Description      The name of the database to which you want to connect.

Valid Values    *string*

where *string* is the name of a MySQL database.

Default        None

Data Type     String

Alias          [Database](#) property. If both the Database and DatabaseName properties are specified in a connection URL, the last property that is positioned in the connection URL is used. For example, if

your application specifies the following connection URL, the value of the Database connection property would be used instead of the value of the DatabaseName connection property.

```
jdbc:datadirect:mysql://server1:3306;DatabaseName=jdbc;  
Database=acct;User=test;Password=secret
```

## EncryptionMethod

Description	Determines whether SSL encryption is used to encrypt and decrypt data transmitted over the network between the driver and database server.  NOTE: Connection hangs can occur if the driver attempts to connect to a database server that does not support SSL. You may want to set a login timeout using the <a href="#">LoginTimeout</a> property to avoid problems when connecting to a server that does not support SSL.
Valid Values	noEncryption   SSL  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.  When SSL is enabled, the following properties also apply:  <a href="#">HostNameInCertificate</a> <a href="#">KeyStore</a> (for SSL client authentication) <a href="#">KeyStorePassword</a> (for SSL client authentication) <a href="#">KeyPassword</a> (for SSL client authentication) <a href="#">TrustStore</a> <a href="#">TrustStorePassword</a> <a href="#">ValidateServerCertificate</a>
Default	noEncryption
Data Type	String

See Also [“Data Encryption” on page 250](#)

[“Performance Considerations” on page 247](#)

## FailoverGranularity

Description	Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. This property is ignored if <a href="#">FailoverMode=connect</a> .
Valid Values	<code>nonAtomic</code>   <code>atomic</code>   <code>atomicWithRepositioning</code>   <code>disableIntegrityCheck</code>  If set to <code>nonAtomic</code> , the driver continues with the failover process and posts any exceptions on the statement on which they occur.  If set to <code>atomic</code> , the driver fails the entire failover process if an exception is generated as the result of restoring the state of the connection. If an exception is generated as a result of restoring the state of work in progress, the driver continues with the failover process, but generates an exception warning that the <code>Select</code> statement must be reissued.  If set to <code>atomicWithRepositioning</code> , the driver fails the entire failover process if any exception is generated as the result of restoring the state of the connection or the state of work in progress.  If set to <code>disableIntegrityCheck</code> , the driver does not verify that the rows restored during the failover process match the original rows. This value is applicable only when <code>FailoverMode=select</code> .
Default	<code>nonAtomic</code>
Data Type	<code>String</code>
See Also	<a href="#">“Configuring Failover” on page 258</a>

## FailoverMode

Description	Specifies the type of failover method the driver uses.
Valid Values	connect   extended   select
	If set to connect, the driver provides failover protection for new connections only.
	If set to extended, the driver provides failover protection for new and lost connections, but not any work in progress.
	If set to select, the driver provides failover protection for new and lost connections. In addition, it preserves the state of work performed by the last Select statement executed on the Statement object.

### NOTES:

- The [AlternateServers](#) property specifies one or multiple alternate servers for failover and is required for all failover methods.
- The [FailoverGranularity](#) property determines which action the driver takes if exceptions occur during the failover process.
- The [FailoverPreconnect](#) property specifies whether the driver tries to connect to multiple database servers (primary and alternate) at the same time.

Default connect

Data Type String

See Also ["Configuring Failover" on page 258](#)

## FailoverPreconnect

Description	Specifies whether the driver tries to connect to the primary and an alternate server at the same time. This property is ignored if <a href="#">FailoverMode=connect</a> .
Valid Values	true   false
	If set to true, the driver tries to connect to the primary and an alternate server at the same time. This can be useful if your application is time-sensitive and cannot absorb the wait for the failover connection to succeed.
	If set to false, the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection. This value provides the best performance, but your application typically experiences a short wait while the failover connection is attempted.
	NOTE: The <a href="#">AlternateServers</a> property specifies one or multiple alternate servers for failover.
Default	false
Data Type	boolean
See Also	<a href="#">“Configuring Failover” on page 258</a>

## HostNameInCertificate

Description	Specifies a host name for certificate validation when SSL encryption is enabled ( <a href="#">EncryptionMethod=SSL</a> ) and validation is enabled ( <a href="#">ValidateServerCertificate=true</a> ). 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.
-------------	--

**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.

**Valid Values** *host\_name* | #SERVERNAME#

where *host\_name* is a valid host name.

If *host\_name* is specified, the driver compares the specified host name to the DNSName value of the SubjectAlternativeName in the certificate. If a DNSName value does not exist in the SubjectAlternativeName or if the certificate does not have a SubjectAlternativeName, the driver compares the host name with the Common Name (CN) part of the certificate's Subject name. 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 specified in the connection URL or data source of the connection to the DNSName value of the SubjectAlternativeName in the certificate. If a DNSName value does not exist in the SubjectAlternativeName or 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.

**Default** empty string

**Data Type** String

## ImportStatementPool

Description	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	<i>string</i>
	where <i>string</i> is the path and file name of the file to be used to load the contents of the statement pool.
Default	empty string
Data Type	String
See Also	<a href="#">"Importing Statements into a Statement Pool" on page 820</a> <a href="#">"Performance Considerations" on page 247</a>

## InitializationString

Description	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.
	You can use this property to override the session wait_timeout value that is used by the <a href="#">InteractiveClient</a> property for the current connection.
Valid Values	<i>string</i>
	where <i>string</i> 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** The following URL sets the session wait\_timeout variable to 3600 (seconds) and turns on a warning count for simple inserts.

```
jdbc:datadirect:mysql://server1:3306;DatabaseName=test;
InitializationString=(set@@wait_timeout=3600;
set@@sql_warnings=1)
```

**Default** None

**Data Type** String

## InOrderColumnAccess

**Description** Determines whether the driver caches column values for the current row, which allows an application to access the column values in any order.

**Valid Values** true | false

If set to true, the driver does not cache column values for the current row. Applications must retrieve the column values in the order that they are listed in the result set. Once a value is retrieved, it cannot be retrieved again. For example, suppose an application selects the following rows:

```
SELECT last_name, first_name, emp_ID FROM Employees
```

The application must retrieve column values for *last\_name*, *first\_name*, and *emp\_ID* for the current row in the order they appear in the result set. Because the driver does not need to cache the column values, performance is improved. If an application attempts to access column values in an order other than the order in which they appear in the result set, the driver throws an exception. For example, if the application attempts to access the column value for *emp\_ID* and then *last\_name*, the

driver throws an exception when the application attempts to access the column value for `last_name`.

If set to false, the driver caches column values for the current row. Columns of the current row can be accessed in any order.

Default false

Data Type boolean

See Also ["Performance Considerations" on page 247](#)

## InInsensitiveResultSetBufferSize

Description 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.

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.

Default	2048
Data Type	int
See Also	<a href="#">"Performance Considerations" on page 247</a>

## InteractiveClient

Description	Controls the length of time (in seconds) a connection can be idle before the server ends the connection by setting the connection's session wait_timeout variable.
Valid Values	true   false
	If set to true, the session wait_timeout variable is set using the MySQL global interactive_timeout variable value. If the value of the global interactive_timeout variable is changed after a connection is established, any connection that is created after the variable is changed uses the new timeout value.
	If set to false, the session wait_timeout variable is set using the MySQL global wait_timeout variable value. If the value of the global wait_timeout variable is changed after a connection is established, any connection that is created after the variable is changed uses the new timeout value.
	If the value of the session wait_timeout variable is changed after a connection is established, only the timeout for that connection is affected. For example, if the session wait_timeout variable is changed to a value of 3600 (seconds) using the <a href="#">InitializationString</a> property as shown in the following URL, the new timeout value is used for the current connection:
	<pre>jdbc:datadirect:mysql://server1:3306;DatabaseName=test; InitializationString=(set@wait_timeout=3600)</pre>
Default	false
Data Type	boolean

## JavaDoubleToString

Description	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
	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.
Default	false
Data Type	boolean

## JDBCBehavior

Description	Determines how the driver describes database data types that map to the following JDBC 4.0 data types: NCHAR, NVARCHAR, NLONGVARCHAR, NCLOB, and SQLXML.  This property is applicable only when the application is using Java SE 6.  NOTE: This property has no effect for the MySQL driver because MySQL does not support JDBC 4.0 data types.
-------------	---

Valid Values 0 | 1

If set to 0, the driver describes the data types as JDBC 4.0 data types when using Java SE 6.

If set to 1, the driver describes the data types using JDBC 3.0-equivalent data types, regardless of JVM. This allows your application to continue using JDBC 3.0 types in a Java SE 6 environment.

Default 1

Data Type int

## KeyPassword

Description 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.

Default None

Data Type String

## KeyStore

Description 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.

NOTE: The keystore and truststore files can be the same file.

Valid Values *string*

where *string* is a valid directory of a keystore file.

Default None

Data Type String

## KeyStorePassword

Description 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.

NOTE: The keystore and truststore files can be the same file.

Valid Values *string*

where *string* is a valid password.

Default None

Data Type String

## LoadBalancing

Description	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 <a href="#">AlternateServers</a> property.
Valid Values	true   false
	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).
Default	false
Data Type	boolean
See Also	<a href="#">“Configuring Failover” on page 258</a>

## LobCommandSize

Description	Specifies the size (in bytes) of the chunks the driver uses to send long data to the server.
Valid Values	x where x is an integer not exceeding the value of the MySQL max_allowed_packet system variable.

If the value of LobCommandSize is larger than the value of the max\_allowed\_packet variable, the driver sends long data in chunks of the size set by the max\_allowed\_packet variable.

**Example** If the total size of a long data value is 3 MB and LobCommandSize is set to the default value of 1048576 (bytes), the driver sends the long data to the server in chunks of 1,048,576 bytes.

**Default** 1048576 (bytes)

**Data Type** int

## LoginTimeout

**Description** 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.

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.

**Default** 0

**Data Type** int

## MaxPooledStatements

**Description** The maximum number of pooled prepared statements for this connection. Setting MaxPooledStatements to an integer greater than zero (0) enables the driver's internal prepared statement pooling, which is useful when the driver is not running from

within an application server or another application that provides its own prepared statement pooling.

**Valid Values** 0 |  $x$

where  $x$  is a positive integer.

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

If set to  $x$ , the driver enables the DataDirect Statement Pool Monitor and uses the specified value to cache a certain number of prepared statements that are created by an application. For 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. If the value set for this property is greater than the number of prepared statements used by the application, all prepared statements created by the application are cached. Because CallableStatement is a sub-class of PreparedStatement, CallableStatements also are cached.

**Default** 0

**Data Type** int

**Alias** [MaxStatements](#) property

**See Also** [“Performance Considerations” on page 247](#)

[Appendix J “Statement Pool Monitor” on page 811](#)

## MaxStatements

**Description** An alias for the [MaxPooledStatements](#) property.

## Password

Description	A password that is used to connect to your MySQL database. A password is required if user ID/Password authentication is enabled on your database. Contact your system administrator to obtain your password.
Valid Values	<i>string</i> where <i>string</i> is a valid password. The password is case-sensitive.
Default	None
Data Type	String

## PortNumber

Description	The TCP port of the primary database server that is listening for connections to the MySQL database.  This property is supported only for data source connections.
Valid Values	<i>port</i> where <i>port</i> is the port number.
Default	3306
Data Type	int

## ProgramID

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

**Valid Values** DDJVVRM

where:

- DDJ is an identifier for a DataDirect Connect *for JDBC* driver.
- VV identifies a 2-digit version number (with high-order 0 in the case of a 1-digit version).
- RR identifies a 2-digit release number (with high-order 0 in the case of a 1-digit release).
- M identifies a 1-character modification level (0-9 or A-Z).

**Example** DDJ04100

**Default** empty string

**Data Type** String

**See Also** [“Client Information for Connections” on page 251](#)

## QueryTimeout

**Description** 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.

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.

Default 0

Data Type int

## ResultSetMetaDataOptions

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

Valid Values 0 | 1

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.

Default 0

Data Type int

## ServerName

Description 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 MySQLServer

**Default** None

**Data Type** String

## SpyAttributes

**Description** 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. See [Appendix H "Tracking JDBC Calls with DataDirect Spy™" on page 783](#) for a list of supported attributes.

NOTE: If coding a path on Windows to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example:

`log=(file)C:\\temp\\spy.log.`

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

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

**Default** None

**Data Type** String

## TreatBinaryAsChar

**Description** Determines the JDBC data type the driver uses to describe and return data that MySQL stores as BINARY or VARBINARY. By default, the driver describes and returns data that MySQL stores

as BINARY or VARBINARY as BINARY or VARBINARY values, respectively.

**Valid Values** true | false

If set to true, the driver describes and returns data that MySQL stores as BINARY or VARBINARY as CHAR or VARCHAR values, respectively.

If set to false, the driver describes and returns data that MySQL describes as BINARY or VARBINARY as BINARY or VARBINARY values, respectively.

**Example** Create the following MySQL table:

```
CREATE TABLE binTable (col1 binary(3))
```

and execute the following Insert statement:

```
INSERT INTO binTable values('abc')
```

Then, set this property to true and execute the following query:

```
SELECT col1 FROM binTable
```

The driver would return the value of col1 as a CHAR value, "abc", instead of a BINARY value "616263".

**Default** false

**Data Type** boolean

## TrustStore

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

This value overrides the directory of the truststore file that is specified by the javax.net.ssl.trustStore Java system property. If

this property is not specified, the truststore directory is specified by the javax.net.ssl.trustStore Java system property.

This property is ignored if [ValidateServerCertificate=false](#).

Valid Values *string*

where *string* is the directory of the truststore file.

Default None

Data Type String

## TrustStorePassword

Description 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.

Default None

Data Type String

## User

Description	The user name that is used to connect to the MySQL database. A user name is required if user ID/password authentication is enabled on your database. Contact your system administrator to obtain your user name.
Valid Values	<i>string</i>
	where <i>string</i> is a valid user name. The user name is case-sensitive.
Default	None
Data Type	String

## ValidateServerCertificate

Description	Determines whether the driver validates the certificate that is sent by the database server when SSL encryption is enabled ( <a href="#">EncryptionMethod=SSL</a> ). 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

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.

NOTE: Truststore information is specified using the [TrustStore](#) and [TrustStorePassword](#) properties or by using Java system properties.

Default true

Data Type boolean

---

## Performance Considerations

You can optimize your application's performance if you set the MySQL driver connection properties as described in this section:

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

**InOrderColumnAccess:** If your application always gets data in the order that it was returned from the database, this property should be set to true. When this option is set to true, the driver does not cache column values of the current row, which improves performance.

**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.

See “[Managing Connections](#)” for more information about using prepared statement pooling to optimize performance.

---

## Data Types

[Table 5-1](#) lists the data types supported by the MySQL driver and describes how they are mapped to JDBC data types.

---

*Table 5-1. MySQL Data Types*

---

MySQL Data Type	JDBC Data Type
BIGINT	BIGINT
BIGINT UNSIGNED	BIGINT
BINARY	BINARY
BIT	BINARY
BLOB	LONGVARBINARY
CHAR( <i>n</i> )	CHAR
DATE	DATE
DATETIME	TIMESTAMP
DECIMAL	DECIMAL
DECIMAL UNSIGNED	DECIMAL
DOUBLE	DOUBLE

---

**Table 5-1. MySQL Data Types (cont.)**

---

MySQL Data Type	JDBC Data Type
DOUBLE UNSIGNED	DOUBLE
FLOAT	REAL
FLOAT UNSIGNED	REAL
INTEGER	INTEGER
INTEGER UNSIGNED	INTEGER
LONGBLOB	LONGVARBINARY
LONGTEXT	LONGVARCHAR
MEDIUMBLOB	LONGVARBINARY
MEDIUMINT	INTEGER
MEDIUMINT UNSIGNED	INTEGER
MEDIUMTEXT	LONGVARCHAR
SMALLINT	SMALLINT
SMALLINT UNSIGNED	SMALLINT
TEXT	LONGVARCHAR
TIME	TIME
TIMESTAMP	TIMESTAMP
TINYBLOB	LONGVARBINARY
TINYINT	TINYINT
TINYINT UNSIGNED	TINYINT
TINYTEXT	LONGVARCHAR
VARBINARY	VARBINARY
VARCHAR( <i>n</i> )	VARCHAR
YEAR	SMALLINT

---

See [Appendix D “getTypeInfo” on page 631](#) for more information about data types.

---

# Data Encryption

SSL secures the integrity of your data by encrypting information and providing authentication. See “[Data Encryption Across the Network](#)” on page 69 for an overview.

**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:

- 1 Set the EncryptionMethod property to SSL.
- 2 Specify the location and password of the truststore file used for SSL server authentication. Either set the TrustStore and TrustStore properties or their corresponding Java system properties (javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword, respectively).
- 3 To validate certificates sent by the database server, set the ValidateServerCertificate property to true.
- 4 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.
- 5 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 KeyStore 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.

---

## Client Information for Connections

The MySQL driver allows applications to store and return the following types of client information associated with a particular connection:

- Name of the application
- User ID
- Host name of the client
- Additional accounting information, such as an accounting ID
- Product name and version of the MySQL driver

This information can be used for database administration and monitoring purposes. See [Appendix C “Client Information for Connections” on page 621](#) for details.

---

## SQL Escape Sequences

See [Appendix F “SQL Escape Sequences for JDBC” on page 717](#) for information about SQL escape sequences supported by the MySQL driver.

---

## Isolation Levels

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

## Using Scrollable Cursors

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

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

---

## Large Object (LOB) Support

The MySQL driver allows you to retrieve and update long data, specifically LONGVARBINARY and LONGVARCHAR 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
- 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.

---

# Parameter Metadata Support

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

## Insert and Update Statements

The MySQL driver supports returning parameter metadata for the following forms of Insert and Update statements:

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

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

## Select Statements

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

Parameter metadata can be returned for a Select statement if one of the following conditions is true:

- The statement contains a predicate value expression that can be targeted against the source tables in the associated FROM clause. For example:

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

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

- The statement contains a predicate value expression part that is a nested query. The nested query's metadata must describe a single column. For example:

```
SELECT * FROM foo WHERE (SELECT x FROM y  
WHERE z = 1) < ?
```

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

```
SELECT col1, col2 FROM foo WHERE col1 = ? and col2 > ?
```

```
SELECT ... WHERE colname = (SELECT col2 FROM t2  
WHERE col3 = ?)
```

```
SELECT ... WHERE colname LIKE ?
```

```
SELECT ... WHERE colname BETWEEN ? and ?
```

```
SELECT ... WHERE colname IN (?, ?, ?)
```

```
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE col1 < ?)
```

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

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

Joins are supported. For example:

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

Fully qualified names and aliases are supported. For example:

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

## Stored Procedures

The MySQL driver supports returning parameter metadata for stored procedure arguments.

---

## ResultSet MetaData Support

If your application requires table name information, the MySQL driver can return table name information in ResultSet metadata for Select statements. If you set the `ResultSetMetaDataOptions` property to 1, the MySQL 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 MySQL 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 MySQL 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 MySQL 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 MySQL 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 MySQL 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 MySQL driver supports any JSR 114 implementation of the RowSet interface, including:

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

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

---

## Auto-Generated Keys Support

The MySQL driver supports retrieving the values of auto-generated keys. An auto-generated key returned by the MySQL driver is the value of an auto-increment 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 that contains parameters.

- When using an Insert statement that contains no parameters, the MySQL driver supports the following forms of the Statement.execute and Statement.executeUpdate methods to inform the driver to return the 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 that contains parameters, the MySQL driver supports the following forms of the `Connection.prepareStatement` method to inform the driver to return the values of auto-generated keys:
- `Connection.prepareStatement(String sql, int autoGeneratedKeys)`
  - `Connection.prepareStatement(String sql, int[] columnIndexes)`
  - `Connection.prepareStatement(String sql, String[] columnNames)`

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.

See “[Retrieving Auto-Generated Keys](#)” for more information.

---

## Configuring Failover

- 1 Specify the primary and alternate servers:
  - Specify your primary server using a connection URL or data source.
  - Specify one or multiple alternate servers by setting the `AlternateServers` property.
- See “[Specifying Primary and Alternate Servers](#)” on page 259.
- 2 Choose a failover method by setting the `FailoverMode` connection property. The default method is `connection`

failover (FailoverMode=connect). See “[Using Failover](#)” on [page 58](#) for an overview of each failover method.

- 3 If FailoverMode=extended or FailoverMode=select, set the FailoverGranularity property to specify how you want the driver to behave if exceptions occur while trying to reestablish a lost connection. The default behavior of the driver is to continue with the failover process and post any exceptions on the statement on which they occur (FailoverGranularity=nonAtomic).
- 4 Optionally, configure the connection retry feature. See “[Specifying Connection Retry](#)” on [page 261](#).
- 5 Optionally, set the FailoverPreconnect property if you want the driver to establish a connection with the primary and an alternate server at the same time. The default behavior is to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection (FailoverPreconnect=false).

## Specifying Primary and Alternate Servers

Connection information for primary and alternate servers can be specified using either one of the following methods:

- Connection URL through the JDBC Driver Manager
- JDBC data source

For example, the following connection URL for the MySQL driver specifies connection information for the primary and alternate servers using a connection URL:

```
jdbc:datadirect:mysql://server1:3306;DatabaseName=TEST;User=test;  
Password=secret;AlternateServers=(server2:3306;DatabaseName=TEST2,  
server3:3306;DatabaseName=TEST3)
```

In this example:

```
...server1:3306;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=(server2:3306;DatabaseName=TEST2,
server3:3306;DatabaseName=TEST3)
```

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

```
MySQLDataSource mds = new MySQLDataSource();
mds.setDescription("My MySQLDataSource");
mds.setServerName("server1");
mds.setPortNumber(3306);
mds.setDatabaseName("TEST");
mds.setUser("test");
mds.setPassword("secret");
mds.setAlternateServers(server2:3306;DatabaseName=TEST2,
server3:3306;DatabaseName=TEST3)
```

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.

*port<sub>1</sub>* is the port number on which the first alternate database server is listening, *port<sub>2</sub>* 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 3306 is used.

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

```
jdbc:datadirect:mysql://server1:3306;DatabaseName=TEST;User=test;  
Password=secret;AlternateServers=(server2:3306;DatabaseName=TEST2,  
server3:3306;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:mysql://server1:3306;DatabaseName=TEST;User=test;  
Password=secret;AlternateServers=(server2:3306,server3:3306)
```

## Specifying Connection Retry

Connection retry allows the MySQL 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:mysql://server1:3306;DatabaseName=TEST;User=test;  
Password=secret;AlternateServers=(server2:3306;DatabaseName=TEST2,  
server3:3306;DatabaseName=TEST3);ConnectionRetryCount=2;  
ConnectionRetryDelay=5
```

In this example, if a successful connection is not established on the MySQL 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.

## Failover Properties

[Table 5-2](#) summarizes the connection properties that control how failover works with the MySQL driver. See “[Connection Properties](#)” on page 219 for details about configuring each property.

---

**Table 5-2. Summary: Failover Properties for the MySQL Driver**

---

Property	Characteristic
AlternateServers	One or multiple alternate database servers. An IP address or server name identifying each server is required. Port number and the connection property <code>DatabaseName</code> are optional. If the port number is unspecified, the port number specified for the primary server is used. If a port number is unspecified for the primary server, the default port number of 3306 is used.
ConnectionRetryCount	Number of times the driver retries the primary database server, and if specified, alternate servers until a successful connection is established. The default is 5.
ConnectionRetryDelay	Wait interval, in seconds, between connection retry attempts when the <code>ConnectionRetryCount</code> property is set to a positive integer. The default is 1.
DatabaseName	Name of the database to which you want to connect.

**Table 5-2. Summary: Failover Properties for the MySQL Driver (cont.)**

<b>Property</b>	<b>Characteristic</b>
FailoverGranularity	Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. The default is nonAtomic (the driver continues with the failover process and posts any exceptions on the statement on which they occur).
FailoverMode	The failover method you want the driver to use. The default is connect (connection failover is used).
FailoverPreconnect	Specifies whether the driver tries to connect to the primary and an alternate server at the same time. The default is false (the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection).
LoadBalancing	Sets whether the driver will use client load balancing in its attempts to connect to the database servers (primary and alternate). If client load balancing is enabled, the driver uses a random pattern instead of a sequential pattern in its attempts to connect. The default is false (client load balancing is disabled).
PortNumber	Port listening for connections on the primary database server. This property is supported only for DataSource connections. The default port number is 3306.
ServerName	IP address or server name for the primary database server. This property is supported only for DataSource connections.

See “[Using Failover](#)” and “[Using Client Load Balancing](#)” for overviews of failover and client load balancing.



# 6 The Oracle Driver

The DataDirect Connect *for JDBC* Oracle driver (the "Oracle driver") supports:

- Oracle 11g (R1 and R2)
- Oracle 10g R1 and R2
- Oracle 9*i* R1 and R2
- Oracle 8*i* R2 (8.1.6) and R3 (8.1.7)

---

## Data Source and Driver Classes

The data source class for the Oracle driver is:

`com.ddtek.jdbcx.oracle.OracleDataSource`

See "[Connecting Through Data Sources](#)" on page 48 for information about DataDirect Connect *for JDBC* data sources.

The driver class for the Oracle driver is:

`com.ddtek.jdbc.oracle.OracleDriver`

---

## Connection URL

The connection URL format for the Oracle driver is:

```
jdbc:datadirect:oracle://hostname:port[;property=value[;...]]
```

where:

*hostname* is the TCP/IP address or TCP/IP host name of the server to which you are connecting. See “[Using IP Addresses](#)” on page 50 for details on using IP addresses.

NOTE: Untrusted applets cannot open a socket to a machine other than the originating host.

*port* is the number of the TCP/IP port.

*property=value* specifies connection properties. For a list of connection properties and their valid values, see “[Connection Properties](#)” on page 267.

For example:

```
jdbc:datadirect:oracle://server3:1521;ServiceName=ORCL;User=test;  
Password=secret
```

See “[Using tnsnames.ora Files](#)” on page 319 for instructions on retrieving connection information from an Oracle tnsnames.ora file.

---

# J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the Oracle resource adapter is:

`com.ddtek.resource.spi.OracleManagedConnectionFactory`

See "[J2EE Connector Architecture Resource Adapters](#)" on [page 37](#) for information about using DataDirect Connect *for JDBC* drivers as J2EE Connector Architecture resource adapters.

---

## Connection Properties

This section lists the JDBC connection properties supported by the Oracle driver and describes each property. The properties have the form:

*property*=*value*

You can use these connection properties with either the JDBC Driver Manager or DataDirect Connect *for JDBC* data sources unless otherwise noted.

### NOTES:

- All connection property names are case-insensitive. For example, Password is the same as password. Required properties are noted as such.
- The data type listed for each connection property is the Java data type used for the property value in a JDBC data source.

## AccountingInfo

Description	Accounting information to be stored in the database. This value sets the CLIENT_INFO value of the V\$SESSION table in the database.
Valid Values	<i>string</i>
where <i>string</i> is the accounting information.	
NOTE:	Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. See <a href="#">"Returning MetaData About Client Information Locations" on page 627</a> .
Default	empty string
Data Type	String
See Also	<a href="#">"Client Information for Connections" on page 355</a>

## AlternateServers

Description	A list of alternate database servers that is used to failover new or lost connections, depending on the failover method selected. See the <a href="#">FailoverMode</a> property for information about choosing a failover method.  If using a tnsnames.ora file to retrieve connection information, do not specify this property. See <a href="#">"Using tnsnames.ora Files" on page 319</a> for more information.
Valid Values	( <i>servername1[:port1][;property=value[;...]]</i> [, <i>servername2[:port2][;property=value[;...]]</i> ]...)

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 1521 is used.

Optional connection properties are [ServiceName](#) and [SID](#).

NOTE: If using failover with an Oracle RAC, the primary server must be a primary node and any alternate server must be a secondary node.

**Example** The following URL contains alternate server entries for server2 and server3. The alternate server entries contain the optional [ServiceName](#) property. Similarly, you can use the optional [SID](#) property instead.

```
jdbc:datadirect:oracle://server1:1521;ServiceName=TEST;
AlternateServers=(server2:1521;ServiceName=TEST2,
server3:1521;ServiceName=TEST3)
```

**Default** None

**Data type** String

**See Also** [“Configuring Failover” on page 365](#)

## ApplicationName

**Description** The name of the application to be stored in the database. This value sets the dbms\_session value in the database. It also sets the PROGRAM value of the V\$SESSION table. These values are used for database administration/monitoring purposes.

**Valid Values** *string*

where *string* is the name of the application.

NOTE: Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. See [“Returning MetaData About Client Information Locations” on page 627](#).

**Default** empty string

Data Type String

See Also ["Client Information for Connections" on page 355](#)

## AuthenticationMethod

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

Valid Values auto | kerberos | kerberosUIDPassword | ntlm | client | userIDPassword

If set to auto, the driver uses user ID/password, Kerberos, or NTLM authentication when establishing a connection. The driver selects an authentication method based on a combination of criteria, such as whether the application provides a user ID, the driver is running on a Windows platform, and the driver can load the DLL required for NTLM authentication. See ["Using the AuthenticationMethod Property" on page 341](#) for more information about using this value.

If set to kerberos, the driver uses Kerberos authentication. The driver ignores any user ID or password specified.

If set to kerberosUIDPassword, the driver first uses Kerberos to authenticate the user. Next, the driver reauthenticates the user using user ID/password authentication. If a user ID and password are not specified, the driver throws an exception. If either Kerberos or user ID/password authentication fails, the connection attempt fails and the driver throws an exception.

If set to ntlm, the driver uses NTLM authentication if the DLL required for NTLM authentication can be loaded. If the driver cannot load the DLL, the driver throws an exception. The driver ignores any user ID or password specified. This value is supported for Windows clients only.

If set to client, the driver uses the user ID of the user logged onto the system on which the driver is running to authenticate the

user to the database. The Oracle database server relies on the client to authenticate the user and does not provide additional authentication. The driver ignores any user ID or password specified.

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

NOTES:

- The values `type2` and `none` are deprecated, but are recognized for backward compatibility. Use the `ntlm` and `userIdPassword` values, respectively, instead.
- The [User](#) property provides the user ID. The [Password](#) property provides the password.

Default `auto`

Data Type `String`

See Also [“Authentication” on page 340](#)

## BatchPerformanceWorkaround

Description Determines the method that is used to execute batch operations.

Valid Values `true` | `false`

If set to `true`, the driver uses the native Oracle batch mechanism. The native batch mechanism does not return individual update counts for each statement or parameter set in the batch. For this reason, the driver returns a value of `SUCCESS_NO_INFO` (-2) for each entry in the returned update count array. If an application can accept not receiving update count information, setting this property to `true` can significantly improve performance.

If set to `false`, the driver uses the JDBC 3.0-compliant batch mechanism.

Default	false
Data Type	boolean
See Also	<a href="#">"Batch Inserts and Updates" on page 356</a>
	<a href="#">"Performance Considerations" on page 314</a>

## BulkLoadBatchSize

Description 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.

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. See ["DDBulkLoad Interface" on page 603](#) for a description of the method.

Valid Values x

where x is a positive integer.

Default 2048

Data Type long

See Also ["Using DataDirect Bulk Load" on page 829](#)

## CatalogIncludesSynonyms

Description	DEPRECATED. This property is recognized for backward compatibility with existing data sources. Use the <a href="#">CatalogOptions</a> property instead to include synonyms in result sets.
-------------	--

## CatalogOptions

Description	Determines which type of metadata information is included in result sets when an application calls DatabaseMetaData methods.
Valid Values	<p>0   1   2   3   6</p> <p>If set to 0, result sets do not contain remarks or synonyms.</p> <p>If set to 1, result sets contain remarks information that is returned from the following DatabaseMetaData methods: getColumnNames() and getTables().</p> <p>If set to 2, result sets contain synonyms that are returned from the following DatabaseMetaData methods: getColumnNames(), getExportedKeys(), getFunctionColumns(), getFunctions(), getImportedKeys(), getIndexInfo(), getPrimaryKeys(), getProcedureColumns(), and getProcedures().</p> <p>If set to 3, result sets contain both remarks and synonyms (as described for values 1 and 2).</p> <p>If set to 6, a hint is provided to the driver to emulate getColumnNames() calls using the ResultSetMetaData object instead of querying database catalogs for column information. Result sets contain synonyms, but no remarks. 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 getColumnNames(). The argument to getColumnNames() 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 getColumnNames() calls.</p>

Default 2

Data Type int

See Also [“Performance Considerations” on page 314](#)

## ClientHostName

Description The host name of the client machine to be stored in the database. This value sets the MACHINE value in the V\$SESSION table in the database. This value is used for database administration/monitoring purposes.

Valid Values *string*

where *string* is the host name of the client machine.

NOTE: Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. See [“Returning MetaData About Client Information Locations” on page 627](#).

Default empty string

Data Type String

See Also [“Client Information for Connections” on page 355](#)

## ClientUser

Description The user ID to be stored in the database. This value sets the OSUSER value in the V\$SESSION table. This value is used for database administration/monitoring purposes.

Valid Values *string*

where *string* is a valid user ID.

NOTE: Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates

it. See “[Returning MetaData About Client Information Locations](#)” on page 627.

Default empty string

Data Type String

See Also [“Client Information for Connections” on page 355](#)

## CodePageOverride

Description	<p>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 that is returned from or written to the database is converted using the specified code page. This option has no effect on how the driver converts character data to the national character set.</p> <p>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.</p>
Valid Values	<p>utf8   sjis   enhanced_sjis   enhanced_sjis_oracle   ms932   euc_jp_solaris</p> <p>If set to utf8, the driver uses the UTF-8 code page to send data to the Oracle server as Unicode. The UTF-8 code page converts data from the Java String format UTF-16 to UTF-8. If you specify this value, the driver forces the value of the <a href="#">WireProtocolMode</a> property to 2.</p> <p>If set to sjis, the driver uses the SHIFT-JIS code page to convert character data to the JA16SJIS character set.</p> <p>If set to enhanced_sjis, the driver uses the ENHANCED_SJIS code page to convert character data from the Java String format UTF-16 to SJIS as defined by the ICU character conversion library.</p>

In addition, it maps the following MS-932 characters to the corresponding SJIS encoding for those characters:

- \UFF5E Wave dash
- \U2225 Double vertical line
- \UFFE0 Cent sign
- \UFF0D Minus sign
- \UFFE1 Pound sign
- \UFFE2 Not sign

This value is provided for backward compatibility. Only use this value when the Oracle database character set is SHIFT\_JIS.

If set to enhanced\_sjis\_oracle, the driver uses the ENHANCED\_SJIS\_ORACLE code page to convert Character data from the Java String format UTF-16 to Oracle's definition of SJIS. When the driver connects to an Oracle database with a JA16SJIS character set, the driver uses this code page by default. The ENHANCED\_SJIS\_ORACLE code page is a super set of the MS932 code page. Only use this value when the Oracle database character set is SHIFT\_JIS.

If set to ms932, the driver uses the Microsoft MS932 code page to convert Character data from the Java String format UTF-16 to SJIS. This value is provided for backward compatibility because earlier versions of the driver used the MS932 code page when converting Character data to JA16SJIS. Only use this value when the Oracle database character set is SHIFT\_JIS.

If set to euc\_jp\_solaris, the driver uses the EUC\_JP\_Solaris code page to convert Character data to the EUC\_JP character set.

Default None

Data Type String

## CommitBehavior

Description	<p>Determines the redo log behavior. Typically, redo changes that are generated by update transactions are written to disk immediately when a transaction is committed, and the session waits for the disk write to complete before returning control to the application. For Oracle 10g R2 and higher databases, the log writer can write the redo changes to disk in its own time instead of immediately and return control to the application before the disk write is complete instead of waiting. This property controls this behavior by setting the value of the Oracle COMMIT_WRITE session parameter.</p> <p>Not waiting for redo log changes to be written to disk improves performance for applications that have both of the following characteristics:</p> <ul style="list-style-type: none"><li>■ Applications that perform update operations.</li><li>■ Applications where data integrity is not critical. For example, most banking applications cannot tolerate data loss in the event that the server has a problem writing the redo log changes to disk or fails during the process, but many logging applications for diagnostic purposes can.</li></ul>
Valid Values	<p>serverDefault   waitImmediate   waitBatch   noWaitImmediate   noWaitBatch</p> <p>If set to serverDefault, the driver uses the redo log behavior that is set by the database server.</p> <p>If set to waitImmediate, the commit operation does not return control to the application until redo changes are written to disk. Redo changes are written to disk immediately. Use this value if your application processes multiple update transactions one at a time.</p> <p>If set to waitBatch, the commit operation does not return control to the application until redo changes are written to disk. The write task may be deferred by the server until additional</p>

transactions are ready to be written to disk. Use this value if your application processes multiple update transactions simultaneously. Using this value when an application performs only a few transactions decreases performance.

If set to noWaitImmediate, redo changes are written to disk immediately, but the commit operation returns control to the application without waiting for this operation to complete. Use this value if your application processes multiple update transactions one at time and data integrity is not critical.

If set to noWaitBatch, the redo write task may be deferred by the server until additional transactions are ready to be written to disk, but the commit operation returns control to the application without waiting for this operation to complete. Use this value if your application processes multiple update transactions simultaneously and data integrity is not critical.

Default serverDefault

Data Type String

See Also [“Performance Considerations” on page 314](#)

## ConnectionRetryCount

Description The number of times the driver retries connection attempts to the primary database server, and if specified, alternate servers until a successful connection is established.

NOTE: 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.

Valid Values 0 | x

where x is a positive integer.

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.

**NOTE:** The [ConnectionRetryDelay](#) property specifies the wait interval, in seconds, to occur between retry attempts.

Example	If this property is set to 2 and alternate servers are specified using the <a href="#">AlternateServers</a> property, the driver retries the list of servers (primary and alternate) twice after the initial retry attempt.
Default	5 (seconds)
Data Type	int
See Also	<a href="#">"Configuring Failover" on page 365</a>

## ConnectionRetryDelay

Description	The number of seconds the driver waits between connection retry attempts when <a href="#">ConnectionRetryCount</a> is set to a positive integer.
Valid Values	0   $x$ where $x$ is a number of seconds.  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 <a href="#">ConnectionRetryCount</a> is set to 2, this property is set to 3, and alternate servers are specified using the <a href="#">AlternateServers</a> 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.

Default 1 (second)

Data Type int

See Also ["Configuring Failover" on page 365](#)

## ConvertNull

Description Controls how data conversions are handled for null values.

Valid Values 0 | 1

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

If set to 1, the driver checks the data type being requested against the data type of the table column storing 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 the data type of the column value.

Default 1

Data Type int

## Database

Description An alias for the [DatabaseName](#) property.

## DatabaseName

Description	The SID or service name that refers to the instance of the Oracle database that is running on the database server. This property is mutually exclusive with the <a href="#">ServiceName</a> and <a href="#">SID</a> properties.
Valid Values	<i>string</i> where <i>string</i> is the name of an Oracle instance.
Default	None
Data Type	String
Alias	<a href="#">Database</a> property. If both the Database and DatabaseName properties are specified in a connection URL, the last property positioned in the connection URL is used. For example, if your application specifies the following connection URL, the value of the Database connection property would be used instead of the value of the DatabaseName connection property.

```
jdbc:datadirect:oracle://server1:1521;DatabaseName=ORCL;
Database=ORCL_acct;User=test;Password=secret
```

## EnableBulkLoad

Description	Specifies whether the driver uses the native bulk load protocols in the database instead of the batch mechanism for batch inserts. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations. This property allows existing applications with batch inserts to take advantage of bulk load without requiring changes to the application code.
Valid Values	true   false  If set to true, the driver uses the native bulk load protocols for batch inserts.  NOTE: If set to true, any value set for <a href="#">BatchPerformanceWorkaround</a> is ignored.

	If set to false, the driver uses the batch mechanism for batch inserts.
Default	false
Data Type	boolean
See Also	<a href="#">"Using Bulk Load for Batch Inserts" on page 837</a>
	<a href="#">"Performance Considerations" on page 314</a>

## EnableCancelTimeout

Description	Determines whether a cancel request that is sent 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 <code>Statement.setQueryTimeout(5)</code> on a statement and that statement is cancelled because its timeout value was exceeded, a cancel request is sent 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.
Default	false
Data Type	boolean

## EnableServerResultCache

Description	Determines whether the driver adds a hint to SQL statements to enable Oracle's server-side resultset caching feature, which stores the result set in database memory so that it can be reused. Server-side resultset caching can improve performance if your application executes the same query multiple times.
	This property only applies to connections to Oracle 11g database servers that support server-side resultset caching.
Valid Values	true   false
	If set to true, the driver adds a hint to SQL statements to enable server-side resultset caching.
	If set to false, the driver does not add a hint to SQL statements.
Example	This example shows a hint added to a SQL statement:
	<pre>SELECT /*+ result_cache */ * FROM employees</pre>
Default	false
Data Type	boolean
See Also	<a href="#">"Performance Considerations" on page 314</a>

## EncryptionMethod

Description	Determines whether data is encrypted and decrypted when transmitted over the network between the driver and database server.
	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 <a href="#">LoginTimeout</a> property to avoid problems when connecting to a server that does not support SSL.

Valid Values noEncryption | SSL

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.

When SSL is enabled, the following properties also apply:

[HostNameInCertificate](#)

[KeyStore](#) (for SSL client authentication)

[KeyStorePassword](#) (for SSL client authentication)

[KeyPassword](#) (for SSL client authentication)

[TrustStore](#)

[TrustStorePassword](#)

[ValidateServerCertificate](#)

Default noEncryption

Data Type String

See Also [“Data Encryption” on page 352](#)

[“Using tnsnames.ora Files” on page 319](#)

[“Performance Considerations” on page 314](#)

## FailoverGranularity

Description Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. This property is ignored if [FailoverMode=connect](#).

Valid Values nonAtomic | atomic | atomicWithRepositioning | disableIntegrityCheck

If set to nonAtomic, the driver continues with the failover process and posts any exceptions on the statement on which they occur.

If set to atomic, the driver fails the entire failover process if an exception is generated as the result of restoring the state of the connection. If an exception is generated as a result of restoring the state of work in progress, the driver continues with the failover process, but generates an exception warning that the Select statement must be reissued.

If set to atomicWithRepositioning, the driver fails the entire failover process if any exception is generated as the result of restoring the state of the connection or the state of work in progress.

If set to disableIntegrityCheck, the driver does not verify that the rows restored during the failover process match the original rows. This value is applicable only when FailoverMode=select.

**Default** nonAtomic

**Data Type** String

**See Also** [“Configuring Failover” on page 365](#)

## **FailoverMode**

**Description** Specifies the type of failover method the driver uses.

**Valid Values** connect | extended | select

If set to connect, the driver provides failover protection for new connections only.

If set to extended, the driver provides failover protection for new and lost connections, but not any work in progress.

If set to select, the driver provides failover protection for new and lost connections. In addition, it preserves the state of work that is performed by the last Select statement that is executed on the Statement object.

## NOTES:

- The [AlternateServers](#) property specifies one or multiple alternate servers for failover and is required for all failover methods.
- The [FailoverGranularity](#) property determines which action the driver takes if exceptions occur during the failover process.
- The [FailoverPreconnect](#) property specifies whether the driver tries to connect to multiple database servers (primary and alternate) at the same time.

Default connect

Data Type String

See Also ["Configuring Failover" on page 365](#)

## FailoverPreconnect

Description Specifies whether the driver tries to connect to the primary and an alternate server at the same time. This property is ignored if [FailoverMode=connect](#).

Valid Values true | false

If set to true, the driver tries to connect to the primary and an alternate server at the same time. This can be useful if your application is time-sensitive and cannot absorb the wait for the failover connection to succeed.

If set to false, the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection. This value provides the best performance, but your application typically experiences a short wait while the failover connection is attempted.

NOTE: The [AlternateServers](#) property specifies one or multiple alternate servers for failover.

Default	false
Data Type	boolean
See Also	<a href="#">"Configuring Failover" on page 365</a>

## FetchTSWTZasTimestamp

Description	Determines whether column values with the TIMESTAMP WITH TIME ZONE data type are returned as a JDBC TIMESTAMP data type.  This property only applies to connections to Oracle 9i and higher.
Valid Values	true   false  If set to true, column values with the TIMESTAMP WITH TIME ZONE data type are returned as a JDBC TIMESTAMP data type.  If set to false, column values with the TIMESTAMP WITH TIME ZONE data type are returned as a string.
Default	false
Data Type	boolean
See Also	<a href="#">"TIMESTAMP WITH TIME ZONE Data Type" on page 333</a>

## HostNameInCertificate

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

**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 that is specified in the connection URL or data source of the connection to validate the certificate.

**Valid Values** *host\_name* | #SERVERNAME#

where *host\_name* is a valid host name.

If *host\_name* is specified, the driver compares the specified host name to the DNSName value of the SubjectAlternativeName in the certificate. If a DNSName value does not exist in the SubjectAlternativeName or if the certificate does not have a SubjectAlternativeName, the driver compares the host name with the Common Name (CN) part of the certificate's Subject name. 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 a DNSName value does not exist in the SubjectAlternativeName or 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.

**Default** empty string

**Data Type** String

## ImportStatementPool

Description	<p>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.</p> <p>If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver throws an exception.</p>
Valid Values	<p><i>string</i></p> <p>where <i>string</i> is the path and file name of the file to be used to load the contents of the statement pool.</p>
Default	empty string
Data Type	String
See Also	<a href="#">“Importing Statements into a Statement Pool” on page 820</a> <a href="#">“Performance Considerations” on page 314</a>

## InitialColumnBufferSize

Description	<p>Specifies the initial size (in bytes) of the buffer in which the driver places column values when they are retrieved. This option sets the initial size only; the buffer will adjust dynamically as needed to accommodate larger values. For optimal memory usage, the size of the buffer only needs to be as large as the data it holds.</p>
Valid Values	<p>-1   0   <i>x</i></p> <p>where <i>x</i> is an integer from 1 to 2147483648.</p> <p>If set to -1 or 0, the driver automatically determines an initial column buffer size that is based on the width of the column being bound.</p>

If set to *x*, the driver adjusts the buffer to the specified size.

- Example** If the driver retrieves data from a CHAR(4000) column and that data is typically no more than 75 bytes in size, reducing the initial column buffer size to 75 bytes (InitialColumnBufferSize=75) will conserve memory on the client or application server.
- Default** -1
- Data Type** String

## InitializationString

**Description** 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:oracle://server1:1521;ServiceName=ORCL;InitializationString=(command1;command2)`

**Default** None

**Data Type** String

## InInsensitiveResultSetBufferSize

Description	Determines the amount of memory that is used by the driver to cache insensitive result set data.
Valid Values	<p>-1   0   <math>x</math></p> <p>where <math>x</math> is a positive integer.</p> <p>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.</p> <p>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.</p> <p>If set to <math>x</math>, 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.</p>
Default	2048
Data Type	int
See Also	<a href="#">"Performance Considerations" on page 314</a>

## JavaDoubleToString

Description	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
	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.
Default	false
Data Type	boolean

## JDBCBehavior

Description	Determines how the driver describes database data types that map to the following JDBC 4.0 data types: NCHAR, NVARCHAR, NLONGVARCHAR, NCLOB, and SQLXML.  This property is applicable only when the application is using Java SE 6.
Valid Values	0   1

If set to 1, the driver describes the data types using JDBC 3.0-equivalent data types, regardless of JVM. This allows your application to continue using JDBC 3.0 types in a Java SE 6 environment.

Default	1
Data Type	int

## KeyPassword

Description	Specifies the password that is used to access the individual keys in the keystore file when SSL is enabled ( <a href="#">EncryptionMethod=SSL</a> ) 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	<i>string</i> where <i>string</i> is a valid password.
Default	None
Data Type	String

## KeyStore

Description	Specifies the directory of the keystore file to be used when SSL is enabled ( <a href="#">EncryptionMethod=SSL</a> ) 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.  NOTE: The keystore and truststore files can be the same file.
-------------	---

Valid Values *string*

where *string* is a valid directory of a keystore file.

Default None

Data Type String

## KeyStorePassword

Description 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.

NOTE: The keystore and truststore files can be the same file.

Valid Values *string*

where *string* is a valid password.

Default None

Data Type String

## LoadBalancing

Description 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.

If using a tnsnames.ora file to retrieve connection information, do not specify this property. See “[Using tnsnames.ora Files](#)” on page 319 for more information.

Valid Values	true   false
	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).
Default	false
Data Type	boolean
See Also	<a href="#">“Configuring Failover” on page 365</a>

## LoadLibraryPath

Description	<p>Specifies the directory for the DLL for NTLM authentication. The driver looks for the DLL in the specified directory.</p> <p>NOTE: When you install the driver, the NTLM authentication DLLs are installed in the <i>install_dir/lib</i> subdirectory, where <i>install_dir</i> is the product installation directory.</p>
Valid Values	<p><i>string</i></p> <p>where <i>string</i> is the fully qualified path of the directory that contains the DLL for NTLM authentication.</p>

If unspecified, the driver looks for the DLL in a directory that is on the Windows system path defined by the PATH environment variable.

If set to *string*, the driver looks in the specified directory for the DLL. Use this value if you install the driver in a directory that is not on the Windows system path.

**Example** If you installed the driver in a directory named "DataDirect" that is not on the Windows system path, you can use this property to specify the directory containing the NTLM authentication DLL as shown in the following URL.

```
jdbc:datadirect:oracle://server3:1521;ServiceName=ORCL;  
LoadLibraryPath=C:\DataDirect\lib;User=test;Password=secret
```

**Default** None

**Data Type** String

**See Also** ["Configuring NTLM Authentication" on page 350](#)

## LoginTimeout

**Description** 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.

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.

**Default** 0

**Data Type** int

## MaxPooledStatements

Description	The maximum number of pooled prepared statements for this connection. Setting MaxPooledStatements to an integer greater than zero (0) enables the driver's internal prepared statement pooling, which is useful when the driver is not running from within an application server or another application that provides its own prepared statement pooling.
Valid Values	0   $x$ where $x$ is a positive integer.  If set to 0, the driver's internal prepared statement pooling is not enabled.  If set to $x$ , the driver enables the DataDirect Statement Pool Monitor and uses the specified value to cache a certain number of prepared statements created by an application. For 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. 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.
Default	0
Data Type	int
Alias	<a href="#">MaxStatements</a> property
See Also	<a href="#">“Performance Considerations” on page 314</a> <a href="#">Appendix J “Statement Pool Monitor” on page 811</a>

## MaxStatements

Description	An alias for the <a href="#">MaxPooledStatements</a> property.
-------------	--

## NewPassword

**Description** Changes a user's password when a connection is established. This property can be used to comply with corporate security policies that require a regularly occurring password change or to ensure access to the database if Oracle has let the user's password expire. If specified and the connection fails, the password is unchanged.

This property is used only if user ID/password authentication is enabled explicitly ([AuthenticationMethod=userIdPassword](#)) or implicitly ([AuthenticationMethod=auto](#) and a user ID and password is supplied); otherwise, the driver throws an exception.

**NOTE:** The current password also must be specified using the Password property. The driver allows you to use the same value for the current password and the new password, but be aware that your Oracle server security settings may not allow this use.

**Valid Values** *string*

where *string* is a valid password.

**Example** The following connection URL specifies the current password (`secret1`) using the Password connection option, and then specifies the new password (`secret2`) using the NewPassword connection option:

```
jdbc:datadirect:oracle://server1:1521;ServiceName=ORCL;
User=test;Password=secret1;NewPassword=secret2
```

Similarly, the same connection information specified using a JDBC data source would look like this:

```
OracleDataSource mds = new OracleDataSource();
mds.setDescription("My OracleDataSource");
mds.setServerName("server1");
mds.setPortNumber(1521);
mds.setServiceName("ORCL");
mds.setUser("test");
mds.setPassword("secret1");
```

```
mds.setNewPassword( "secret2" );
...
Default    None
Data Type   String
```

## Password

Description	A password that is used to connect to your Oracle database. A password is required if user ID/password authentication is enabled on your database. Contact your system administrator to obtain your password.
Valid Values	<i>string</i> where <i>string</i> is a valid password. The password is case-insensitive.
Default	None
Data Type	String

## PortNumber

Description	The TCP port of the Oracle listener running on the Oracle database server.  This property is supported only for data source connections.  If using a tnsnames.ora file to provide connection information, do not specify this property. See “ <a href="#">Using tnsnames.ora Files</a> ” on page 319 for information about specifying a port number for the Oracle listener using a tnsnames.ora file.
Valid Values	<i>port</i> where <i>port</i> is the port number.

Default 1521  
 Data Type int

## ProgramID

Description The product and version information of the driver on the client to be stored in the database. This value sets the PROCESS value in the V\$SESSION table. This value is used for database administration/monitoring purposes.

Valid Values DDJVVRM

where:

- *DDJ* is an identifier for a DataDirect Connect for JDBC driver.
- *VV* identifies a 2-digit version number (with high-order 0 in the case of a 1-digit version).
- *RR* identifies a 2-digit release number (with high-order 0 in the case of a 1-digit release).
- *M* identifies a 1-character modification level (0-9 or A-Z).

Example DDJ04100

Default empty string

Data Type String

See Also ["Client Information for Connections" on page 355](#)

## QueryTimeout

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

Valid Values -1 | 0 | *x*

where *x* is a number of seconds.

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

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

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

Default 0

Data Type int

## ResultSetMetaDataOptions

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

Valid Values 0 | 1

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.

Default 0

Data Type	int
See Also	<a href="#">"Performance Considerations" on page 314</a>
<h2>SDUSize</h2>	
Description	<p>Specifies the Session Data Unit (SDU) that the driver requests when connecting to the server. The SDU is equivalent to the maximum packet size (size of database protocol packets sent across the network). This property serves only as a suggestion to the database server. The actual SDU that is used is negotiated with the database server.</p> <p>This property is mutually exclusive with the <a href="#">TNSNamesFile</a> and <a href="#">TNSServerName</a> properties. The driver generates an exception if SDUSize is specified in conjunction with either of these properties.</p>
Valid Values	<p>x</p> <p>where x is an integer from 512 and 32767.</p> <p>If set to x, the driver sets the SDU to the specified size. For optimal performance, set this option to the maximum SDU size configured on the database server.</p> <p>NOTE: If your application sends queries that only retrieve small result sets, you may want to use an SDU size smaller than the maximum SDU size configured on the database server. If a result set that contains only one or two rows of data does not completely fill a larger packet, performance will not improve by setting the value to the maximum SDU size.</p>
Default	8192 (the default SDU size for Oracle 11g servers)
Data Type	int
See Also	<a href="#">"Performance Considerations" on page 314</a>

## SendFloatParametersAsString

Description	Determines whether FLOAT, BINARY_FLOAT, and BINARY_DOUBLE parameters are sent to the database server as a string or as a floating point number.
Valid Values	true   false
	If set to true, the driver sends FLOAT, BINARY_FLOAT, and BINARY_DOUBLE parameters to the database server as string values.
	<b>NOTE:</b> Numbers larger than 1.0E127 or smaller than 1.0E-130 cannot be converted to Oracle's number format for Oracle 8 <i>i</i> and Oracle 9 <i>i</i> databases using floating point numbers. When a number larger than 1.0E127 or smaller than 1.0E-130 is encountered, the driver throws an exception. If your application uses numbers in this range against an Oracle 8 <i>i</i> or Oracle 9 <i>i</i> database, set this property to true.
	If set to false, the driver sends FLOAT, BINARY_FLOAT, and BINARY_DOUBLE parameters to the database server as floating point numbers. When Oracle overloaded stored procedures are used, this value ensures that the database server can determine the correct stored procedure to call based on the parameter's data type.
Default	false
Data Type	boolean

## ServerName

Description	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.

If using a tnsnames.ora file to provide connection information, do not specify this property. See “[Using tnsnames.ora Files](#)” on page 319 for information about specifying a server name using a tnsnames.ora file.

Valid Values *string*

where *string* is a valid IP address or server name.

Example 122.23.15.12 or OracleAppServer

Default None

Data Type String

## ServerType

Description Specifies whether the connection is established using a shared or dedicated server process (UNIX) or thread (Windows).

If using a tnsnames.ora file to provide connection information, do not specify this property. See “[Using tnsnames.ora Files](#)” on page 319 for information about specifying the server type using a tnsnames.ora file.

Valid Values shared | dedicated

If set to shared, the server process to be used is retrieved from a pool. The socket connection between the client and server is made to a dispatcher process on the server. This setting allows there to be fewer processes than the number of connections, reducing the need for server resources. Use this value when a server must handle many users with fewer server resources.

If set to dedicated, a server process is created to service only that connection. When that connection ends, so does the process (UNIX) or thread (Windows). The socket connection is made directly between the application and the dedicated server process or thread. When connecting to UNIX servers, a dedicated server process can provide significant performance improvement, but

uses more resources on the server. When connecting to Windows servers, the server resource penalty is insignificant. Use this value if you have a batch environment with low numbers of users.

If unspecified, the driver uses the server type set on the server.

**Default** None

**Data Type** String

**See Also** [“Performance Considerations” on page 314](#) for information about configuring this property for optimal performance.

## ServiceName

**Description** The database service name that specifies the database that is used for the connection. This property is mutually exclusive with the [SID](#) property. The service name is a string that is the global database name—a name that typically comprises the database name and domain name.

This property is useful to specify connections to an Oracle Real Application Cluster (RAC) system rather than a specific Oracle instance because the nodes in a RAC system share a common service name.

If using a tnsnames.ora file to provide connection information, do not specify this property. See [“Using tnsnames.ora Files” on page 319](#) for information about specifying the database service name using a tnsnames.ora file.

**Valid Values** *string*

where *string* is a valid service name.

**Example** sales.us.acme.com

**Default** None

**Data Type** String

## SID

Description	The Oracle System Identifier (SID) that refers to the instance of the Oracle database running on the server. This property is mutually exclusive with the <a href="#">ServiceName</a> property.
	If using a tnsnames.ora file to provide connection information, do not specify this property. See <a href="#">"Using tnsnames.ora Files" on page 319</a> for information about specifying an Oracle SID using a tnsnames.ora file.
Valid Values	<i>string</i> where <i>string</i> is a valid SID.
Default	ORCL
Data Type	String

## SpyAttributes

Description	Enables DataDirect Spy to log detailed information about calls that are issued by the driver on behalf of the application. DataDirect Spy is not enabled by default.
Valid Values	( <i>spy_attribute</i> [; <i>spy_attribute</i> ]...)
	where <i>spy_attribute</i> is any valid DataDirect Spy attribute. See <a href="#">Appendix H "Tracking JDBC Calls with DataDirect Spy™" on page 783</a> for a list of supported attributes.
	NOTE: 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: <code>log=(file)C:\\temp\\spy.log.</code>
Example	The following value instructs the driver to log all JDBC activity to a file using a maximum of 80 characters for each line.
	<code>(log=(file)/tmp/spy.log;linelimit=80)</code>

Default	None
Data Type	String

## StringParamsMustMatchCharColumns

Description	Determines whether the driver uses ORA_CHAR or ORA_VARCHAR bindings for string parameters in a Where clause. Using ORA_VARCHAR bindings can improve performance, but may cause matching problems for CHAR columns.
Valid Values	true   false
	If set to true, the driver uses ORA_CHAR bindings.
	If set to false, the driver uses ORA_VARCHAR bindings, which can improve performance. Use this value if your application does not match string parameters against CHAR columns. For example, in the following code, if col1 is defined as a CHAR(10) and the column name has the string 'abc' in it, the match will fail.
	<pre>ps = con.prepareStatement("SELECT * FROM employees WHERE col1=?"); ps.setString(1, "abc"); rs = ps.executeQuery();</pre>
Default	true
Data Type	boolean
See Also	<a href="#">"Performance Considerations" on page 314</a>

## SupportLinks

Description	Determines whether the driver supports Oracle linked servers, which means a mapping has been defined in one Oracle server to another Oracle server.
-------------	---

Valid Values true | false

If set to true, the driver supports Oracle linked servers. When Oracle linked servers are supported, the driver does not support distributed transactions.

If set to false, the driver does not support Oracle linked servers. In addition, the driver supports distributed transactions. In most cases, setting this property to false provides the best performance.

Default false

Data Type boolean

## SysLoginRole

Description Specifies whether the user is logged on the database with the Oracle system privilege SYSDBA or the Oracle system privilege SYSOPER. For example, you may want the user to be granted the SYSDBA privilege to allow the user to create or drop a database.

Refer to your Oracle documentation for information about which operations are authorized for the SYSDBA and SYSOPER system privileges.

Valid Values sysdba | sysoper

If set to sysdba, the user is logged on the database with the Oracle system privilege SYSDBA. The user must be granted SYSDBA system privileges before the connection is attempted by the driver. If not, the driver throws an exception and the connection attempt fails.

If set to sysoper, the user is logged on the database with the Oracle system privilege SYSOPER. The user must be granted SYSOPER system privileges before the connection is attempted by the driver. If not, the driver throws an exception and the connection attempt fails.

If this property is set to an empty string or is unspecified, the user is logged in without SYSDBA or SYSOPER privileges.

**Default** empty string

**Data Type** String

## TNSNamesFile

**Description** The path and filename to the tnsnames.ora file from which connection information is retrieved. The tnsnames.ora file contains connection information that is mapped to Oracle net service names. Using a tnsnames.ora file to centralize connection information simplifies maintenance when changes occur.

If you specify this property, you also must specify the [TNSServerName](#) property.

If this property is specified, do not specify the following properties to prevent connection information conflicts:

[AlternateServers](#)  
[LoadBalancing](#)  
[PortNumber](#)  
[ServerName](#)  
[ServerType](#)  
[ServiceName](#)  
[SID](#)

If any of these properties are specified in addition to this property, the driver throws an exception.

**Valid Values** *string*

where *string* is a valid path to and file name of a tnsnames.ora file.

**Default** None

Data Type String

See Also ["Using tnsnames.ora Files" on page 319](#)

## TNSServerName

Description The Oracle net service name that is used to reference the connection information in a tnsnames.ora file. The value of this property must be a valid net service name entry in the tnsnames.ora file specified by the [TNSNamesFile](#) property.

If this property is specified, you also must specify the [TNSNamesFile](#) property.

If this property is specified, do not specify the following properties to prevent connection information conflicts:

[AlternateServers](#)

[LoadBalancing](#)

[PortNumber](#)

[ServerName](#)

[ServerType](#)

[ServiceName](#)

[SID](#)

If any of these properties are specified in addition to this property, the driver throws an exception.

Valid Values *string*

where *string* is a valid Oracle net service name.

Default None

Data Type String

See Also ["Using tnsnames.ora Files" on page 319](#)

## TrustStore

Description	<p>Specifies the directory of the truststore file to be used when SSL is enabled (<a href="#">EncryptionMethod=SSL</a>) and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.</p> <p>This value overrides the directory of the truststore file that is specified by the <code>javax.net.ssl.trustStore</code> Java system property. If this property is not specified, the truststore directory is specified by the <code>javax.net.ssl.trustStore</code> Java system property.</p> <p>This property is ignored if <a href="#">ValidateServerCertificate=false</a>.</p>
Valid Values	<p><i>string</i></p> <p>where <i>string</i> is the directory of the truststore file.</p>
Default	None
Data Type	String

## TrustStorePassword

Description	<p>Specifies the password that is used to access the truststore file when SSL is enabled (<a href="#">EncryptionMethod=SSL</a>) and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.</p> <p>This value overrides the password of the truststore file that is specified by the <code>javax.net.ssl.trustStorePassword</code> Java system property. If this property is not specified, the truststore password is specified by the <code>javax.net.ssl.trustStorePassword</code> Java system property.</p> <p>This property is ignored if <a href="#">ValidateServerCertificate=false</a>.</p>
Valid Values	<p><i>string</i></p> <p>where <i>string</i> is a valid password for the truststore file.</p>

Default None  
Data Type String

## User

Description The user name that is used to connect to the Oracle database. A user name is required only if user ID/password authentication is enabled on your database. Contact your system administrator to obtain your user name.

Valid Values *string*  
where *string* is a valid user name. The user name is case-insensitive.

Default None  
Data Type String

## ValidateServerCertificate

Description 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  
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.

Truststore information is specified using the [TrustStore](#) and [TrustStorePassword](#) properties or by using Java system properties.

Default true

Data Type boolean

## WireProtocolMode

Description Specifies whether the driver optimizes network traffic to the Oracle server for result sets for repeating data in some or all columns, and for inserts and updates of images, pictures, long text, or binary data (Blob and Clob data).

Valid Values 1 | 2

If set to 1, the driver operates in normal wire protocol mode without optimizing network traffic for result sets for repeating data in some or all columns, and inserts and updates of Blob and Clob data.

If set to 2, the driver optimizes network traffic to the Oracle server for:

- Result sets containing multiple rows that have repeating data in some or all columns. Specifically, if the same column contains identical data across multiple consecutive rows in the result set, setting this value can improve performance.

Setting this value may degrade performance for single row result sets or result sets that do not contain repeating data.

- Inserts and updates of Blob and Clob data.

Default 1

Data Type int

See Also [“Performance Considerations” on page 314](#)

---

## Performance Considerations

You can optimize your application’s performance if you set the Oracle driver connection properties as described in this section:

**BatchPerformanceWorkaround:** The driver can use a JDBC 3.0-compliant batch mechanism or the native Oracle batch mechanism to execute batch operations. If your application does not use update count information, performance can be improved by using the native Oracle batch environment. The JDBC 3.0-compliant mechanism returns individual update counts for each statement or parameter set in the batch as required by the JDBC 3.0 specification. The native Oracle batch mechanism does not return individual update counts for each statement or parameter set in the batch. For this reason, when the native Oracle batch mechanism is used, the driver returns a value of SUCCESS\_NO\_INFO (-2) in the returned update count array.

**CatalogOptions:** Retrieving synonym and remarks information is expensive. If your application does not need to return this information, the driver can improve performance. Default driver behavior is to include synonyms in the result set of calls to the following DatabaseMetaData methods: `getColumns()`, `getExportedKeys()`, `getFunctionColumns()`, `getFunctions()`, `getImportedKeys()`, `getIndexInfo()`, `getPrimaryKeys()`, `getProcedureColumns()`, and `getProcedures()`. In addition, the

driver can include Remarks information in the result sets of calls to the following DatabaseMetaData methods: getColumns() and getTables(). If your application needs to return synonyms for getColumns() calls, but not remarks, the driver can emulate getColumns() calls using the ResultSetMetaData object instead of querying database catalogs for the column information. Using emulation can improve performance because the SQL statement formulated by the emulation is less complex than the SQL statement formulated using getColumns().

**CommitBehavior:** Typically, redo changes generated by update transactions are written to disk immediately when the transaction is committed, and the session waits for the disk write to complete before returning control to the application.

Oracle 10g R2 can let the log writer write the redo changes to disk in its own time instead of immediately and return control to the application before the disk write is complete instead of waiting. Not waiting for the disk write improves performance for applications that perform update operations and where data integrity is not critical. For example, most banking applications cannot tolerate data loss in the event that the server has a problem writing the redo changes to disk or fails during the process, but many logging applications for diagnostic purposes can.

**EnableBulkLoad:** For batch inserts, the driver can use native bulk load protocols instead of the batch mechanism. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations. Set this property to true to allow existing applications with batch inserts to take advantage of bulk load without requiring changes to the code.

**EnableServerResultCache:** If your application connects to Oracle 11g and executes the same query multiple times, you can improve performance by using the Oracle feature server-side resultset caching. When enabled, Oracle stores the result set in database memory. On subsequent executions of the same query, the result set is returned from database memory if the

underlying tables have not been modified. Without result set caching, the server would process the query and formulate a new result set.

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

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

**MaxPooledStatements:** To improve performance, the driver's own internal prepared statement pooling should be enabled when the driver does not run from within an application server or from within another application that 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.

See “[Managing Connections](#)” on page 711 for more information about using prepared statement pooling to optimize performance.

**SDUSize:** Typically, it is optimal for the client to use the maximum Session Data Unit (SDU) size that the server allows. This reduces the total number of round trips required to return data to the client, thus improving performance. Therefore, performance can

be improved if this property is set to the maximum SDU size of the database server.

**ServerType:** When using a dedicated server connection, a server process on UNIX (a thread on Windows) is created to serve only your application connection. When you disconnect, the process goes away. The socket connection is made directly between your application and this dedicated server process. This can provide considerable performance improvements, but will use significantly more resources on UNIX servers. Because this is a thread on Oracle servers running on Windows platforms, the additional resource usage on the server is significantly less. The ServerType property should be set to dedicated when you have a batch environment with lower numbers of connections, your Oracle server has excess processing capacity and memory available when at maximum load, or if you have a performance-sensitive application that would be degraded by sharing Oracle resources with other applications.

**StringParamsMustMatchCharColumns:** Using ORA\_VARCHAR bindings for string parameters in a Where clause can improve performance, but may cause matching problems for CHAR columns. If your application does not match string parameters against CHAR columns, set this property to false for the driver to use ORA\_VARCHAR bindings.

**WireProtocolMode:** Set this property to 2 if: 1) your application executes Select statements that return more than one row, the rows returned have repeating data in some or all of the columns, and the repeated data is in consecutive rows (for example, the data in column1/row1 is the same as the data in column1/row2) or 2) your application updates or inserts images, pictures, or long text or binary data. In either of these cases, performance can be improved by setting this property to 2. When set to 2, the driver optimizes network traffic to the Oracle server for repeating or long data. If your application: 1) returns single row result sets or result sets that do not contain repeating data or 2) does not update or insert long data, this property should be set to 1; otherwise, performance may be degraded.

---

## Support for Oracle RAC

Oracle introduced Real Application Cluster (RAC) with Oracle 9*i*, and RAC is also a key feature of Oracle 10g. Oracle RAC allows a single physical Oracle database to be accessed by concurrent instances of Oracle running across several different CPUs.

An Oracle RAC is composed of a group of independent servers, or nodes, that cooperate as a single system. A cluster architecture such as this provides applications access to more computing power when needed, while allowing computing resources to be used for other applications when database resources are not as heavily required. For example, in the event of a sudden increase in network traffic, an Oracle RAC can distribute the load over many nodes, a feature referred to as *server load balancing*. Oracle RAC features are available to you simply by connecting to an Oracle RAC system with the Oracle driver. There is no additional configuration required.

Failover and client load balancing can be used with an Oracle RAC system, but they are not specifically part of Oracle RAC. These features also can be used with any other database supported by DataDirect Connect *for JDBC*. See “[Using Failover](#)” on page 58 and “[Using Client Load Balancing](#)” on page 65 for an overview.

See “[Configuring Failover](#)” on page 365 for information about using failover with Oracle databases. See “[Using tnsnames.ora Files](#)” on page 319 for information about configuring failover and client load balancing using a tnsnames.ora file.

---

# Using tnsnames.ora Files

The tnsnames.ora file is used to map connection information for each Oracle service to a logical alias. The Oracle driver allows you to retrieve basic connection information from a tnsnames.ora file, including:

- Oracle server name and port
- Oracle System Identifier (SID) or Oracle service name
- Server process type (shared or dedicated)
- Failover instructions
- Client load balancing instructions
- Data encryption instructions

In a tnsnames.ora file, connection information for an Oracle service is associated with an alias, or Oracle net service name. Each net service name entry contains connect descriptors that define listener and service information. The following example in [Figure 6-1](#) shows connection information in a tnsnames.ora file configured for the net service name entries, FITZGERALD.SALES and ARMSTRONG.ACCT.

---

**Figure 6-1. tnsnames.ora Example**

---

```
FITZGERALD.SALES =
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = server1)(PORT = 1521))
  (CONNECT_DATA =
    (SID = ORCL)
  )
)
ARMSTRONG.ACCT =
(DESCRIPTION =
(ADDRESS_LIST=
  (FAILOVER = on)
  (LOAD_BALANCE = on))
```

```
(ADDRESS= (PROTOCOL = TCP)(HOST = server1)(PORT = 1521))
(ADDRESS= (PROTOCOL = TCP)(HOST = server2)(PORT = 1521))
(ADDRESS= (PROTOCOL = TCP)(HOST = server3)(PORT = 1521))
)
(CONNECT_DATA=
  (SERVICE_NAME = acct.us.yourcompany.com)
  (FAILOVER_MODE =
    (BACKUP=server2)
    (TYPE=select)
    (METHOD=preconnect)
    (RETRIES=20)
    (DELAY=3)
  )
)
```

---

Using this example, if the Oracle driver referenced the Oracle net service name entry FITGERALD.SALES, the driver would connect to the Oracle database instance identified by the Oracle SID ORCL (SID=ORCL). Similarly, if the Oracle driver referenced ARMSTRONG.ACCT, the driver would connect to the Oracle database identified by the service name acct.us.yourcompany.com (SERVICE\_NAME=acct.us.yourcompany.com). In addition, the driver would enable failover (FAILOVER=on) and client load balancing (LOAD\_BALANCE=on).

Typically, a tnsnames.ora file is installed when you install an Oracle database. By default, the tnsnames.ora file is located in the ORACLE\_HOME\network\admin directory on Windows and the \$ORACLE\_HOME/network/admin directory on UNIX.

## Connecting to the Database

To retrieve connection information from an Oracle tnsnames.ora file with the Oracle driver, you must inform the driver which tnsnames.ora file (using the TNSNamesFile property) and Oracle service name entry (using the TNSServerName property) to use so that the driver can reference the correct connection information. For example, the following connection URL:

```
jdbc:datadirect:oracle:TNSNamesFile=c:\\oracle92\\NETWORK\\ADMIN\\tnsnames.ora;  
TNSServerName=FITZGERALD.SALES
```

specifies the path and filename of the tnsnames.ora file (TNSNamesFile=c:\\oracle92\\NETWORK\\ADMIN\\tnsnames.ora) and the net service name entry (TNSServerName=FITZGERALD.SALES) to use for the connection.

### NOTES:

- The connection URL does not specify the server name and port of the database server; that information is specified in the tnsnames.ora file referenced by the TNSNamesFile property.
- If coding a path on Windows to the tnsnames.ora file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example:  
`TNSNamesFile=c:\\oracle92\\\\NETWORK\\\\ADMIN\\\\tnsnames.ora.`

If using tnsnames.ora files with a Security Manager on a Java 2 Platform, read permission must be granted to the tnsnames.ora file. See ["Granting Access to Oracle tnsnames.ora Files" on page 79](#) for an example.

## Configuring the tnsnames.ora File

If using a tnsnames.ora file to retrieve connection information, do not specify the following connection properties to prevent connection information conflicts:

AlternateServers	ServiceName
EncryptionMethod	PortNumber
LoadBalancing	ServerType
ServerName	SID

If any of these properties are specified in addition to the TNSNamesFile and TNSServerName properties, the driver throws an exception. For example, if the net service name entry ARMSTRONG.ACCT specifies the LOAD\_BALANCE parameter as shown in the following example:

```
ARMSTRONG.ACCT =
(DESCRIPTION =
(ADDRESS_LIST=
  (FAILOVER = on)
  (LOAD_BALANCE = on)
  (ADDRESS= (PROTOCOL = TCP)(HOST = server1)(PORT = 1521))
  (ADDRESS= (PROTOCOL = TCP)(HOST = server2)(PORT = 1521))
  (ADDRESS= (PROTOCOL = TCP)(HOST = server3)(PORT = 1521))
)
```

and you specify the LoadBalancing property in the driver connection URL as shown in the following example, the driver throws an exception.

```
jdbc:datadirect:oracle:TNSNamesFile=c:\\oracle92\\NETWORK\\ADMIN\\
tnsnames.ora;TNSServerName=FITZGERALD.SALES;LoadBalancing=true
```

**Table 6-1** lists the Oracle driver properties that correspond to tnsnames.ora connect descriptor parameters. To prevent connection information conflicts, do not specify the listed properties if you use a tnsnames.ora file.

---

**Table 6-1. Oracle Driver Property Mappings to tnsnames.ora Connect Descriptor Parameters**

---

Oracle Driver Property	tnsnames.ora Attribute
AlternateServers = <i>servers_list</i>	<p>ADDRESS_LIST = <i>servers_list</i></p> <p>The ADDRESS_LIST parameter contains connection information for one or multiple servers, using the ADDRESS parameter to specify the primary and alternate servers. For example:</p> <pre>(ADDRESS_LIST=   (ADDRESS= (PROTOCOL = TCP)(HOST = server1)     (PORT = 1521))   (ADDRESS= (PROTOCOL = TCP)(HOST = server2)     (PORT = 1521))   (ADDRESS= (PROTOCOL = TCP)(HOST = server3)     (PORT = 1521)) )</pre> <p>The first ADDRESS parameter specifies connection information for the primary server. The second and third ADDRESS parameter specifies connection information for alternate servers.</p> <p>When multiple servers are specified by the ADDRESS_LIST parameter, connection failover is automatically enabled. If FAILOVER=off, connection failover is disabled. You also can explicitly specify connection failover using the FAILOVER parameter.</p>

---

**Table 6-1. Oracle Driver Property Mappings to tnsnames.ora Connect Descriptor Parameters (cont.)**

---

Oracle Driver Property	tnsnames.ora Attribute
FailoverMode = {connect   extended   select}	<pre>FAILOVER_MODE = {TYPE={session   select   none} [BACKUP=value] [METHOD=basic   preconnect] [RETRIES=value] [DELAY=value] }  The FAILOVER_MODE parameter is specified in the CONNECT_DATA section to provide failover instructions for the driver. The FAILOVER_MODE parameter requires the TYPE parameter, which specifies the type of failover to be used. Other parameters are optional.  (CONNECT_DATA=   (SERVICE_NAME = acct.us.yourcompany.com)   (FAILOVER_MODE =     (BACKUP=server2)     (TYPE=select)     (METHOD=preconnect)     (RETRIES=20)     (DELAY=3)   ) )</pre> <p>The FAILOVER_MODE parameter can only be used to enable extended connection failover or select failover. When multiple servers are specified by the ADDRESS_LIST parameter, connection failover is automatically enabled. If FAILOVER=off, connection failover is disabled. You also can explicitly specify connection failover using the FAILOVER parameter.</p>

**Table 6-1. Oracle Driver Property Mappings to tnsnames.ora Connect Descriptor Parameters (cont.)**

---

Oracle Driver Property	tnsnames.ora Attribute
LoadBalancing = {true   false}	<p>LOAD_BALANCE = {on   off}</p> <p>If LOAD_BALANCE=on, enables client load balancing. For example:</p> <pre>(ADDRESS_LIST=   (LOAD_BALANCE=on)   (ADDRESS= (PROTOCOL = TCP)(HOST = server1)     (PORT = 1521))   (ADDRESS= (PROTOCOL = TCP)(HOST = server2)     (PORT = 1521))   (ADDRESS= (PROTOCOL = TCP)(HOST = server3)     (PORT = 1521)) )</pre> <p>If the LOAD_BALANCE parameter is unspecified or LOAD_BALANCE=off, client load balancing is disabled.</p>
PortNumber = <i>port</i>	<p>PORt = <i>port</i></p> <p>The ADDRESS_LIST parameter contains connection information for one or multiple servers, using the ADDRESS parameter to specify the primary and alternate servers. The PORT parameter is used within the ADDRESS parameter to specify the port number for each server entry. For example:</p> <pre>(ADDRESS_LIST=   (ADDRESS= (PROTOCOL = TCP)(HOST = server1)     (PORT = 1521))   ... )</pre> <p>A port of 1521, the default port number when installing an Oracle database, is specified for server1.</p>

---

**Table 6-1. Oracle Driver Property Mappings to *tnsnames.ora* Connect Descriptor Parameters (cont.)**

---

Oracle Driver Property	<i>tnsnames.ora</i> Attribute
EncryptionMethod= {noEncryption   SSL}	<p>PROTOCOL={TCP   TCPS}</p> <p>The ADDRESS_LIST parameter contains connection information for one or multiple servers, using the ADDRESS parameter to specify the primary and alternate servers. The PROTOCOL parameter is used within the ADDRESS parameter to specify the network protocol to be used. It also is used to specify whether data is encrypted and decrypted when transmitted over the network between the driver and the server.</p> <p>For example, the following entry specifies that the TCP/IP protocol will be used with no encryption:</p> <pre>(ADDRESS_LIST=   (ADDRESS= (PROTOCOL = TCP)(HOST = server1)     (PORT = 1521))   ... )</pre> <p>A port of 1521, the default port number when installing an Oracle database, is specified for server1.</p> <p>The following entry specifies that the TCP/IP protocol will be used with SSL encryption:</p> <pre>(ADDRESS_LIST=   (ADDRESS= (PROTOCOL = TCPS)(HOST = server1)     (PORT = 2484))   ... )</pre> <p>A port of 2484, the port number recommended by Oracle for SSL, is specified for server1.</p> <p>NOTE: Truststore information must still be specified using either the TrustStore and TrustStorePassword properties or Java system properties. Optionally, you can specify the ValidateServerCertificate and HostNameInCertificate properties.</p>

**Table 6-1. Oracle Driver Property Mappings to tnsnames.ora Connect Descriptor Parameters (cont.)**

---

Oracle Driver Property	tnsnames.ora Attribute
ServerName = <i>server_name</i>	<p>HOST = <i>server_name</i></p> <p>The ADDRESS_LIST parameter contains connection information for one or multiple servers, using the ADDRESS parameter to specify the primary and alternate servers. The HOST parameter is used within the ADDRESS parameter to specify the server name for each server entry. The server entry can be an IP address or a server name. For example:</p> <pre>(ADDRESS_LIST=   (ADDRESS= (PROTOCOL = TCP)(HOST = server1)             (PORT = 1521))   ... )</pre> <p>The server name server1 is specified in the first server entry.</p>
ServerType = {shared   dedicated}	<p>SERVER = {shared   dedicated}.</p> <p>If SERVER=shared is specified in the CONNECT_DATA parameter in the tnsnames.ora file, the server process (UNIX) or thread (Windows) to be used is retrieved from a pool. For example:</p> <pre>(CONNECT_DATA=   (SERVER=shared) )</pre> <p>When SERVER=shared, this setting allows there to be fewer processes than the number of connections, reducing the need for server resources.</p> <p>When SERVER=dedicated, a server process is created to service only that connection. When that connection ends, so does the process (UNIX) or thread (Windows).</p>

---

**Table 6-1. Oracle Driver Property Mappings to tnsnames.ora Connect Descriptor Parameters (cont.)**

---

Oracle Driver Property	tnsnames.ora Attribute
ServiceName = <i>service_name</i>	<p><b>SERVICE_NAME = <i>service_name</i></b></p> <p>The database service name that specifies the database used for the connection. The service name is a string that is the global database name—a name that typically comprises the database name and domain name. For example:</p> <pre>sales.us.acme.com</pre> <p>The service name is specified in the CONNECT_DATA parameter. For example:</p> <pre>(CONNECT_DATA=   (SERVICE_NAME=sales.us.acme.com) )</pre> <p>This parameter is mutually exclusive with the SID attribute and is useful to specify connections to an Oracle Real Application Clusters (RAC) system rather than a specific Oracle instance.</p>
SID = <i>SID</i>	<p><b>SID = <i>SID</i></b></p> <p>The Oracle System Identifier (SID) that refers to the instance of the Oracle database running on the server. The default Oracle SID that is configured when installing your Oracle database software is ORCL. The SID is specified in the CONNECT_DATA parameter. For example:</p> <pre>(CONNECT_DATA=   (SID=ORCL) )</pre> <p>This parameter is mutually exclusive with the SERVICE_NAME attribute.</p>

---

For more information about configuring tnsnames.ora files, refer to your Oracle documentation.

## Connecting to Oracle Instances Running in Restricted Mode

An Oracle instance may be running in restricted mode, which allows access only to users who have the RESTRICTED SESSION system privilege. To connect to an instance running in restricted mode, add the `UR = A` clause to the `CONNECT_DATA` parameter of the `tnsnames.ora` file as shown.

```
...
(CONNECT_DATA=
  (SERVICE_NAME = acct.us.yourcompany.com)
  (UR = A)
)
...
```

---

## Data Types

[Table 6-2](#) lists the data types supported by the Oracle driver and describes how they are mapped to the JDBC data types.

---

***Table 6-2. Oracle Data Types***

---

Oracle Data Type	JDBC Data Type
BFILE	BLOB
BINARY_DOUBLE <sup>1</sup>	DOUBLE
BINARY_FLOAT <sup>1</sup>	REAL
BLOB	BLOB
CHAR	CHAR
CLOB	CLOB
DATE	TIMESTAMP
FLOAT(n)	DOUBLE
LONG	LONGVARCHAR
LONG RAW	LONGVARBINARY
NCHAR	CHAR NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NCHAR (if using Java SE 6) or CHAR (if using another JVM).
NCLOB	CLOB NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NCLOB (if using Java SE 6) or CLOB (if using another JVM).
NUMBER	DECIMAL
NUMBER (p, s)	DECIMAL

**Table 6-2. Oracle Data Types (cont.)**

Oracle Data Type	JDBC Data Type
NVARCHAR2	VARCHAR NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NVARCHAR (if using Java SE 6) or VARCHAR (if using another JVM).
RAW	VARBINARY
TIMESTAMP <sup>2</sup>	TIMESTAMP
TIMESTAMP WITH LOCAL TIME ZONE <sup>2</sup>	TIMESTAMP
TIMESTAMP WITH TIME ZONE <sup>2</sup>	TIMESTAMP
UROWID <sup>2</sup>	VARCHAR
VARCHAR2 <sup>2</sup>	VARCHAR
XMLType <sup>2</sup>	CLOB NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: SQLXML (if using Java SE 6) or CLOB (if using another JVM).

1. Supported only for Oracle 10g and higher.  
2. Supported only for Oracle 9i and higher.

See “[Returning and Inserting/Updating XML Data](#)” on page 335 for more information about the XMLType data type. See [Appendix D “getTypeInfo” on page 631](#) for a description of the data types returned by the `getTypeInfo()` method.

---

# Using Oracle Date/Time Data Types

Oracle 9*i* and higher supports the following date/time data types:

- TIMESTAMP
- TIMESTAMP WITH LOCAL TIME ZONE
- TIMESTAMP WITH TIME ZONE

## Date/Time Session Parameters

To understand how the Oracle driver supports the date/time data types, you first must understand the values that the Oracle driver assigns to the Oracle date/time session parameters. [Table 6-3](#) describes the session parameters the Oracle driver sets at connection time.

---

**Table 6-3. Oracle Date/Time Session Parameters**

---

Session Parameter	Description
TIME_ZONE	The Oracle session time zone. The Oracle driver sets the time zone to the current time zone as reported by the JVM.
NLS_TIMESTAMP_FORMAT	The default timestamp format. The Oracle driver uses the JDBC timestamp escape format: YYYY-MM_DD HH24:MI:SS.FF
NLS_TIMESTAMP_TZ_FORMAT	The default timestamp with time zone format. The Oracle driver uses the JDBC timestamp escape format with the time zone field appended: YYYY-MM_DD HH24:MI:SS.FF TZH:TZM

---

## TIMESTAMP Data Type

The Oracle TIMESTAMP data type is mapped to the JDBC TIMESTAMP data type.

## TIMESTAMP WITH LOCAL TIME ZONE Data Type

The Oracle TIMESTAMP WITH LOCAL TIME ZONE data type is mapped to the TIMESTAMP JDBC data type.

When retrieving TIMESTAMP WITH LOCAL TIME ZONE columns, the value returned to the user is converted to the time zone specified by the TIME\_ZONE session parameter.

When setting TIMESTAMP WITH LOCAL TIME ZONE columns:

- Using a timestamp (using PreparedStatement.setTimestamp, for example), the value set is converted to the time zone specified by the TIME\_ZONE session parameter.
- Using a string (using PreparedStatement.setString, for example), the string is passed as-is to the server. The supplied string must be in the format specified by the NLS\_TIMESTAMP\_TZ\_FORMAT session parameter. If not, the Oracle server generates an error when it attempts to convert the string to the TIMESTAMP WITH LOCAL TIME ZONE type.

## TIMESTAMP WITH TIME ZONE Data Type

By default, the Oracle TIMESTAMP WITH TIME ZONE data type is mapped to the VARCHAR JDBC data type.

When retrieving TIMESTAMP WITH TIME ZONE values as a string (using resultSet.getString, for example), the value is returned as

the string representation of the timestamp including time zone information. The string representation is formatted as:

'YYYY-MM-DD HH24:MI:SS.FF TZH:TZM'

where:

- *YYYY* is the 4-digit year.
- *MM* is the month.
- *DD* is the day.
- *HH24* is the hour in 24-hour format.
- *MI* is the minutes.
- *SS* is the seconds.
- *FF* is the fractional seconds.
- *TZH* is the time zone hours and *TZM* is the time zone minutes.  
The time zone is represented as the difference in hours and minutes between the time zone and GMT.

By default, retrieving `TIMESTAMP WITH TIME ZONE` values as a timestamp (using `resultSet.getTimeStamp`, for example) is not supported because the time zone information stored in the database would be lost when the data is converted to a timestamp. To provide backward compatibility with existing applications, you can use the `FetchTSWTZasTimestamp` property to allow `TIMESTAMP WITH TIME ZONE` values to be returned as a timestamp. The default value of the `FetchTSWTSasTimestamp` property is false, which disables retrieving `TIMESTAMP WITH TIME ZONE` values as timestamps.

When setting `TIMESTAMP WITH TIME ZONE` columns:

- Using a timestamp (using `PreparedStatement.setTimestamp()`, for example), the value set is converted to the time zone specified by the `TIME_ZONE` session parameter.
- Using a string (using `PreparedStatement.setString()`, for example), the string is passed as-is to the server. The supplied string must be in the format specified by the `NLS_TIMESTAMP_TZ_FORMAT` session parameter. If not, the

Oracle server generates an error when it attempts to convert the string to the TIMESTAMP WITH TIME ZONE type.

---

## Returning and Inserting/Updating XML Data

For Oracle 9*i* and higher, the Oracle driver supports the Oracle XMLType data type. Which JDBC data type the XMLType data type is mapped to depends on whether the JDBCBehavior property is set. If JDBCBehavior=0, XML data is mapped to SQLXML or CLOB, depending on which JVM your application is using. The driver maps the XMLType data type to the JDBC SQLXML data type if your application is using Java SE 6. If your application is using another JVM, the driver maps the XMLType data type to the JDBC CLOB data type.

### Returning XML Data

The driver can return XML data as character data. For example, given a database table defined as:

```
CREATE TABLE xmlTable (id int, xmlCol XMLType NOT NULL)
```

the driver can return the XML data as character data using the following code:

```
String sql="SELECT xmlCol FROM xmlTable";
ResultSet rs=stmt.executeQuery(sql)
String charXML=rs.getString(1)
```

The result set column is described with a column type of CLOB and the column type name is xmlType.

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

```
ResultSet.getString()  
ResultSet.getCharacterStream()  
ResultSet.getBlob()  
CallableStatement.getString()  
CallableStatement.getBlob()
```

The driver converts the XML data returned from the database server from the character set 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 character set encoding used by the database server 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

When inserting to or updating `XMLType` columns, the data to be inserted or updated must be the `XMLType` data type. Oracle provides the `xmltype()` function to construct an `XMLType` data object. The `xmlData` argument of the `xmltype()` function can be specified as a string literal or a parameter marker. If specified as a

parameter marker, the parameter value can be set using the following methods:

```
PreparedStatement.setString()  
PreparedStatement.setCharacterStream()  
PreparedStatement.setClob()  
PreparedStatement.setAsciiStream()
```

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.

The following code inserts data into an XMLType column using a statement with a string literal as the xmlData argument of the `xmltype()` function:

```
// Insert xml data as a literal  
String sql = "INSERT INTO XMLTable VALUES (1, xmltype(' " +  
    "<emp><empNo>123</empNo><empName>Mark</empName></emp>' ))";  
  
Statement stmt = con.createStatement();  
stmt.executeUpdate(sql);
```

The following code inserts data into an XMLType column using a prepared statement:

```
// Insert xml data as a String parameter  
String xmlStr = "<emp><empNo>234</empNo><empName>Trish</empName></emp>";  
String sql = "INSERT INTO XMLTable VALUES (?, xmltype(?))";  
  
PreparedStatement prepStmt = con.prepareStatement(sql);  
prepStmt.setInt(1, 2);  
prepStmt.setString(2, xmlStr);  
prepStmt.executeUpdate();
```

## REF CURSOR Data Type

REF CURSOR is the Oracle data type for a cursor variable. Because JDBC does not support a cursor variable data type, the Oracle driver returns REF CURSOR output parameters and return values to the application as result sets. The Oracle driver automatically converts the REF CURSOR data to a result set, which can be returned using `getResultSet()` or `getMoreResults()`. Because REF CURSOR data is returned as result sets and not as output parameters, REF CURSOR output parameters are not included in results from `DatabaseMetaData.getProcedureColumns()` calls.

In your application, omit any parameter markers for the REF CURSOR and do not declare an output parameter for the REF CURSOR as shown in the following examples. These examples reference the following stored procedure definition:

```
CREATE PACKAGE foo_pkg AS
  TYPE EmpCurTyp IS REF CURSOR RETURN fooTbl%ROWTYPE;
  PROCEDURE selectEmployeeManager(empId IN INT, empCursor OUT EmpCurTyp,
    mgrCursor OUT EmpCurTyp);
  FUNCTION selectEmployee2 (empId IN INT) RETURN EmpCurTyp;
END foo_pkg;
```

### Example A: Calling a Stored Procedure That Returns a Single REF CURSOR

```
// Call a function that accepts an input parameter
// and returns a REF CURSOR as the return value. Omit the
// placeholder for the refcursor return value parameter.
// The REF CURSOR is returned as a result set.
sql = "{call foo_pkg.selectEmployee2(?)}";

callStmt = con.prepareCall(sql);
callStmt.setInt(1, 2);
moreResults = callStmt.execute();

while (true) {
  if (moreResults) {
```

```
// Get the result set that represents the REF CURSOR
resultSet = callStmt.getResultSet();
displayResults(resultSet);

resultSet.close();
resultSet = null;

System.out.println();
}

else {

    updateCnt = callStmt.getUpdateCount();
    if (updateCnt == -1) {
        break;
    }
    System.out.println("Update Count: " + updateCnt);
}
moreResults = callStmt.getMoreResults();
}
```

### Example B: Calling a Stored Procedure that Returns Multiple REF CURSORS

```
// Call the stored procedure that accepts an input parameter
// and returns two REF CURSORS. Omit the placeholder for
// REF CURSOR parameters. The REF CURSORS are returned as
// result sets.
sql = "{call foo_pkg.selectEmployeeManager(?)}";

callStmt = con.prepareCall(sql);
callStmt.setInt(1, 2);
moreResults = callStmt.execute();

while (true) {

    if (moreResults) {
```

```
// Get the result set that represents the REF CURSOR
resultSet = callStmt.getResultSet();
displayResults(resultSet);
resultSet.close();
}
else {
    updateCnt = callStmt.getUpdateCount();
    if (updateCnt == -1) {
        break;
    }
}
moreResults = callStmt.getMoreResults();
}
```

---

## Authentication

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See “[Authentication](#)” on page 67 for an overview.

The Oracle driver supports the following methods of authentication:

- User ID/password authentication authenticates the user to the database using a database user name and password specified 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 and MIT Kerberos.

- NTLM authentication is a single sign-on OS authentication method for Windows environments. This method provides authentication from Windows clients only and requires minimal configuration.
- Client authentication uses the user ID of the user logged onto the system on which the driver is running to authenticate the user to the database. The Oracle database server relies on the client to authenticate the user and does not provide additional authentication.

NOTE: Because the database server does not authenticate the user when client authentication is used, use this method of authentication if you can guarantee that only trusted clients can access the database server.

Except for NTLM authentication, which provides authentication for Windows clients only, these authentication methods provide authentication when the driver is running on any supported platform.

The AuthenticationMethod connection property controls which authentication mechanism the driver uses when establishing connections. See ["Using the AuthenticationMethod Property" on page 341](#) for information about setting the value for this property.

## Using the AuthenticationMethod Property

The AuthenticationMethod connection property controls which authentication mechanism the driver uses when establishing connections. When AuthenticationMethod=auto (the default), the driver uses user ID/password, Kerberos, or NTLM

authentication when establishing a connection based on the following criteria:

- If a user ID and password is specified, the driver uses user ID/password authentication when establishing a connection. The User property provides the user ID. The Password property provides the password.
- If a user ID and password is not specified and the driver is not running on a Windows platform, the driver uses Kerberos authentication when establishing a connection.
- If a user ID and password is not specified and the driver is running on a Windows platform, the driver uses NTLM authentication when establishing a connection if the driver can load the DLL required for NTLM authentication. If the driver cannot load the DLL, the driver uses Kerberos authentication.

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

When AuthenticationMethod=kerberosUIDPassword, the driver first uses Kerberos when establishing a connection. Next, the driver reauthenticates the user using user ID/password authentication. The User property provides the user ID. The Password property provides the password. If a user ID and password are not specified, the driver throws an exception. If either Kerberos or user ID/password authentication fails, the connection attempt fails and the driver throws an exception.

When AuthenticationMethod=ntlm, the driver uses NTLM authentication when establishing a connection if the driver can load the DLL required for NTLM authentication. If the driver cannot load the DLL, the driver throws an exception. The driver ignores any values specified by the User and Password properties.

When AuthenticationMethod=client, the driver uses client authentication when establishing a connection. The Oracle

database server relies on the client to authenticate the user and does not provide additional authentication. The driver ignores any values specified by the User and Password properties.

When AuthenticationMethod=userIdPassword, the driver uses user ID/password authentication when establishing a connection. The User property provides the user ID. The Password property provides the password. If a user ID is not specified, the driver throws an exception.

## Configuring User ID/Password Authentication

- 1 Set the AuthenticationMethod property to auto or userIdPassword. See ["Using the AuthenticationMethod Property" on page 341](#) for more information about setting a value for this property.
- 2 Set the User property to provide the user ID.
- 3 Set the Password property to provide the password.

## Configuring Kerberos Authentication

This section provides requirements and instructions for configuring Kerberos authentication for the Oracle driver.

### ***Product Requirements***

Verify that your environment meets the requirements listed in [Table 6-4](#) before you configure the driver for Kerberos authentication.

**Table 6-4. Kerberos Authentication Requirements for the Oracle Driver**

<b>Component</b>	<b>Requirements</b>
Database server	<p>The database server must be administered by the same domain controller that administers the client and must be running one of the following databases:</p> <ul style="list-style-type: none"> <li>■ Oracle 11g (R1 and R2)</li> <li>■ Oracle 10g (R1 and R2)</li> <li>■ Oracle 9i (R2)</li> </ul> <p>In addition, Oracle Advanced Security is required.</p>
Kerberos server	<p>The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos KDC.</p> <p>Network authentication must be provided by one of the following methods:</p> <ul style="list-style-type: none"> <li>■ Windows Active Directory on one of the following operating systems: Windows Server 2003 or Windows 2000 Server Service Pack 3 or higher</li> <li>■ MIT Kerberos 1.4.2 or higher</li> </ul>
Client	<p>The client must be administered by the same domain controller that administers the database server. In addition, J2SE 1.4.2 or higher must be installed.</p>

## **Configuring the Driver**

During installation, DataDirect Connect for JDBC installs the following files required for Kerberos authentication in the /lib subdirectory of your DataDirect Connect for JDBC installation directory:

- krb5.conf is a Kerberos configuration file containing values for the Kerberos realm and the KDC name for that realm. DataDirect Connect for JDBC installs a generic file that you must modify for your environment.

- JDBCDriverLogin.conf file is a Java Authentication and Authorization Service (JAAS) login module for Kerberos authentication. This file is configured to load automatically unless the java.security.auth.login.config system property is set to load another configuration file.

NOTE: Do not modify the JDBCDriverLogin.conf file.

#### To configure the driver:

- 1 Set the driver's AuthenticationMethod property to auto (the default) or kerberos. See ["Using the AuthenticationMethod Property" on page 341](#) for more information about setting a value for this property.
- 2 Modify the krb5.conf file to contain your Kerberos realm name and the KDC name for that Kerberos realm by editing the file with a text editor or by specifying the system properties, java.security.krb5.realm and java.security.krb5.kdc.

NOTE: In Windows Active Directory, the Kerberos realm name is the Windows domain name and the KDC name is the Windows domain controller name.

For example, if your Kerberos realm name is XYZ.COM and your KDC name is kdc1, your krb5.conf file would look like this:

```
[libdefaults]
    default_realm = XYZ.COM

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

If the krb5.conf file does not contain a valid Kerberos realm and KDC name, the following exception is thrown:

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

The krb5.conf file installed by DataDirect Connect for JDBC is configured to load automatically unless the java.security.krb5.conf system property is set to point to another Kerberos configuration file.

- 3 If using Kerberos authentication with a Security Manager on a Java 2 Platform, you must grant security permissions to the application and driver. See ["Permissions for Kerberos Authentication" on page 79](#) for an example.

## Specifying User Credentials for Kerberos Authentication (Delegation of Credentials)

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

Many application servers or Web servers act on behalf of the client user logged on the machine on which the application is running, rather than the server user. If you want the driver to use a set of user credentials other than the operating system user name and password, include code in your application to obtain

and pass a jjavax.security.auth.Subject used for authentication as shown in the following example.

```
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import java.sql.*;

// The following code creates a javax.security.auth.Subject instance
// used for authentication. Refer to the Java Authentication
// and Authorization Service documentation for details on using a
// LoginContext to obtain a Subject.

LoginContext lc = null;
Subject subject = null;

try {

    lc = new LoginContext("JaasSample", new TextCallbackHandler());
    lc.login();
    subject = lc.getSubject();
}
catch (Exception le) {
    ... // display login error
}

// This application passes the javax.security.auth.Subject
// to the driver by executing the driver code as the subject

Connection con =
    (Connection) Subject.doAs(subject, new PrivilegedExceptionAction() {

        public Object run() {

            Connection con = null;
            try {

                Class.forName("com.ddtek.jdbc.oracle.OracleDriver");
                String url = "jdbc:datadirect:oracle://myServer:1521";
                con = DriverManager.getConnection(url);
            }
            catch (Exception except) {
```

```
    ... //log the connection error
        Return null;
    }

    return con;
}
});

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

... // do something with the results
```

## Obtaining a Kerberos Ticket Granting Ticket

To use Kerberos authentication, the application user first must obtain a Kerberos Ticket Granting Ticket (TGT) from the Kerberos server. The Kerberos server verifies the identity of the user and controls access to services using the credentials contained in the TGT.

If the application uses Kerberos authentication from a Windows client and the Kerberos authentication is provided by Windows Active Directory, the application user is not required to log onto the Kerberos server and explicitly obtain a TGT. Windows Active Directory automatically obtains a TGT for the user.

The application user must explicitly obtain a TGT in the following cases:

- If the application uses Kerberos authentication from a UNIX or Linux client
- If the application uses Kerberos authentication from a Windows client and Kerberos authentication is provided by MIT Kerberos

To explicitly obtain a TGT, the user must log onto the Kerberos server using the kinit command. For example, the following command requests a TGT from the server with a lifetime of 10 hours, which is renewable for 5 days:

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

where *user* is the application user.

Refer to your Kerberos documentation for more information about using the kinit command and obtaining TGTs for users.

# Configuring NTLM Authentication

This section provides requirements and instructions for configuring NTLM authentication for the Oracle driver.

## ***Product Requirements***

Verify that your environment meets the requirements listed in [Table 6-5](#) before you configure the driver for NTLM authentication.

---

***Table 6-5. NTLM Authentication Requirements for the Oracle Driver***

---

Component	Requirements
Database server	The database server must be administered by the same domain controller that administers the client and must be running one of the following databases: <ul style="list-style-type: none"> <li>■ Oracle 11g (R1 and R2)</li> <li>■ Oracle 10g (R1 and R2)</li> <li>■ Oracle 9i (R1 and R2)</li> <li>■ Oracle 8i (8.1.6 and 8.1.7)</li> </ul>
Domain controller	The domain controller must administer both the database server and the client. Network authentication must be provided by NTLM on one of the following operating systems: <ul style="list-style-type: none"> <li>■ Windows Server 2003</li> <li>■ Windows 2000 Server Service Pack 3 or higher</li> </ul>
Client	The client must be administered by the same domain controller that administers the database server and must be running on one of the following operating systems: <ul style="list-style-type: none"> <li>■ Windows Vista</li> <li>■ Windows Server 2003</li> <li>■ Windows XP Service Pack 1 or higher</li> <li>■ Windows 2000 Service Pack 4 or higher</li> <li>■ Windows NT 4.0</li> </ul> In addition, J2SE 1.4 or higher must be installed.

---

## Configuring the Driver

DataDirect Connect for JDBC provides the following NTLM authentication DLLs:

- DDJDBCAuthxx.dll (32-bit)
- DDJDBC64Authxx.dll (Itanium 64-bit)
- DDJDBCx64Authxx.dll (AMD64 and Intel EM64T 64-bit)

where *xx* is a two-digit number.

The DLLs are located in the *install\_dir/lib* directory (where *install\_dir* is your DataDirect Connect for JDBC installation directory). If the application using NTLM authentication is running in a 32-bit JVM, the driver automatically uses DDJDBCAuthxx.dll. Similarly, if the application is running in a 64-bit JVM, the driver uses DDJDBC64Authxx.dll or DDJDBCx64Authxx.dll.

### To configure the driver:

- 1 Set the AuthenticationMethod property to auto or ntlm. See ["Using the AuthenticationMethod Property" on page 341](#) for more information about setting a value for this property.
- 2 By default, the driver looks for the NTLM authentication DLLs in a directory on the Windows system path defined by the PATH environment variable. If you install the driver in a directory that is not on the Windows system path, perform one of the following actions to ensure the driver can load the DLLs:
  - Add the *install\_dir/lib* directory to the Windows system path, where *install\_dir* is the DataDirect Connect for JDBC installation directory.
  - Copy the NTLM authentication DLLs from *install\_dir/lib* to a directory that is on the Windows system path, where *install\_dir* is the DataDirect Connect for JDBC installation directory.

- Set the LoadLibraryPath property to specify the location of the NTLM authentication DLLs. For example, if you install the driver in a directory named "DataDirect" that is not on the Windows system path, you can use the LoadLibraryPath property to specify the directory containing the NTLM authentication DLLs:

```
jdbc:datadirect:oracle://server3:1521;  
ServiceName=ORCL;LoadLibraryPath=C:\DataDirect\lib;  
User=test;Password=secret
```

- 3 If using NTLM authentication with a Security Manager on a Java 2 Platform, security permissions must be granted to allow the driver to establish connections. See ["Permissions for Establishing Connections" on page 77](#) for an example.

## Configuring Client Authentication

Set the AuthenticationMethod property to client. See ["Using the AuthenticationMethod Property" on page 341](#) for more information about setting a value for this property.

---

## Data Encryption

The Oracle driver supports SSL encryption for the following databases:

- Oracle 11g (R1 and R2)
- Oracle 10g (R1 and R2)
- Oracle 9*i* (R1 and R2)
- Oracle 8*i* (8.1.6 and 8.1.7)

In addition, Oracle Advanced Security is required.

SSL secures the integrity of your data by encrypting information and providing authentication. See “[Data Encryption Across the Network](#)” on page 69 for an overview.

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.

See “[Using tnsnames.ora Files](#)” on page 319 for information about configuring a tnsnames.ora file for SSL encryption.

#### To configure SSL encryption:

- 1 Set the EncryptionMethod property to SSL.
- 2 Specify the location and password of the truststore file used for SSL server authentication. Either set the TrustStore and TrustStore properties or their corresponding Java system properties (javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword, respectively).
- 3 To validate certificates sent by the database server, set the ValidateServerCertificate property to true.
- 4 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.
- 5 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 KeyStore 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.

---

## Reauthentication

The Oracle driver supports reauthentication for Oracle 8.1.6 and higher. The user performing the switch must have been granted the database permission CONNECT THROUGH.

See ["Using Reauthentication" on page 55](#) for an introduction to reauthentication. See [Appendix I "Connection Pool Manager" on page 789](#) for information about using reauthentication with the DataDirect Connection Pool Manager.

**NOTE:** Before performing reauthentication, applications must ensure that any statements or result sets created as one user are closed before switching the connection to another user.

Your application can use the setCurrentUser() method in the ExtConnection interface to switch a user on a connection. Optionally, the setCurrentUser() method accepts driver-specific reauthentication options; however, no options are supported for the Oracle driver. See ["ExtConnection Interface" on page 613](#) for more information about the setCurrentUser() method.

---

## Client Information for Connections

The Oracle driver allows applications to store and return the following types of client information associated with a particular connection:

- Name of the application
- User ID
- Host name of the client
- Additional accounting information, such as an accounting ID
- Product name and version of the Oracle driver

This information can be used for database administration and monitoring purposes. See [Appendix C “Client Information for Connections” on page 621](#) for details.

---

## SQL Escape Sequences

See [Appendix F “SQL Escape Sequences for JDBC” on page 717](#) for information about the SQL escape sequences supported by the Oracle driver.

---

## Isolation Levels

The Oracle driver supports the Read Committed and Serializable isolation levels. The default is Read Committed.

## Using Scrollable Cursors

The Oracle driver supports scroll-sensitive result sets, scroll-insensitive result sets, and updatable result sets.

NOTE: When the Oracle 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

To use JDBC distributed transactions through JTA, one of the following databases is required:

- Oracle 11g
- Oracle 10g
- Oracle 9i
- Oracle 8.1.7

---

## Batch Inserts and Updates

The Oracle driver provides two mechanisms for supporting batch operations:

- The first mechanism uses native Oracle batch functionality. This mechanism typically is the faster of the two mechanisms, but it is not compliant with the JDBC specification because the native Oracle functionality returns a single update count for all operations in the batch. Because that single update count cannot be resolved into individual update counts for the driver, the driver returns a value of SUCCESS\_NO\_INFO (-2)

for each entry in the update count array. The JDBC specification requires individual update counts to be returned for each operation in the batch.

- The second mechanism uses code that resides in the driver to execute the batch operations and complies with the JDBC specification, but it is slower than using native Oracle batch functionality.

The `BatchPerformanceWorkaround` property determines which batch mechanism is used. If the value of the `BatchPerformanceWorkaround` property is true, the native Oracle batch mechanism is used; otherwise, the JDBC-compliant mechanism is used. The default value of the `BatchPerformanceWorkaround` property is false.

---

## Parameter Metadata Support

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

### Insert and Update Statements

The Oracle driver supports returning parameter metadata for the following forms of Insert and Update statements:

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

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

## Select Statements

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

Parameter metadata can be returned for a Select statement if one of the following conditions is true:

- The statement contains a predicate value expression that can be targeted against the source tables in the associated FROM clause. For example:

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

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

- The statement contains a predicate value expression part that is a nested query. The nested query's metadata must describe a single column. For example:

```
SELECT * FROM foo WHERE (SELECT x FROM y  
WHERE z = 1) < ?
```

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

```
SELECT col1, col2 FROM foo WHERE col1 = ? and col2 > ?
```

```
SELECT ... WHERE colname = (SELECT col2 FROM t2  
WHERE col3 = ?)
```

```
SELECT ... WHERE colname LIKE ?
```

```
SELECT ... WHERE colname BETWEEN ? and ?
```

```
SELECT ... WHERE colname IN (?, ?, ?)
```

```
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE col1 < ?)
```

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

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

Joins are supported. For example:

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

Fully qualified names and aliases are supported. For example:

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

When parameter metadata is requested for a column defined as NUMBER with no precision and scale argument, the driver returns a precision of 0 and a scale of 0 to indicate that the precision and scale of the column are unknown.

## Stored Procedures

The Oracle driver does not support returning parameter metadata for stored procedure arguments.

---

## ResultSet MetaData Support

If your application requires table name information, the Oracle driver can return table name information in ResultSet metadata for Select statements. By setting the `ResultSetMetaDataOptions` property to 1, the Oracle 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 Oracle 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 Oracle 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 Oracle 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 Oracle driver also can return schema name and catalog name information when the `ResultSetMetaData.getSchemaName()` and `ResultSetMetaData.getCatalogName()` methods are called if the driver can determine that information. For example, for the following statement, the Oracle driver returns "test" for the catalog name, "test1" for the schema name, and "foo" for the table name:

```
SELECT * FROM test.test1.foo
```

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

---

## Executing Insert/Update/Delete Statements with a RETURNING Clause

For Oracle 8.1.6 and higher, the Oracle driver supports executing Insert, Update, and Delete statements with the RETURNING clause, which allows your application to return inserted, updated, or deleted values of a row into a variable and eliminate the need to execute additional statements to return this information.

The driver returns the values for each column named in the RETURNING clause as an output parameter. Your application must execute the Insert, Update, or Delete statement with the RETURNING clause using a CallableStatement object. In addition, your application must specify the data type of each returned value using the `CallableStatement.registerOutParameter()` method. The registered data type for a returned value must match the data type of the database column. For example, if the

database column is defined with a JDBC type of CHAR, the data type of the returned value for that column must be registered as Types.CHAR.

The RETURNING clause can return a single row or multiple rows. The method your application uses to retrieve the values of returned columns depends on the number of rows the RETURNING clause returns as shown in the following examples.

#### **Example A: Retrieving a Result Value From an Insert/Update/Delete of a Single Row**

Given the table defined by:

```
CREATE TABLE employees (id int, name varchar(30))
```

You can use the following Insert statement with the RETURNING clause to return the updated ID for Smith:

```
String sql = "INSERT INTO employees VALUES(100, 'Smith')  
    RETURNING id INTO ?";  
CallableStatement callStmt = con.prepareCall(sql);  
callStmt.registerOutParameter(1, Types.INTEGER);  
int updateCnt = callStmt.executeUpdate();  
int newId = callStmt.getInt(1);  
System.out.println("The id of the inserted row is: " + newId);
```

The database server returns a single result value for the requested column. An application can retrieve the result value using any of the following CallableStatement methods: getInt(), getString(), getObject(), and so on. The object type returned by getObject() is based on the data type specified in the registerOutParameter() call for the returned columns. Refer to the JDBC specification for details about JDBC data type to Java object mapping.

### Example B: Retrieving Result Values From an Insert/Update/Delete of Multiple Rows

Given the table defined by:

```
CREATE TABLE employees (id int, name varchar(30))
```

You can use the following Update statement with the RETURNING clause to return all rows with an updated ID value.

```
String sql = "UPDATE employees SET id = id + 1000" +
            "RETURNING id INTO ?";
CallableStatement callStmt = con.prepareCall(sql);
callStmt.registerOutParameter(1, Types.INTEGER);
int updateCnt = callStmt.executeUpdate();
Integer[] newIds = (int []) callStmt.getArray(1).getArray();
for (int index = 0; index < newIds.length; index++) {
    System.out.println("New Id value: " + newIds[index]);
}
```

The database server returns multiple result values for the requested column. An application can retrieve the result values using the CallableStatement.getArray() method.

**NOTE:** If you use the CallableStatement.getxxx() methods to retrieve the result values, the driver only returns the first result value for the requested column.

The data type of the returned array, and the data type of the array elements, match the data type specified in the registerOutParameter() call for the returned column. The elements of the array are an object type. For example, if the application registered the data type of the returned value as Types.INTEGER, the elements of the array are returned as Integer objects. The result set generated by the CallableStatement.getArray() method is a forward-only result set with a result set concurrency of read only. It contains a single column and has a row for each entry in the array.

---

## Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

See <http://www.jcp.org/en/jsr/detail?id=114> for more information about JSR 114.

---

## Auto-Generated Keys Support

The Oracle driver supports retrieving the values of auto-generated keys. An auto-generated key returned by the Oracle 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 that contains parameters:

- When using an Insert statement that contains no parameters, the Oracle 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 that contains parameters, the Oracle 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.

See “[Retrieving Auto-Generated Keys](#)” on page 709 for information about how auto-generated keys can improve performance.

---

## Configuring Failover

1 Specify the primary and alternate servers:

- Specify your primary server using a connection URL or data source.
- Specify one or multiple alternate servers by setting the `AlternateServers` property.

See “[Specifying Primary and Alternate Servers](#)” on page 366.

NOTE: If using failover with an Oracle RAC, the primary server must be a primary node and any alternate server must be a secondary node.

- 2 Choose a failover method by setting the FailoverMode connection property. The default method is connection failover (FailoverMode=connect). See ["Using Failover" on page 58](#) for an overview of each failover method.
- 3 If FailoverMode=extended or FailoverMode=select, set the FailoverGranularity property to specify how you want the driver to behave if exceptions occur while trying to reestablish a lost connection. The default behavior of the driver is to continue with the failover process and post any exceptions on the statement on which they occur (FailoverGranularity=nonAtomic).
- 4 Optionally, configure the connection retry feature. See ["Specifying Connection Retry" on page 369](#).
- 5 Optionally, set the FailoverPreconnect property if you want the driver to establish a connection with the primary and an alternate server at the same time. The default behavior is to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection (FailoverPreconnect=false).

## Specifying Primary and Alternate Servers

Connection information for primary and alternate servers can be specified using either one of the following methods:

- Connection URL through the JDBC Driver Manager
- JDBC data source

For example, the following connection URL for the Oracle driver specifies connection information for the primary and alternate servers using a connection URL:

```
jdbc:datadirect:oracle://server1:1521;ServiceName=TEST;User=test;  
Password=secret;AlternateServers=(server2:1521;ServiceName=TEST2,  
server3:1521;ServiceName=TEST3)
```

In this example:

```
...server1:1521;ServiceName=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=(server2:1521;ServiceName=TEST2,  
server3:1521;ServiceName=TEST3)
```

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

```
OracleDataSource mds = new OracleDataSource();  
mds.setDescription("My OracleDataSource");  
mds.setServerName("server1");  
mds.setPortNumber(1521);  
mds.setServiceName("TEST");  
mds.setUser("test");  
mds.setPassword("secret");  
AlternateServers=(server2:1521;ServiceName=TEST2,  
server3:1521;ServiceName=TEST3)
```

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

For information about specifying connection information for primary and alternate servers using a `tnsnames.ora` file, see ["Using tnsnames.ora Files" on page 319](#).

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 server name of the first alternate database server, `servername2` is the server name of the second alternate database server, and so on. The 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. The port number is 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 1521 is used.

`property=value` is one of the following connection properties: `ServiceName` or `SID`. These connection properties are optional for each alternate server entry and are mutually exclusive. For example:

```
jdbc:datadirect:oracle://server1:1521;ServiceName=TEST;User=test;
Password=secret;AlternateServers=(server2:1521;ServiceName=TEST2,
server3:1521)
```

or

```
jdbc:datadirect:oracle://server1:1521;SID=ORCL;User=test;Password=secret;
AlternateServers=(server2:1521;SID=ORCL2,server3:1521)
```

If you do not specify an optional connection property in an alternate server entry, the connection to that alternate server uses the property specified for the primary server. For example, if you specify `SID=ORCL` for the primary server, but do not specify a `SID` in the alternate server entry as shown in the following URL,

the driver uses the SID specified for the primary server and tries to connect to the ORCL database on the alternate server:

```
jdbc:datadirect:oracle://server1:1521;SID=ORCL;User=test;Password=secret;  
AlternateServers=(server2:1521,server3:1521)
```

## Specifying Connection Retry

Connection retry allows the Oracle 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:oracle://server1:1521;ServiceName=TEST;User=test;  
Password=secret;AlternateServers=(server2:1521;ServiceName=TEST2,  
server3:1521;ServiceName=TEST3);ConnectionRetryCount=2;  
ConnectionRetryDelay=5
```

In this example, if a successful connection is not established on the Oracle 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.

## Failover Properties

[Table 6-6](#) summarizes the connection properties that control how failover works with the Oracle driver. See ["Connection Properties" on page 267](#) for details about configuring each property.

**Table 6-6. Summary: Failover Properties for the Oracle Driver**

<b>Property</b>	<b>Characteristic</b>
AlternateServers	One or multiple alternate database servers. An IP address or server name identifying each server is required. Port number and the ServiceName or SID connection properties are optional. If the port number is unspecified, the port specified for the primary server is used. If a port number is not specified for the primary server, the default port number of 1521 is used.
ConnectionRetryCount	Number of times the driver retries the primary database server, and if specified, alternate servers until a successful connection is established. The default is 5.
ConnectionRetryDelay	Wait interval, in seconds, between connection retry attempts when the ConnectionRetryCount property is set to a positive integer. The default is 1.
FailoverGranularity	Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. The default is nonAtomic (the driver continues with the failover process and posts any exceptions on the statement on which they occur).
FailoverMode	The failover method you want the driver to use. The default is connect (connection failover is used).
FailoverPreconnect	Specifies whether the driver tries to connect to the primary and an alternate server at the same time. The default is false (the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection).
LoadBalancing	Sets whether the driver will use client load balancing in its attempts to connect to the database servers (primary and alternate). If client load balancing is enabled, the driver uses a random pattern instead of a sequential pattern in its attempts to connect. The default is false (client load balancing is disabled).
PortNumber	Port listening for connections on the primary database server. This property is supported only for data source connections. The default is 1521.

---

**Table 6-6. Summary: Failover Properties for the Oracle Driver (cont.)**

---

Property	Characteristic
ServerName	IP address or server name of primary database server. This property is supported only for data source connections.
ServiceName	Database service name that specifies the database used for the connection. This property is mutually exclusive with the SID property.
SID	Oracle System Identifier that refers to the instance of the Oracle database running on the server. The default is ORCL. This property is mutually exclusive with the ServiceName property.

---

See “[Using Failover](#)” on page 58 and “[Using Client Load Balancing](#)” on page 65 for overviews of failover and client load balancing.

---

## Bulk Load

For Oracle 9*i* R2 and higher, the Oracle driver supports DataDirect Bulk Load, a feature that allows your application to send large numbers of rows of data to the database in a continuous stream instead of in numerous smaller database protocol packets. Similar to batch operations, performance improves because far fewer network round trips are required. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations. See “[Using DataDirect Bulk Load](#)” on page 829 for more information.



# 7 The Microsoft SQL Server Driver

The DataDirect Connect *for JDBC* SQL Server driver (the "SQL Server driver") supports the following versions:

- Microsoft SQL Server 2008
- Microsoft SQL Server 2005
- Microsoft SQL Server 2000
- Microsoft SQL Server 2000 Desktop Engine (MSDE 2000)
- Microsoft SQL Server 2000 Enterprise Edition (64-bit)
- Microsoft SQL Server 7.0

To use JDBC distributed transactions through JTA, you must install stored procedures for Microsoft SQL Server. See "["JTA Support: Installing Stored Procedures" on page 449](#)" for more information about JTA.

---

## Data Source and Driver Classes

The data source class for the SQL Server driver is:

`com.ddtek.jdbcx.sqlserver.SQLServerDataSource`

See "["Connecting Through Data Sources" on page 48](#)" for information about DataDirect Connect *for JDBC* data sources.

The driver class for the SQL Server driver is:

`com.ddtek.jdbc.sqlserver.SQLServerDriver`

## Connection URL

The connection URL format for the SQL Server driver is:

```
jdbc:datadirect:sqlserver://hostname:port[;property=value[;...]]
```

where:

*hostname* is the TCP/IP address or TCP/IP host name of the server to which you are connecting. See “[Using IP Addresses](#)” on page 50 for details on using IP addresses.

NOTE: Untrusted applets cannot open a socket to a machine other than the originating host.

*port* is the number of the TCP/IP port.

*property=value* specifies connection properties. For a list of connection properties and their valid values, see “[Connection Properties](#)” on page 375.

For example:

```
jdbc:datadirect:sqlserver://server1:1433;User=test;Password=secret
```

See “[Connecting to Named Instances](#)” on page 420 for instructions on connecting to named instances.

---

# J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the SQL Server resource adapter is:

`com.ddtek.resource.spi.SQLServerManagedConnectionFactory`

See “[J2EE Connector Architecture Resource Adapters](#)” on [page 37](#) for information about using DataDirect Connect *for JDBC* drivers as J2EE Connector Architecture resource adapters.

---

## Connection Properties

This section lists the JDBC connection properties supported by the SQL Server driver and describes each property. The properties have the form:

*property=value*

You can use these connection properties with either the JDBC Driver Manager or DataDirect Connect *for JDBC* data sources unless otherwise noted.

### NOTES:

- All connection property names are case-insensitive. For example, Password is the same as password. Required properties are noted as such.
- The data type listed for each connection property is the Java data type used for the property value in a JDBC data source.

## AccountingInfo

Description	Accounting information to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is the accounting information.
Default	empty string
Data Type	String
See Also	<a href="#">"Client Information for Connections" on page 446</a>

## AlternateServers

Description	A list of alternate database servers that is used to failover new or lost connections, depending on the failover method selected. See the <a href="#">FailoverMode</a> property for information about choosing a failover method.
Valid Values	( <i>servername1[:port1][;property=value[;...]]</i> [ , <i>servername2[:port2][;property=value[;...]]</i> ]...)
	The server name ( <i>servername1</i> , <i>servername2</i> , and so on) is required for each alternate server entry. Port number ( <i>port1</i> , <i>port2</i> , and so on) and connection properties ( <i>property=value</i> ) are optional for each alternate server entry. If the port is unspecified, the port number of the primary server is used. If a port number for the primary server is unspecified, a default port number of 1433 is used.
	The driver allows only one optional connection property, <a href="#">DatabaseName</a> .

NOTE: If using failover with Microsoft Cluster Server (MSCS), which determines the alternate server for failover instead of the

driver, any alternate server specified must be the same as the primary server. For example:

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=TEST;
User=test;Password=secret;AlternateServers=(server1:1433;
DatabaseName=TEST)
```

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

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=TEST;
User=test;Password=secret;AlternateServers=(server2:1433;
DatabaseName=TEST2,server2:1433;DatabaseName=TEST3)
```

**Default** None

**Data type** String

**See Also** ["Configuring Failover" on page 463](#)

## AlwaysReportTriggerResults

**Description** Determines how the driver reports results that are generated by database triggers (procedures that are stored in the database and executed, or fired, when a table is modified). For Microsoft SQL Server 2005 and higher, this includes triggers that are fired by Data Definition Language (DDL) events.

**Valid Values** If set to true, the driver returns all results, including results that are generated by triggers. Multiple trigger results are returned sequentially. Use the `Statement.getMoreResults()` method to return individual trigger results. Warnings and errors are reported in the results as they are encountered.

If set to false, the following behavior occurs:

- For Microsoft SQL Server 2005 and higher, the driver does not report trigger results if the statement is a single INSERT, UPDATE, DELETE, CREATE, ALTER, DROP, GRANT, REVOKE, or DENY statement.

- For other Microsoft SQL Server databases, the driver does not report trigger results if the statement is a single INSERT, UPDATE, or DELETE statement.

The only result that is returned is the update count that is generated by the statement that was executed (if errors do not occur). Although trigger results are ignored, any errors that are generated by the trigger are reported. Any warnings that are generated by the trigger are enqueued. If errors are reported, the update count is not reported.

Default false

Data Type boolean

## ApplicationName

Description The name of the application to be stored in the database. For Microsoft SQL Server 2000 and higher, this value sets the program\_name value in the sysprocesses table in the database. For Microsoft SQL Server 7, this value is stored locally. This value is used for database administration/monitoring purposes.

Valid Values *string*

where *string* is the name of the application.

NOTE: Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. See “[Returning MetaData About Client Information Locations](#)” on page 627.

Default empty string

Data Type String

Alias [ProgramName](#) property

See Also [“Client Information for Connections” on page 446](#)

## AuthenticationMethod

Description	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	<p>auto   kerberos   ntlm   userIdPassword</p> <p>If set to auto, the driver uses SQL Server authentication, Kerberos authentication, or NTLM authentication when establishing a connection. The driver selects an authentication method based on a combination of criteria, such as whether the application provides a user ID, the driver is running on a Windows platform, and the driver can load the DLL required for NTLM authentication. See “<a href="#">Using the AuthenticationMethod Property</a>” on page 429 for more information about using the default value.</p> <p>If set to kerberos, the driver uses Kerberos authentication. The driver ignores any user ID or password specified. This value is supported only when connecting to Microsoft SQL Server 2000 or higher.</p> <p>If set to ntlm, the driver uses NTLM authentication if the DLL required for NTLM authentication can be loaded. If the driver cannot load the DLL, the driver throws an exception. The driver ignores any user ID or password specified.</p> <p>If set to userIdPassword, the driver uses SQL Server authentication when establishing a connection. If a user ID is not specified, the driver throws an exception.</p>

**NOTES:**

- The [User](#) property provides the user ID. The [Password](#) property provides the password.
- The values type4, type2, and none are deprecated, but are recognized for backward compatibility. Use the kerberos, ntlm, and userIdPassword value, respectively, instead.

**Default** userIdPassword

**Data Type** String

**See Also** [“Authentication” on page 428](#)

**BulkLoadBatchSize**

**Description** 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.

**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 set by this property. See [“DDBulkLoad Interface” on page 603](#) for a description of the method.

**Valid Values** x

where x is a positive integer.

**Default** 2048

Data Type long

See Also ["Using DataDirect Bulk Load" on page 829](#)

## BulkLoadOptions

Description Enables options of the bulk load protocol of which the driver can take advantage.

Valid Values This value is the cumulative value of all enabled options. The following list describes the value and the corresponding option that is enabled:

Value	Option Enabled
1	The KeepIdentity option preserves identity values. If unspecified, identity values are ignored in the source and are assigned by the destination.  NOTE: If using the bulk load feature with batch inserts, this option has no effect if enabled.
2	The TableLock option assigns a table lock for the duration of the bulk copy operation. Other applications cannot update the table until the operation completes. If unspecified, the default bulk locking mechanism specified by the database server is used.
16	The CheckConstraints option checks integrity constraints while data is being copied. If unspecified, constraints are not checked.
32	The FireTriggers option causes the database server to fire insert triggers for the rows being inserted into the database. If unspecified, triggers are not fired.
64	The KeepNulls option preserves null values in the destination table regardless of the settings for default values. If unspecified, null values are replaced by column default values where applicable.

If set to 0, all the options are disabled.

Example	A value of 67 means the KeepIdentity, TableLock, and KeepNulls option are enabled (1 +2 + 64).
Default	0
Data Type	long
See Also	<a href="#">"Using DataDirect Bulk Load" on page 829</a>

## ClientHostName

Description	The host name, or workstation ID, of the client machine to be stored in the database. For Microsoft SQL Server 2000 and higher, this value sets the hostname value of the sysprocesses table in the database. For Microsoft SQL Server 7, this value is stored locally. This value is used for database administration/monitoring purposes.
Valid Values	<i>string</i>
	where <i>string</i> is the host name of the client machine.
	NOTE: Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. See <a href="#">"Returning MetaData About Client Information Locations" on page 627</a> .
Default	empty string
Data Type	String
Alias	WSID property
See Also	<a href="#">"Client Information for Connections" on page 446</a>

## ClientUser

Description	The user ID to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is a valid user ID.
Default	empty string
Data Type	String
See Also	<a href="#">"Client Information for Connections" on page 446</a>

## CodePageOverride

Description	<p>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.</p> <p>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.</p> <p>If a value is set for this property and the <a href="#">SendStringParametersAsUnicode</a> property is set to true, the driver ignores the SendStringParametersAsUnicode property and generates a warning. The driver always sends parameters using the code page that is specified by CodePageOverride, if specified.</p>
Valid Values	<i>string</i> where <i>string</i> is the name of a valid code page that is supported by your JVM.
Example	CP950

Default None  
Data Type String

## ConnectionRetryCount

Description	The number of times the driver retries connection attempts to the primary database server, and if specified, alternate servers until a successful connection is established.
	<p>NOTE: If an application sets a login timeout value (for example, using <code>DataSource.loginTimeout</code> or <code>DriverManager.loginTimeout</code>), and the login timeout expires, the driver ceases connection attempts.</p>
Valid Values	<p>0   <math>x</math></p> <p>where <math>x</math> is a positive integer.</p> <p>If set to 0, the driver does not try to reconnect after the initial unsuccessful attempt.</p> <p>If set to <math>x</math>, 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.</p> <p>NOTE: The <a href="#">ConnectionRetryDelay</a> property specifies the wait interval, in seconds, to occur between retry attempts.</p>
Example	If this property is set to 2 and alternate servers are specified using the <a href="#">AlternateServers</a> property, the driver retries the list of servers (primary and alternate) twice after the initial retry attempt.
Default	5 (seconds)
Data Type	int
See Also	<a href="#">“Configuring Failover” on page 463</a>

## ConnectionRetryDelay

Description	The number of seconds the driver waits between connection retry attempts when <a href="#">ConnectionRetryCount</a> is set to a positive integer.
Valid Values	0   $x$ where $x$ is a number of seconds.  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 <a href="#">ConnectionRetryCount</a> is set to 2, this property is set to 3, and alternate servers are specified using the <a href="#">AlternateServers</a> 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.
Default	1 (second)
Data Type	int
See Also	<a href="#">"Configuring Failover" on page 463</a>

## ConvertNull

Description	Controls how data conversions are handled for null values.
Valid Values	0   1  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 this 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 the data type of the column value.

Default 1

Data Type int

## Database

Description An alias for the [DatabaseName](#) property.

## DatabaseName

Description The name of the database to which you want to connect.

Valid Values *string*

where *string* is the name of a Microsoft SQL Server database.

Default None

Data Type String

Alias [Database](#) property. If both the Database and DatabaseName properties are specified in a connection URL, the last property that is positioned in the connection URL is used. For example, if your application specifies the following connection URL, the value of the Database connection property would be used instead of the value of the DatabaseName connection property.

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=jdbc;  
Database=acct;User=test;Password=secret
```

## DescribeParameters

Description Controls whether the driver attempts to determine, at execute time, how to send String parameters to the server based on the database data type. Sending String parameters as the type the

	database expects improves performance and prevents unexpected locking issues caused by data type mismatches.
Valid Values	noDescribe   describefString
	If set to noDescribe, the driver does not attempt to describe SQL parameters to determine the database data type. The driver sends String parameter values to the server based on the setting of the <a href="#">SendStringParametersAsUnicode</a> property.
	If set to describefString, the driver attempts to describe SQL parameters to determine the database data type if one or multiple parameters has been bound as a String (using the PreparedStatement methods <code>setString()</code> , <code>setCharacterStream()</code> , and <code>setAsciiStream()</code> ). If the driver can determine the database data type, the driver sends the String parameter data to the server as Unicode if the database type is an n-type (for example, <code>nvarchar</code> ). If the database type is not an n-type, the driver converts the data to the character encoding defined by the parameter's collation and sends the data to the server in that character encoding. If the driver cannot determine the data type of the parameters, it sends String parameter values to the server based on the setting of the <a href="#">SendStringParametersAsUnicode</a> property.
	NOTE: The <a href="#">SendStringParametersAsUnicode</a> property controls whether the driver sends String parameter values to the server as Unicode (for example, <code>nvarchar</code> ) or non-Unicode (for example, <code>varchar</code> ). This property helps applications in which character columns are all Unicode or all non-Unicode. For applications that access both Unicode and non-Unicode columns, a data type mismatch still occurs for some columns if the driver always sends String parameter values to the server in only one format.
Default	noDescribe
Data Type	String

## EnableBulkLoad

Description	Specifies whether the driver uses the native bulk load protocols in the database instead of the batch mechanism for batch inserts. Bulk load bypasses the data parsing that is usually done by the database, providing an additional performance gain over batch operations. This property allows existing applications with batch inserts to take advantage of bulk load without requiring changes to the application code.
Valid Values	true   false
	If set to true, the driver uses the native bulk load protocols for batch inserts.
	If set to false, the driver uses the batch mechanism for batch inserts.
Default	false
Data Type	boolean
See Also	<a href="#">"Using Bulk Load for Batch Inserts" on page 837</a> <a href="#">"Performance Considerations" on page 417</a>

## EnableCancelTimeout

Description	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 <code>Statement.setQueryTimeout(5)</code> 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.

Default false

Data Type boolean

## EncryptionMethod

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

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.

Valid Values noEncryption | SSL | requestSSL | loginSSL

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.

If set to loginSSL, the login request is encrypted using SSL. Data is encrypted using SSL If the database server is configured to require SSL. If the database server does not require SSL, data is not encrypted and only the login request is encrypted.

If SSL is enabled, the following properties also apply:

[HostNameInCertificate](#)  
[TrustStore](#)  
[TrustStorePassword](#)  
[ValidateServerCertificate](#)

NOTE: If SSL is enabled, the driver communicates with database protocol packets that are set by the server's default packet size. Any value set by the [PacketSize](#) property is ignored.

Default	noEncryption
Data Type	String
See Also	<a href="#">"Data Encryption" on page 439</a> <a href="#">"Performance Considerations" on page 417</a>

## FailoverGranularity

Description	Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. This property is ignored if <a href="#">FailoverMode=connect</a> .
Valid Values	<p>nonAtomic   atomic   atomicWithRepositioning   disableIntegrityCheck</p> <p>If set to nonAtomic, the driver continues with the failover process and posts any exceptions on the statement on which they occur.</p> <p>If set to atomic, the driver fails the entire failover process if an exception is generated as the result of restoring the state of the connection. If an exception is generated as a result of restoring the state of work in progress, the driver continues with the failover process, but generates an exception warning that the Select statement must be reissued.</p>

If set to atomicWithRepositioning, the driver fails the entire failover process if any exception is generated as the result of restoring the state of the connection or the state of work in progress.

If set to disableIntegrityCheck, the driver does not verify that the rows that are restored during the failover process match the original rows. This value is applicable only when FailoverMode=select.

Default nonAtomic

Data Type String

See Also ["Configuring Failover" on page 463](#)

## FailoverMode

Description Specifies the type of failover method the driver uses.

Valid Values connect | extended | select

If set to connect, the driver provides failover protection for new connections only.

If set to extended, the driver provides failover protection for new and lost connections, but not any work in progress.

If set to select, the driver provides failover protection for new and lost connections. In addition, it preserves the state of work performed by the last Select statement executed on the Statement object.

### NOTES:

- The [AlternateServers](#) property specifies one or multiple alternate servers for failover and is required for all failover methods.

- The [FailoverGranularity](#) property determines which action the driver takes if exceptions occur during the failover process.
- The [FailoverPreconnect](#) property specifies whether the driver tries to connect to multiple database servers (primary and alternate) at the same time.

Default connect

Data Type String

See Also [“Configuring Failover” on page 463](#)

## FailoverPreconnect

Description Specifies whether the driver tries to connect to the primary and an alternate server at the same time. This property is ignored if [FailoverMode=connect](#).

Valid Values true | false

If set to true, the driver tries to connect to the primary and an alternate server at the same time. This can be useful if your application is time-sensitive and cannot absorb the wait for the failover connection to succeed.

If set to false, the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection. This value provides the best performance, but your application typically experiences a short wait while the failover connection is attempted.

NOTE: The [AlternateServers](#) property specifies one or multiple alternate servers for failover.

Default false

Data Type boolean

See Also [“Configuring Failover” on page 463](#)

## HostNameInCertificate

Description	<p>Specifies a host name for certificate validation when SSL encryption is enabled (using the <a href="#">EncryptionMethod</a> property) and validation is enabled (<a href="#">ValidateServerCertificate=true</a>). This property is optional and provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server that the driver is connecting to is the server that was requested.</p> <p>NOTES:</p> <ul style="list-style-type: none"><li>■ If SSL encryption or certificate validation is not enabled, this property is ignored.</li><li>■ If SSL encryption and validation is enabled and this property is unspecified, the driver uses the server name that is specified in the connection URL or data source of the connection to validate the certificate.</li></ul>
Valid Values	<p><i>host_name</i>   #SERVERNAME#</p> <p>where <i>host_name</i> is a valid host name.</p> <p>If <i>host_name</i> is specified, the driver compares the specified host name to the DNSName value of the SubjectAlternativeName in the certificate. If a DNSName value does not exist in the SubjectAlternativeName or if the certificate does not have a SubjectAlternativeName, the driver compares the host name with the Common Name (CN) part of the certificate's Subject name. If the values do not match, the connection fails and the driver throws an exception.</p> <p>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 a DNSName value does not exist in the SubjectAlternativeName or 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</p>

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.

Default empty string

Data Type String

## HostProcess

Description An alias for the [ProgramID](#) property.

## ImportStatementPool

Description 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.

Default empty string

Data Type String

See Also [“Importing Statements into a Statement Pool” on page 820](#)

[“Performance Considerations” on page 417](#)

## InitializationString

Description	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	<p><i>string</i></p> <p>where <i>string</i> is one or multiple SQL commands.</p> <p>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.</p>
Example	The following connection URL sets the handling of null values to the Microsoft SQL Server default and allows delimited identifiers:
	<pre>jdbc:datadirect:sqlserver://server1:1433; InitializationString=(set ANSI_NULLS off; set QUOTED_IDENTIFIER on);DatabaseName=test</pre>
Default	None
Data Type	String

## InInsensitiveResultSetBufferSize

Description	Determines the amount of memory used by the driver to cache insensitive result set data.
Valid Values	<p>-1   0   <i>x</i></p> <p>where <i>x</i> is a positive integer.</p> <p>If set to -1, the driver caches insensitive result set data in memory. If the size of the result set exceeds available memory,</p>

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.

Default 2048

Data Type int

See Also ["Performance Considerations" on page 417](#)

## JavaDoubleToString

Description 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

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.

Default false

Data Type boolean

## JDBCBehavior

Description Determines how the driver describes database data types that map to the following JDBC 4.0 data types: NCHAR, NVARCHAR, NLONGVARCHAR, NCLOB, and SQLXML.

This property is applicable only when the application is using Java SE 6.

Valid Values 0 | 1

If set to 0, the driver describes the data types as JDBC 4.0 data types when using Java SE 6.

If set to 1, the driver describes the data types using JDBC 3.0-equivalent data types, regardless of JVM. This allows your application to continue using JDBC 3.0 types in a Java SE 6 environment. Additionally, the PROCEDURE\_NAME column contains procedure name qualifiers. For example, for the fully qualified procedure named 1.sp\_productadd, the driver would return sp\_productadd;1.

Default 1

Data Type int

## LoadBalancing

Description	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 <a href="#">AlternateServers</a> property.
Valid Values	true   false
	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).
Default	false
Data Type	boolean
See Also	<a href="#">"Configuring Failover" on page 463</a>

## LoadLibraryPath

Description	Specifies the directory for the DLL for NTLM authentication. The driver looks for the DLL in the specified directory.  NOTE: When you install the driver, the NTLM authentication DLLs are installed in the <code>install_dir/lib</code> subdirectory, where <code>install_dir</code> is the product installation directory.
-------------	--

**Valid Values** *string*

where *string* is the fully qualified path of the directory that contains the DLL for NTLM authentication.

If unspecified, the driver looks for the DLL in a directory on the Windows system path defined by the PATH environment variable.

If set to *string*, the driver looks in the specified directory for the DLL. Use this value if you install the driver in a directory that is not on the Windows system path.

**Example** If you install the driver in a directory named "DataDirect" that is not on the Windows system path, use this property to specify the directory containing the NTLM authentication DLL.

```
jdbc:datadirect:sqlserver://server3:1433;DatabaseName=test;  
LoadLibraryPath=C:\DataDirect\lib;User=test;Password=secret
```

**Default** None

**Data Type** String

**See Also** ["Configuring NTLM Authentication" on page 437](#)

## LoginTimeout

**Description** 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 number of seconds.

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.

Default 0

Data Type int

## LongDataCacheSize

Description Determines whether the driver caches long data (images, pictures, long text, or binary data) in result sets. To improve performance, you can disable long data caching if your application retrieves columns in the order in which they are defined in the result set.

Valid Values -1 | 0 |  $x$

where  $x$  is a positive integer.

If set to -1, the driver does not cache long data in result sets. It is cached on the server. Use this value only if your application retrieves columns in the order in which they are defined in the result set.

If set to 0, the driver caches long data in result sets in memory. If the size of the result set data exceeds available memory, the driver pages the result set data to disk.

If set to  $x$ , the driver caches long data in result sets in memory and uses this value to set the size (in KB) of the memory buffer for caching result set data. If the size of the result set data exceeds available memory, the driver pages the result set data to disk.

Default 2048

Data Type int

See Also [“Performance Considerations” on page 417](#)

## MaxPooledStatements

Description	The maximum number of pooled prepared statements for this connection. Setting MaxPooledStatements to an integer greater than zero (0) enables the driver's internal prepared statement pooling, which is useful when the driver is not running from within an application server or another application that provides its own prepared statement pooling.
Valid Values	0   $x$ where $x$ is a positive integer.  If set to 0, the driver's internal prepared statement pooling is not enabled.  If set to $x$ , the driver enables the DataDirect Statement Pool Monitor and uses the specified value to cache a certain number of prepared statements that are created by an application. For 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. If the value set for this property is greater than the number of prepared statements that are used by the application, all prepared statements created by the application are cached. Because CallableStatement is a sub-class of PreparedStatement, CallableStatements also are cached.
Default	0
Data Type	int
Alias	<a href="#">MaxStatements</a> property
See Also	<a href="#">“Performance Considerations” on page 417</a> <a href="#">Appendix J “Statement Pool Monitor” on page 811</a>

## MaxStatements

Description	An alias for the <a href="#">MaxPooledStatements</a> property.
-------------	--

## NetAddress

Description	The Media Access Control (MAC) address of the network interface card of the application connecting to Microsoft SQL Server. The value of this property may be useful for database administration purposes. This value is stored in the <code>net_address</code> column of the:
	<ul style="list-style-type: none"><li>■ <code>sys.sysprocesses</code> table (Microsoft SQL Server 2005 and higher)</li><li>■ <code>master.dbo.sysprocesses</code> table (Microsoft SQL Server 2000)</li></ul>
	Microsoft SQL Server 7 does not store this value.
Valid Values	<code>string</code> where <code>string</code> is a maximum of 12 characters.
Default	000000000000
Data Type	String

## PacketSize

Description	Determines the number of bytes for each database protocol packet that is transferred from the database server to the client machine (Microsoft SQL Server refers to this packet as a network packet).
	Adjusting the packet size can improve performance. The optimal value depends on the typical size of data that is inserted, updated, or returned by the application and the environment in which it is running. Typically, larger packet sizes work better for large amounts of data. For example, if an application regularly returns character values that are 10,000 characters in length, using a value of 32 (16 KB) typically results in improved performance.

NOTE: If SSL encryption is enabled using the [EncryptionMethod](#) property, any value set for the PacketSize property is ignored.

Valid Values -1 | 0 |  $x$

where  $x$  is an integer from 1 to 128.

If set to -1, the driver uses the maximum packet size that the database server accepts.

If set to 0, the driver uses the default packet size of the database server.

If set to  $x$ , the driver uses a packet size that is a multiple of 512 bytes. For example, PacketSize=8 means to set the packet size to  $8 * 512$  bytes (4096 bytes).

NOTE: If your application sends queries that only retrieve small result sets, you may want to use a packet size that is smaller than the maximum packet size that is configured on the database server. If a result set that contains only one or two rows of data does not completely fill a larger packet, performance will not improve by setting the value to the maximum packet size.

Default -1

Data Type int

See Also [“Performance Considerations” on page 417](#)

## Password

Description A password that is used to connect to your Microsoft SQL Server database. A password is required if SQL Server authentication is enabled on your database. Contact your system administrator to obtain your password.

Valid Values *string*

where *string* is a valid password. The password is case-insensitive.

Default None  
Data Type String

## PortNumber

Description The TCP port of the primary database server that is listening for connections to the Microsoft SQL Server database.

This property is supported only for data source connections.

Valid Values *port*

where *port* is the port number.

Default 1433

Data Type int

## ProgramID

Description The product and version information of the driver on the client to be stored in the database. For Microsoft SQL Server 2000 and higher, this value sets the hostprocess value in the sysprocesses table. For Microsoft SQL Server 7, this value is stored locally. This value is used for database administration/monitoring purposes.

Valid Values DDJVVRM

where:

- *DD* is an identifier for a DataDirect Connect for JDBC driver.
- *VV* identifies a 2-digit version number (with high-order 0 in the case of a 1-digit version).
- *RR* identifies a 2-digit release number (with high-order 0 in the case of a 1-digit release).
- *M* identifies a 1-character modification level (0-9 or A-Z).

Example	DDJ04100
Default	empty string
Data Type	String
Alias	<a href="#">HostProcess</a> property
See Also	<a href="#">“Client Information for Connections” on page 446</a>

## ProgramName

Description	An alias for the <a href="#">ApplicationName</a> property.
-------------	--

## QueryTimeout

Description	Sets the default query timeout (in seconds) for all statements that are created by a connection.
Valid Values	-1   0   $x$ where $x$ is a number of seconds.  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.
Default	0
Data Type	int

## ReceiveStringParameterType

Description	Specifies how the driver describes String stored procedure output parameters to the database.
Valid Values	NVARCHAR   VARCHAR   DESCRIBE
	If set to NVARCHAR, the driver describes String stored procedure output parameters as nvarchar (4000). Use this value if all output parameters that are returned by the connection are nchar or nvarchar. If the output parameter is char or varchar, the driver returns the output parameter value, but the returned value is limited to 4000 characters.
	If set to VARCHAR, the driver describes String stored procedure output parameters as varchar (8000). Use this value if all output parameters that are returned by the connection are char or varchar. If the output parameter is nchar or nvarchar, data may not be returned correctly. This can occur when the returned data uses a code page other than the database default code page.
	If set to DESCRIBE, the driver submits a request to the database to describe the parameters of the stored procedure. The driver uses the parameter data types that are returned by the driver to determine whether to describe the String output parameters as nvarchar or varchar. Use this value if there is a combination of nvarchar and varchar output parameters and if the varchar output parameters can return values that are greater than 4000 characters. This method always works, but it incurs the expense of having to describe the output parameters.
Default	NVARCHAR
Data Type	String

## ResultSetMetaDataOptions

Description	Determines whether the driver returns table name information in the ResultSet metadata for Select statements.
Valid Values	0   1
	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.
Default	0
Data Type	int
See Also	<a href="#">“Performance Considerations” on page 417</a>

## SelectMethod

Description	A hint to the driver that determines whether the driver requests a database cursor for Select statements. Performance and behavior of the driver are affected by this property, which is defined as a hint because the driver may not always be able to satisfy the requested method.
Valid Values	direct   cursor
	If set to direct, the database server sends the complete result set in a single response to the driver when responding to a query. A server-side database cursor is not created if the requested result

set type is a forward-only result set. Typically, responses are not cached by the driver. Using this method, the driver must process the entire response to a query before another query is submitted. If another query is submitted (using a different statement on the same connection, for example), the driver caches the response to the first query before submitting the second query. Typically, the direct method performs better than the cursor method.

If set to cursor, a server-side cursor is requested. When returning forward-only result sets, the rows are returned from the server in blocks. The `setFetchSize()` method can be used to control the number of rows that are returned for each request when forward-only result sets are returned. Performance tests show that, when returning forward-only result sets, the value of `Statement.setFetchSize()` significantly impacts performance. There is no simple rule for determining the `setFetchSize()` value that you should use. We recommend that you experiment with different `setFetchSize()` values to determine which value gives the best performance for your application. The cursor method is useful for queries that produce a large amount of data, particularly if multiple open result sets are used.

Default direct

Data Type String

See Also ["Performance Considerations" on page 417](#)

## SendStringParametersAsUnicode

Description Determines whether string parameters are sent to the Microsoft SQL Server database in Unicode or in the default character encoding of the database.

Valid Values true | false

If set to true, string parameters are sent to Microsoft SQL Server in Unicode.

NOTE: If a value is specified for the [CodePageOverride](#) property and this property is set to true, this property is ignored and a warning is generated.

If set to false, the driver sends string parameters to the database in the default character encoding of the database, which can improve performance because the server does not need to convert Unicode characters to the default encoding.

Default true

Data Type boolean

See Also ["Performance Considerations" on page 417](#)

## ServerName

Description 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.

To connect to a named instance, specify *server\_name*\*instance\_name* for this property, where *server\_name* is the IP address and *instance\_name* is the name of the instance to which you want to connect on the specified server.

Example 122.23.15.12 or SQLServerServer

Default None

Data Type String

See Also ["Connecting to Named Instances" on page 420](#)

## SnapshotSerializable

Description	For Microsoft SQL Server 2005 and higher only. Allows your application to use Snapshot Isolation for connections.
	This property is useful for applications that have the Serializable isolation level set. Using the SnapshotSerializable property allows you to use Snapshot Isolation with no or minimum code changes. If you are developing a new application, you may find that using the constant TRANSACTION_SNAPSHOT is a better choice.
Valid Values	true   false
	If set to true and your application has the transaction isolation level set to Serializable, the application uses Snapshot Isolation for connections.
	NOTE: To use Snapshot Isolation, your database also must be configured for Snapshot Isolation.
	If set to false and your application has the transaction isolation level set to Serializable, the application uses the Serializable isolation level.
Default	false
Data Type	boolean
See Also	<a href="#">"Isolation Levels" on page 447</a> <a href="#">"Performance Considerations" on page 417</a>

## SpyAttributes

Description	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	( <i>spy_attribute</i> [; <i>spy_attribute</i> ]...)
	where <i>spy_attribute</i> is any valid DataDirect Spy attribute. See <a href="#">Appendix H "Tracking JDBC Calls with DataDirect Spy™" on page 783</a> for a list of supported attributes.
	<b>NOTE:</b> If coding a path on Windows to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: log=(file)C:\\temp\\\\spy.log.
Example	The following value instructs the driver to log all JDBC activity to a file using a maximum of 80 characters for each line.  (log=(file)/tmp/spy.log;linelimit=80)
Default	None
Data Type	String

## TransactionMode

Description	Controls how the driver delimits the start of a local transaction.
Valid Values	implicit   explicit
	If set to implicit, the driver uses implicit transaction mode. This means that Microsoft SQL Server, not the driver, automatically starts a transaction when a transactionable statement is executed. Typically, implicit transaction mode is more efficient than explicit transaction mode because the driver does not have to send commands to start a transaction and a transaction is not started until it is needed. When TRUNCATE TABLE statements are used with implicit transaction mode, Microsoft SQL Server may roll back the transaction if an error occurs. If this occurs, use the explicit value for this property.
	If set to explicit, the driver uses explicit transaction mode. This means that the driver, not Microsoft SQL Server, starts a new

transaction if the previous transaction was committed or rolled back.

Default implicit

Data Type String

## TruncateFractionalSeconds

Description Determines whether the driver truncates timestamp values to three fractional seconds. For example, a value of the datetime2 data type can have a maximum of seven fractional seconds.

Valid Values true | false

If set to true, the driver truncates all timestamp values to three fractional seconds.

If set to false, the driver does not truncate fractional seconds.

Default true

Data Type boolean

## TrustStore

Description Specifies the directory of the truststore file to be used when SSL is enabled (using the [EncryptionMethod](#) property) and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.

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

This property is ignored if [ValidateServerCertificate](#)=false.

Valid Values *string*

where *string* is the directory of the truststore file.

Default None

Data Type String

## TrustStorePassword

Description Specifies the password that is used to access the truststore file when SSL is enabled (using the [EncryptionMethod](#) property) 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.

Default None

Data Type String

## User

Description The user name that is used to connect to the Microsoft SQL Server database. A user name is required only if SQL Server authentication is enabled on your database. Contact your system administrator to obtain your user name.

Valid Values *string*

where *string* is a valid user name. The user name is case-insensitive.

Default None

Data Type String

## UseServerSideUpdatableCursors

Description Determines whether the driver uses server-side cursors when an updatable result set is requested.

Valid Values true | false

If set to true, server-side updatable cursors are created when an updatable result set is requested.

If set to false, the client-side updatable cursors are created when an updatable result set is requested.

Default false

Data Type boolean

See Also ["Server-Side Updatable Cursors" on page 448](#)

["Performance Considerations" on page 417](#)

## ValidateServerCertificate

Description 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
	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 <a href="#">HostNameInCertificate</a> property is specified, the driver also validates the certificate using a host name. The <a href="#">HostNameInCertificate</a> 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 <a href="#">TrustStore</a> and <a href="#">TrustStorePassword</a> properties or Java system properties.
	NOTE: Truststore information is specified using the <a href="#">TrustStore</a> and <a href="#">TrustStorePassword</a> properties or by using Java system properties.
Default	true
Data Type	boolean

## WSID

Description	An alias for the <a href="#">ClientHostName</a> property.
-------------	---

## XATransactionGroup

Description	The transaction group ID that identifies any distributed transactions that are initiated by the connection. This ID can be used for distributed transaction cleanup purposes.
-------------	---

You can use the XAResource.recover method to roll back any transactions left in an unprepared state. When you call XAResource.recover, any unprepared transactions that match the ID on the connection used to call XAResource.recover are rolled back.

**Valid Values** *string*

where *string* is a valid transaction group ID.

**Example** If you specify XATransactionGroup=ACCT200 and call XAResource.recover on the same connection, any transactions that are left in an unprepared state identified by the transaction group ID of ACCT200 are rolled back.

**Default** None

**Data Type** String

**See Also** [“Distributed Transaction Cleanup” on page 452](#)

## XMLDescribeType

**Description** Determines whether the driver maps XML data to the LONGVARCHAR or LONGVARBINARY data type.

**Valid Values** longvarchar | longvarbinary

If set to longvarchar (the default), the driver maps XML data to the LONGVARCHAR data type.

If set to longvarbinary, the driver maps XML data to the LONGVARBINARY data type.

**Default** None

**Data Type** String

**See Also** [“Returning and Inserting/Updating XML Data” on page 424](#)

---

# Performance Considerations

You can optimize your application's performance if you set the SQL Server driver connection properties as described in this section:

**EnableBulkLoad:** For batch inserts, the driver can use native bulk load protocols instead of the batch mechanism. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations. Set this property to true to allow existing applications with batch inserts to take advantage of bulk load without requiring changes to the code.

**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.

**LongDataCacheSize:** To improve performance when your application retrieves images, pictures, long text, or binary data, you can disable caching for long data on the client if your application retrieves long data column values in the order they are defined in the result set. If your application retrieves long data column values out of order, long data values must be cached.

**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.

See ["Managing Connections" on page 711](#) for more information about using prepared statement pooling to optimize performance.

**PacketSize:** Typically, it is optimal for the client to use the maximum packet size that the server allows. This reduces the total number of round trips required to return data to the client, thus improving performance. Therefore, performance can be improved if this property is set to the maximum packet size of the database server.

**SelectMethod:** In most cases, using server-side database cursors impacts performance negatively. However, if the following four variables are true in your application, the best setting for this property is cursor, which means use server-side database cursors:

- Your application contains queries that retrieve large amounts of data.
- Your application executes a SQL statement before processing or closing a previous large result set and does this multiple times.
- Large result sets use forward-only cursors.

**SendStringParametersAsUnicode:** If your application is inserting or updating the database with string data that is in the same character encoding as the database, performance is improved because the server does not need to convert characters. In the case, this property should be set to true.

**SnapshotSerializable:** You must have your Microsoft SQL Server 2005 or higher database configured for Snapshot Isolation for this connection property to work. See ["Using the Snapshot Isolation Level \(Microsoft SQL Server 2005 and Higher\)" on page 447](#) for details.

Snapshot Isolation provides transaction-level read consistency and an optimistic approach to data modifications by not acquiring locks on data until data is to be modified. This Microsoft SQL Server 2005 and higher feature can be useful if you want to consistently return the same result set even if another transaction has changed the data and 1) your application executes many read operations or 2) your application has long running transactions that could potentially block users from reading data. This feature has the potential to eliminate data contention between read operations and update operations. When this connection property is set to true (thereby, you are using Snapshot Isolation), performance is improved due to increased concurrency.

**UseServerSideUpdatableCursors:** In most cases, using server-side updatable cursors improves performance. However, this type of cursor cannot be used with insensitive result sets or with sensitive results sets that are not generated from a database table that contains a primary key.

See ["Server-Side Updatable Cursors" on page 448](#) for more information about using server-side updatable cursors.

---

## Connecting to Named Instances

Microsoft SQL Server 2000 and higher support multiple instances of a Microsoft SQL Server database running concurrently on the same server. An instance is identified by an instance name. To connect to a named instance using a connection URL, use the following URL format:

```
jdbc:datadirect:sqlserver://server_name\\instance_name
```

NOTE: The first backslash character (\) in \\*instance\_name* is an escape character.

where:

*server\_name* is the IP address or hostname of the server.

*instance\_name* is the name of the instance to which you want to connect on the server.

For example, the following connection URL connects to an instance named *instance1* on *server1*:

```
jdbc:datadirect:sqlserver://server1\\instance1;User=test;Password=secret
```

To connect to a named instance using a data source, you specify the *ServerName* property. For example:

```
SQLServerDataSource mds = new SQLServerDataSource();
mds.setDescription("My SQLServerDataSource");
mds.setServerName("server1\\instance1");
mds.setDatabaseName("TEST");
mds.setUser("test");
mds.setPassword("secret");
```

---

# Data Types

[Table 7-1](#) lists the data types supported by the SQL Server driver and how they are mapped to the JDBC data types.

---

**Table 7-1. Microsoft SQL Server Data Types**

---

Microsoft SQL Server Data Type	JDBC Data Type
bigint <sup>1</sup>	BIGINT
bigint identity <sup>1</sup>	BIGINT
binary	BINARY
bit	BIT
char	CHAR
date	DATE
datetime	TIMESTAMP
datetime2	TIMESTAMP
datetimeoffset	VARCHAR
decimal	DECIMAL
decimal() identity	DECIMAL
float	FLOAT
image	LONGVARBINARY
int	INTEGER
int identity	INTEGER
money	DECIMAL
nchar	CHAR
	NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NCHAR (if using Java SE 6) or CHAR (if using another JVM).

**Table 7-1. Microsoft SQL Server Data Types (cont.)**

<b>Microsoft SQL Server Data Type</b>	<b>JDBC Data Type</b>
ntext	LONGVARCHAR  NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: LONGNVARCHAR (if using Java SE 6) or LONGVARCHAR (if using another JVM).
numeric	NUMERIC
numeric() identity	NUMERIC
nvarchar	VARCHAR  NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NVARCHAR (if using Java SE 6) or VARCHAR (if using another JVM).
nvarchar(max) <sup>2</sup>	LONGVARCHAR  NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: LONGNVARCHAR (if using Java SE 6) or LONGVARCHAR (if using another JVM).
real	REAL
smalldatetime	TIMESTAMP
smallint	SMALLINT
smallint identity	SMALLINT
smallmoney	DECIMAL
sql_variant <sup>1</sup>	VARCHAR
sysname	VARCHAR
text	LONGVARCHAR
time	TIMESTAMP
timestamp	BINARY

---

**Table 7-1. Microsoft SQL Server Data Types (cont.)**

---

Microsoft SQL Server Data Type	JDBC Data Type
tinyint	TINYINT
tinyint identity	TINYINT
uniqueidentifier	CHAR
varbinary	VARBINARY
varbinary(max) <sup>2</sup>	LONGVARBINARY
varchar	VARCHAR
varchar(max) <sup>2</sup>	LONGVARCHAR
xml <sup>2,3</sup>	LONGVARCHAR
NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: SQLXML (if using Java SE 6) or LONGVARCHAR (if using another JVM).	
<hr/> <ol style="list-style-type: none"><li>1. Supported only for Microsoft SQL Server 2000 and higher.</li><li>2. Supported only for Microsoft SQL Server 2005 and higher.</li><li>3. The XMLDescribeType property overrides the mappings for XML data.</li></ol> <hr/>	

See [Appendix D “getTypeInfo” on page 631](#) for more information about data types.

---

## Returning and Inserting/Updating XML Data

For Microsoft SQL Server 2005 and higher, the SQL Server driver supports the xml data type. Which JDBC data type the xml data type is mapped to depends on whether the JDBCBehavior and XMLDescribeType properties are set:

- If XMLDescribeType=longvarchar or XMLDescribeType=longvarbinary, the driver maps the XML data type to the JDBC LONGVARCHAR or LONGVARBINARY data type, respectively, regardless of the setting of the JDBCBehavior property.
- If JDBCBehavior=1 (the default) and the XMLDescribeType property is not set, the driver maps XML data to the JDBC LONGVARCHAR data type.
- If JDBCBehavior=0 and the XMLDescribeType property is not set, XML data is mapped to SQLXML or LONGVARCHAR, depending on which JVM your application is using. The driver maps the XML data type to the JDBC SQLXML data type if your application is using Java SE 6. If your application is using another JVM, the driver maps the XML data type to the JDBC LONGVARCHAR data type.

## Returning XML Data

You can specify whether XML data is returned as character or binary data by setting the XMLDescribeType property. For example, consider a database table defined as:

```
CREATE TABLE xmlTable (id int, xmlCol xml NOT NULL)
```

and the following code:

```
String sql="SELECT xmlCol FROM xmlTable";
ResultSet rs=stmt.executeQuery(sql);
```

If your application uses the following connection URL, which specifies that the XML data type be mapped to the LONGVARBINARY data type, the driver would return XML data as binary data:

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=jdbc;User=test;  
Password=secret;XMLDescribeType=longvarbinary
```

## **Character Data**

When XMLDescribeType=longvarchar, the driver returns XML data as character data. The result set column is described with a column type of LONGVARCHAR and the column type name is xml.

When XMLDescribeType=longvarchar, 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:** This 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.

If `XMLDescribeType=longvarbinary`, your application should not use any of the methods for returning character data described in this section. In this case, the driver applies the standard JDBC character-to-binary conversion to the data, which returns the hexadecimal representation of the character data.

## ***Binary Data***

When `XMLDescribeType=longvarbinary`, the driver returns XML data as binary data. The result set column is described with a column type of `LONGVARBINARY` and the column type name is `xml`.

Your application can use the following methods to return XML data as binary data:

```
ResultSet.getBytes()  
ResultSet.getBinaryStream()  
ResultSet.getBlob()  
ResultSet.getObject()  
CallableStatement.getBytes()  
CallableStatement.getBlob()  
CallableStatement.getObject()
```

The driver does not apply any data conversions to the XML data returned from the database server. These methods return a byte array or binary stream that contains the XML data encoded as UTF-8.

If `XMLDescribeType=longvarchar`, your application should not use any of the methods for returning binary data described in this section. In this case, the driver applies the standard JDBC

binary-to-character conversion to the data, which returns the hexadecimal representation of the binary data.

## 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 returned by 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()  
ResultSet.updateObject()
```

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

---

## **Authentication**

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See “[Authentication](#)” on page 67 for an overview.

The SQL Server driver supports the following methods of authentication:

- SQL Server authentication, or user ID/password authentication, authenticates the user to the database 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.

- NTLM authentication is a single sign-on Windows authentication method. This method provides authentication from Windows clients only and requires minimal configuration.

Except for NTLM authentication, which provides authentication for Windows clients only, these authentication methods provide authentication when the driver is running on any supported platform.

The AuthenticationMethod connection property controls which authentication mechanism the driver uses when establishing connections. See ["Using the AuthenticationMethod Property" on page 429](#) for information about setting the value for this property.

## Using the AuthenticationMethod Property

The AuthenticationMethod connection property controls which authentication mechanism the driver uses when establishing connections. When AuthenticationMethod=auto, the driver uses SQL Server authentication, Kerberos authentication, or NTLM authentication when establishing a connection based on the following criteria:

- If a user ID and password is specified, the driver uses SQL Server authentication when establishing a connection. The User property provides the user ID. The Password property provides the password.

- If a user ID and password is not specified and the driver is not running on a Windows platform, the driver uses Kerberos authentication when establishing a connection.
- If a user ID and password is not specified and the driver is running on a Windows platform, the driver uses NTLM authentication when establishing a connection if the driver can load the DLL required for NTLM authentication. If the driver cannot load the DLL, the driver uses Kerberos authentication.

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

When AuthenticationMethod=ntlm, the driver uses NTLM authentication when establishing a connection if the driver can load the DLL required for NTLM authentication. If the driver cannot load the DLL, the driver throws an exception. The driver ignores any values specified by the User and Password properties.

When AuthenticationMethod=userIdPassword (the default), the driver uses SQL Server authentication when establishing a connection. The User property provides the user ID. The Password property provides the password. If a user ID is not specified, the driver throws an exception.

## Configuring SQL Server Authentication

- 1 Set the AuthenticationMethod property to auto or userIdPassword (the default). See “[Using the AuthenticationMethod Property](#)” on page 429 for more information about setting a value for this property.
- 2 Set the User property to provide the user ID.
- 3 Set the Password property to provide the password.

# Configuring Kerberos Authentication

This section provides requirements and instructions for configuring Kerberos authentication for the Microsoft SQL Server driver.

## ***Product Requirements***

Verify that your environment meets the requirements listed in [Table 7-2](#) before you configure the driver for Kerberos authentication.

---

***Table 7-2. Kerberos Authentication Requirements for the SQL Server Driver***

---

Component	Requirements
Microsoft SQL Server database server	The database server must be administered by the same domain controller that administers the client and must be running one of the following databases: <ul style="list-style-type: none"> <li>■ Microsoft SQL Server 2008</li> <li>■ Microsoft SQL Server 2005</li> <li>■ Microsoft SQL Server 2000</li> <li>■ Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher</li> </ul>
Kerberos server	The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos KDC.  Network authentication must be provided by Windows Active Directory on one of the following operating systems: <ul style="list-style-type: none"> <li>■ Windows Server 2003</li> <li>■ Windows 2000 Server Service Pack 3 or higher</li> </ul>
Client	The client must be administered by the same domain controller that administers the database server. In addition, J2SE 1.4.2 or higher must be installed.

---

## ***Configuring the Driver***

During installation, DataDirect Connect for JDBC installs the following files required for Kerberos authentication in the /lib subdirectory of your DataDirect Connect for JDBC installation directory:

- krb5.conf is a Kerberos configuration file containing values for the Kerberos realm and the KDC name for that realm. DataDirect Connect for JDBC installs a generic file that you must modify for your environment.
- JDBCDriverLogin.conf file is a configuration file that specifies which Java Authentication and Authorization Service (JAAS) login module to use for Kerberos authentication. This file is configured to load automatically unless the java.security.auth.login.config system property is set to load another configuration file. You can modify this file, but the driver must be able to find the JDBC\_DRIVER\_01 entry in this file or another specified login configuration file to configure the JAAS login module. Refer to your JDK documentation for information about setting configuration options in this file

### **To configure the driver:**

- 1 Set the driver's AuthenticationMethod property to auto (the default) or kerberos. See "[Using the AuthenticationMethod Property](#)" on page 429 for more information about setting a value for this property.
- 2 Modify the krb5.conf file to contain your Kerberos realm name and the KDC name for that Kerberos realm. Modify the krb5.conf file by editing the file with a text editor or by specifying the system properties, java.security.krb5.realm and java.security.krb5.kdc.

NOTE: In Windows Active Directory, the Kerberos realm name is the Windows domain name and the KDC name is the Windows domain controller name.

For example, if your Kerberos realm name is XYZ.COM and your KDC name is kdc1, your krb5.conf file would look like this:

```
[libdefaults]
    default_realm = XYZ.COM

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

If the krb5.conf file does not contain a valid Kerberos realm and KDC name, the following exception is thrown:

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

The krb5.conf file installed by DataDirect Connect for JDBC is configured to load automatically unless the java.security.krb5.conf system property is set to point to another Kerberos configuration file.

- 3 If using Kerberos authentication with a Security Manager on a Java 2 Platform, you must grant security permissions to the application and driver. See “[Permissions for Kerberos Authentication](#)” on page 79 for an example.

See the following URL for more information about configuring and testing your environment for Windows authentication with the SQL Server driver:

<http://www.datadirect.com/developer/jdbc/index.ssp>

## Specifying User Credentials for Kerberos Authentication (Delegation of Credentials)

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

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

```
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import java.sql.*;

// The following code creates a javax.security.auth.Subject instance
// used for authentication. Refer to the Java Authentication
// and Authorization Service documentation for details on using a
// LoginContext to obtain a Subject.

LoginContext lc = null;
Subject subject = null;

try {

    lc = new LoginContext("JaasSample", new TextCallbackHandler());
    lc.login();
    subject = lc.getSubject();
}
```

```
catch (Exception le) {
    ... // display login error
}

// This application passes the javax.security.auth.Subject
// to the driver by executing the driver code as the subject

Connection con =
( Connection ) Subject.doAs( subject, new PrivilegedExceptionAction() {

    public Object run() {

        Connection con = null;
        try {

            Class.forName( "com.ddtek.jdbc.sqlserver.SQLServerDriver" );
            String url = "jdbc:datadirect:sqlserver://myServer:1433";
            con = DriverManager.getConnection(url);
        }
        catch (Exception except) {

            ... //log the connection error
            return null;
        }

        return con;
    }
});

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

... // do something with the results
```

## Obtaining a Kerberos Ticket Granting Ticket

To use Kerberos authentication, the application user first must obtain a Kerberos Ticket Granting Ticket (TGT) from the Kerberos server. The Kerberos server verifies the identity of the user and controls access to services using the credentials contained in the TGT.

If the application uses Kerberos authentication from a Windows client, the application user is not required to log onto the Kerberos server and explicitly obtain a TGT. Windows Active Directory automatically obtains a TGT for the user.

If an application uses Kerberos authentication from a UNIX or Linux client, the user must log onto the Kerberos server using the kinit command to obtain a TGT. For example, the following command requests a TGT from the server with a lifetime of 10 hours, which is renewable for 5 days:

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

where *user* is the application user.

Refer to your Kerberos documentation for more information about using the kinit command and obtaining TGTs for users.

# Configuring NTLM Authentication

This section provides requirements and instructions for configuring NTLM authentication for the Microsoft SQL Server driver.

## ***Product Requirements***

Verify that your environment meets the requirements listed in [Table 7-3](#) before you configure your environment for NTLM authentication.

---

***Table 7-3. NTLM Authentication Requirements for the SQL Server Driver***

---

Component	Requirements
Database server	The database server must be administered by the same domain controller that administers the client and must be running on one of the following databases: <ul style="list-style-type: none"> <li>■ Microsoft SQL Server 2008</li> <li>■ Microsoft SQL Server 2005</li> <li>■ Microsoft SQL Server 2000 Service Pack 3 or higher</li> <li>■ Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher</li> </ul>
Domain controller	The domain controller must administer both the database server and the client. Network authentication must be provided by NTLM on one of the following operating systems: <ul style="list-style-type: none"> <li>■ Windows Server 2003</li> <li>■ Windows 2000 Server Service Pack 3 or higher</li> </ul>
Client	The client must be administered by the same domain controller that administers the database server and must be running on one of the following operating systems: <ul style="list-style-type: none"> <li>■ Windows Vista</li> <li>■ Windows Server 2003</li> <li>■ Windows XP Service Pack 2 or higher</li> <li>■ Windows 2000 Service Pack 4 or higher</li> <li>■ Windows NT 4.0</li> </ul> In addition, J2SE 1.4 or higher must be installed.

## Configuring the Driver

DataDirect Connect *for JDBC* provides the following NTLM authentication DLLs (where xx is a two-digit number):

- DDJDBCAuthxx.dll (32-bit)
- DDJDBC64Authxx.dll (Itanium 64-bit)
- DDJDBCx64Authxx.dll (AMD64 and Intel EM64T 64-bit)

The DLLs are located in the *install\_dir/lib* directory (where *install\_dir* is your DataDirect Connect *for JDBC* installation directory). If the application using NTLM authentication is running in a 32-bit JVM, the driver automatically uses DDJDBCAuthxx.dll. Similarly, if the application is running in a 64-bit JVM, the driver uses DDJDBC64Authxx.dll or DDJDBCx64Authxx.dll.

### To configure the driver:

- 1 Set the AuthenticationMethod property to auto (the default) or ntlm. See [“Using the AuthenticationMethod Property” on page 429](#) for more information about setting a value for this property.
- 2 By default, the driver looks for the NTLM authentication DLLs in a directory on the Windows system path defined by the PATH environment variable. If you install the driver in a directory that is not on the Windows system path, perform one of the following actions to ensure the driver can load the DLLs:
  - Add the *install\_dir/lib* directory to the Windows system path, where *install\_dir* is the DataDirect Connect *for JDBC* installation directory.
  - Copy the NTLM authentication DLLs from *install\_dir/lib* to a directory that is on the Windows system path, where *install\_dir* is the DataDirect Connect *for JDBC* installation directory.

- Set the LoadLibraryPath property to specify the location of the NTLM authentication DLLs. For example, if you install the driver in a directory named "DataDirect" that is not on the Windows system path, you can use the LoadLibraryPath property to specify the directory containing the NTLM authentication DLLs:

```
jdbc:datadirect:sqlserver://server3:1521;  
DatabaseName=test;LoadLibraryPath=C:\DataDirect\lib;  
User=test;Password=secret
```

- 3 If using NTLM authentication with a Security Manager on a Java 2 Platform, security permissions must be granted to allow the driver to establish connections. See "[Permissions for Establishing Connections](#)" on page 77 for an example.

---

## Data Encryption

The SQL Server driver supports SSL encryption for the following databases:

- Microsoft SQL Server 2008
- Microsoft SQL Server 2005
- Microsoft SQL Server 2000 or higher
- Microsoft SQL Server 2000 Enterprise Edition (64-bit) or higher

SSL secures the integrity of your data by encrypting information and providing authentication. See "[Data Encryption Across the Network](#)" on page 69 for an overview.

Depending on your Microsoft SQL Server configuration, you can choose to encrypt all data, including the login request, or

encrypt the login request only. Encrypting login requests, but not data, is useful for the following scenarios:

- When your application needs security, but cannot afford to pay the performance penalty for encrypting data transferred between the driver and server.
- Microsoft SQL Server 2005 and higher only. When the server is not configured for SSL, but your application still requires a minimum degree of security.

NOTE: When SSL is enabled, the driver communicates with database protocol packets set by the server's default packet size. Any value set by the `PacketSize` property is ignored.

## Using SSL with Microsoft SQL Server

If your Microsoft SQL Server database server has been configured with an SSL certificate signed by a trusted CA, the server can be configured so that SSL encryption is either optional or required. When required, connections from clients that do support SSL encryption fail.

Although a signed trusted SSL certificate is recommended for the best degree of security, Microsoft SQL Server 2005 and higher can provide limited security protection even if an SSL certificate has not been configured on the server. If a trusted certificate is not installed, the server will use a self-signed certificate to encrypt the login request, but not the data.

**Table 7-4** shows how the different `EncryptionMethod` property values behave with different Microsoft SQL Server configurations.

---

**Table 7-4. *EncryptionMethod Property and Microsoft SQL Server Configurations***

---

Value	No SSL Certificate	SSL Certificate	
		SSL Optional	SSL Required
noEncryption	Login request and data are not encrypted.	Login request and data are not encrypted.	Connection attempt fails.
SSL	Connection attempt fails.	Login request and data are encrypted.	Login request and data are encrypted.
requestSSL	Login request and data are not encrypted.	Login request and data are encrypted.	Login request and data are encrypted.
loginSSL	Microsoft SQL Server 2005 and higher: Login request is encrypted, but data is not encrypted  Microsoft SQL Server 2000: Connection attempt fails.	Login request is encrypted, but data is not encrypted.	Login request and data are encrypted.

---

## Configuring SSL Encryption

1 Choose the type of encryption for your application:

- If you want the driver to encrypt all data, including the login request, set the `EncryptionMethod` property to `SSL` or `requestSSL`.
- If you want the driver to encrypt only the login request, set the `EncryptionMethod` property to `loginSSL`.

- 2 Specify the location and password of the truststore file used for SSL server authentication. Either set the TrustStore and TrustStore properties or their corresponding Java system properties (javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword, respectively).
- 3 To validate certificates sent by the database server, set the ValidateServerCertificate property to true.
- 4 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.

---

## DML with Results (Microsoft SQL Server 2005 and Higher)

The SQL Server driver supports the Microsoft SQL Server 2005 and higher Output clause for Insert, Update, and Delete statements. For example, suppose you created a table with the following statement:

```
CREATE TABLE table1(id int, name varchar(30))
```

The following Update statement updates the values in the id column of table1 and returns a result set that includes the old ID (replaced by the new ID), the new ID, and the name associated with these IDs:

```
UPDATE table1 SET id=id*10 OUTPUT deleted.id as oldId,  
inserted.id as newId, inserted.name
```

The driver returns the results of Insert, Update, or Delete statements and the update count in separate result sets. The output result set is returned first, followed by the update count

for the Insert, Update, or Delete statement. To execute DML with Results statements in an application, use the Statement.execute() or PreparedStatement.execute() method. Then, use Statement.getMoreResults () to obtain the output result set and the update count. For example:

```
String sql = "UPDATE table1 SET id=id*10 OUTPUT deleted.id as oldId,
    inserted.id as newId, inserted.name";
boolean isResultSet = stmt.execute(sql);

int    updateCount = 0;
while (true) {

    if (isResultSet) {
        resultSet = stmt.getResultSet();
        while (resultSet.next()) {

            System.out.println("oldId: " + resultSet.getInt(1) +
                "newId: " + resultSet.getInt(2) +
                "name: " + resultSet.getString(3));
        }
        resultSet.close();
    }
    else {
        updateCount = stmt.getUpdateCount();
        if (updateCount == -1) {
            break;
        }

        System.out.println("Update Count: " + updateCount);
    }

    isResultSet = stmt.getMoreResults();
}
```

---

## Reauthentication

The SQL Server driver supports reauthentication for Microsoft SQL Server 2005 and higher. The user performing the switch must have been granted the database permission IMPERSONATE.

See “[Using Reauthentication](#)” on page 55 for an introduction to reauthentication. See [Appendix I “Connection Pool Manager” on page 789](#) for information about using reauthentication with the DataDirect Connection Pool Manager.

**NOTE:** Before performing reauthentication, applications must ensure that any statements or result sets created as one user are closed before switching the connection to another user.

Your application can use the `setCurrentUser()` method in the `ExtConnection` interface located in the `com.ddtek.jdbc.extensions` package to switch a user on a connection. The `setCurrentUser()` method accepts driver-specific reauthentication options. The reauthentication options supported for the SQL Server driver are:

<code>CURRENT_DATABASE</code>	Specifies the name of the current database. The value must be a valid Microsoft SQL Server database name. If the <code>setCurrentUser()</code> method is called and this option is specified as an empty string or is not specified, only the user is switched; the database is not switched.
-------------------------------	--

**REVERT\_USER**

{true | false}. Determines whether the driver reverts the current user to the initial user before setting the user to a new user for connections that have already reauthenticated.

If set to true and the `setCurrentUser()` method is called, the driver reverts the current user to the initial user before setting the connection to the new user. For example, consider a connection that was initially created by User A and was later switched to User B. Before the connection could be further switched to User C, the driver reverts the connection back to User A and then sets it to User C.

If set to false and the `setCurrentUser()` method is called, the driver does not revert the current user to the initial user before performing the switch. For example, if the connection was initially created by User A, switched to User B, and then switched to User C, the driver does not revert the user to User A before switching to User C.

See “[ExtConnection Interface](#)” on page 613 for more information about the `setCurrentUser()` method.

---

## Client Information for Connections

The SQL Server driver allows applications to store and return the following types of client information associated with a particular connection:

- Name of the application
- User ID
- Host name of the client
- Additional accounting information, such as an accounting ID
- Product name and version of the SQL Server driver

This information can be used for database administration and monitoring purposes. See [Appendix C “Client Information for Connections” on page 621](#) for details.

---

## SQL Escape Sequences

See [Appendix F “SQL Escape Sequences for JDBC” on page 717](#) for information about the SQL escape sequences supported by the SQL Server driver.

---

## Isolation Levels

The SQL Server driver supports the following isolation levels for Microsoft SQL Server:

- Read Committed with Locks \* or Read Committed
- Read Committed with Snapshots \*
- Read Uncommitted
- Repeatable Read
- Serializable
- Snapshot \*

\* Supported for Microsoft SQL Server 2005 and higher only.

The default is Read Committed with Locks (Microsoft SQL Server 2005 and higher) or Read Committed.

---

## Using the Snapshot Isolation Level (Microsoft SQL Server 2005 and Higher)

You can use the Snapshot isolation level in either of the following ways:

- Setting the `SnapshotSerializable` property changes the behavior of the Serializable isolation level to use the Snapshot isolation level. This allows an application to use the Snapshot isolation level with no or minimum code changes. See the [“SnapshotSerializable” on page 410](#) for more information.
- Importing the `ExtConstants` class allows you to specify the `TRANSACTION_SNAPSHOT` or `TRANSACTION_SERIALIZABLE` isolation levels for an individual statement in the same application. The `ExtConstants` class in the `com.ddtek.jdbc.extensions` package defines the

TRANSACTION\_SNAPSHOT constant. For example, the following code imports the ExtConstants class and sets the TRANSACTION\_SNAPSHOT isolation level:

```
import com.ddtek.jdbc.extensions.ExtConstants;  
Connection.setTransactionIsolation(ExtConstants.TRANSACTION_SNAPSHOT);
```

---

## Using Scrollable Cursors

The SQL Server driver supports scroll-sensitive result sets, scroll-insensitive result sets, and updatable result sets.

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

---

## Server-Side Updatable Cursors

The SQL Server driver can use client-side cursors or server-side cursors to support updatable result sets. By default, the SQL Server driver uses client-side cursors because this type of cursor can work with any result set type. Using server-side cursors typically can improve performance, but server-side cursors cannot be used with scroll-insensitive result sets or with scroll-sensitive result sets that are not generated from a database table that contains a primary key. To use server-side cursors, set the UseServerSideUpdatableCursors property to true.

When the UseServerSideUpdatableCursors property is set to true and a scroll-insensitive updatable result set is requested, the driver downgrades the request to a scroll-insensitive read-only result set. Similarly, when a scroll-sensitive updatable result set is

requested and the table from which the result set was generated does not contain a primary key, the driver downgrades the request to a scroll-sensitive read-only result set. In both cases, a warning is generated.

When server-side updatable cursors are used with sensitive result sets that are generated from a database table that contains a primary key, the following changes you make to the result set are visible:

- Own Inserts are visible. Others Inserts are not visible.
- Own and Others Updates are visible.
- Own and Others Deletes are visible.

Using the default behavior of the driver (UseServerSideUpdatableCursors=false), those changes are not visible.

---

## JTA Support: Installing Stored Procedures

To use JDBC distributed transactions through JTA, use the following procedure to install Microsoft SQL Server JDBC XA procedures. Repeat this procedure for any Microsoft SQL Server installation that uses distributed transactions.

If you have multiple instances of Microsoft SQL Server on the same machine, you can edit the instjdbc.sql script with a text editor to specify a fully qualified path to the sqljdbc.dll file for a particular instance. For example, if you want to install XA Procedures for an instance named "MSSQL.2," modify the instjdbc.sql script as shown and run it as described in the following procedure.

```
/*
**  add references for the stored procedures
*/

print 'creating JDBC XA procedures'
go

sp_addextendedproc 'xp_jdbc_open',
'C:\Program Files\Microsoft SQL Server\MSSQL.2\MSSQL\Binn\sqljdbc.dll'
go
sp_addextendedproc 'xp_jdbc_open2',
'C:\Program Files\Microsoft SQL Server\MSSQL.2\MSSQL\Binn\sqljdbc.dll'
go
sp_addextendedproc 'xp_jdbc_close',
'C:\Program Files\Microsoft SQL Server\MSSQL.2\MSSQL\Binn\sqljdbc.dll'
go
sp_addextendedproc 'xp_jdbc_close2',
'C:\Program Files\Microsoft SQL Server\MSSQL.2\MSSQL\Binn\sqljdbc.dll'
go
sp_addextendedproc 'xp_jdbc_start',
'C:\Program Files\Microsoft SQL Server\MSSQL.2\MSSQL\Binn\sqljdbc.dll'
...
...
```

**NOTE:** For Microsoft SQL Server 2005 and higher, you can use the Microsoft SQL Server Configuration Manager tool to view Microsoft SQL Server services and determine the fully qualified path to the binn subdirectory of each Microsoft SQL Server instance on a machine. Using the Configuration Manager, right-click on a service and select Properties. Select the Service tab. The path is shown as a value of the Binary Path attribute. For Microsoft SQL Server 2000, you can use the SQL Server Enterprise Manager. Refer to your Microsoft SQL Server documentation for details.

**To install stored procedures for JTA:**

- 1 Copy the appropriate 32-bit or 64-bit sqljdbc.dll file to the *SQL\_Server\_Root/bin* directory of the Microsoft SQL Server database server:

<b>sqljdbc.dll Version</b>	<b>File Location</b>
32-bit	<i>install_dir/SQLServer JTA/32-bit</i>
64-bit Itanium	<i>install_dir/SQLServer JTA/64-bit</i>
64-bit AMD64 and Intel EM64T	<i>install_dir/SQLServer JTA/x64-bit</i>

where:

*install\_dir* is your DataDirect Connect *for JDBC* installation directory.

*SQL\_Server\_Root* is your Microsoft SQL Server installation directory.

- 2 From the database server, use the ISQL utility to run the instjdbc.sql script. As a precaution, have your system administrator back up the master database before running instjdbc.sql.

At a command prompt, run instjdbc.sql:

```
ISQL -Usa -Psa_password -Sserver_name
-olocation\instjdbc.sql
```

where:

*sa\_password* is the password of the system administrator.

*server\_name* is the name of the server on which the Microsoft SQL Server database resides.

*location* is the full path to instjdbc.sql. This script is located in the *install\_dir/SQLServer JTA* directory where *install\_dir* is your DataDirect Connect *for JDBC* installation directory.

- 3 The instjdbc.sql script generates many messages. In general, these messages can be ignored; however, the system

administrator should scan the output for any messages that may indicate an execution error. The last message should indicate that `instjdbc.sql` ran successfully. The script fails when there is insufficient space available in the master database to store the JDBC XA procedures or to log changes to existing procedures.

---

## Distributed Transaction Cleanup

Connections associated with distributed transactions can become orphaned if the connection to the server is lost before the transaction has completed. When connections associated with distributed transactions are orphaned, any locks held by the database for that transaction are maintained, which can cause data to become unavailable. By cleaning up distributed transactions, connections associated with those transactions are freed and any locks held by the database are released.

You can use the `XAResource.recover` method to clean up distributed transactions that have been prepared, but not committed or rolled back. Calling this method returns a list of active distributed transactions that have been prepared, but not committed or rolled back. An application can use the list returned by the `XAResource.recover` method to clean up those transactions by explicitly committing them or rolling them back. The list of transactions returned by the `XAResource.recover` method does not include transactions that are active and have not been prepared.

In addition, the SQL Server driver supports the following methods of distributed transaction cleanup:

- `Transaction timeout` sets a timeout value that is used to audit active transactions. Any active transactions that have a life span greater than the specified timeout value are rolled back.

Setting a transaction timeout allows distributed transactions to be cleaned up automatically based on the timeout value.

- Explicit transaction cleanup allows you to explicitly roll back any transactions left in an unprepared state based on a transaction group identifier. Explicit transaction cleanup provides more control than transaction timeout over when distributed transactions are cleaned up.

## Transaction Timeout

To set a timeout value for transaction cleanup, you use the XAResource.setTransactionTimeout method. Setting this value causes sqljdbc.dll on the server side to maintain a list of active transactions. Distributed transactions are placed in the list of active transactions when they are started and removed from this list when they are prepared, rolled back, committed, or forgotten using the appropriate XAResource methods.

When a timeout value is set for transaction cleanup using the XAResource.setTransactionTimeout method, sqljdbc.dll periodically audits the list of active transactions for expired transactions. Any active transactions that have a life span greater than the timeout value are rolled back. If an exception is generated when rolling back a transaction, the exception is written to the sqljdbc.log file, which is located in the same directory as the sqljdbc.dll file.

Setting the transaction timeout value too low means running the risk of rolling back a transaction that otherwise would have completed successfully. As a general guideline, set the timeout value to allow sufficient time for a transaction to complete under heavy traffic load.

Setting a value of 0 (the default) disables transaction timeout cleanup.

## Explicit Transaction Cleanup

The SQL Server driver allows you to associate an identifier with a group of transactions using the XATransactionGroup connection property. When you specify a transaction group ID, all distributed transactions initiated by the connection are identified by this ID.

Setting this value causes sqljdbc.dll on the server side to maintain a list of active transactions. Distributed transactions are placed in the list of active transactions when they are started and removed from this list when they are prepared, rolled back, committed, or forgotten using the appropriate XAResource methods.

You can use the XAResource.recover method to roll back any transactions left in an unprepared state that match the transaction group ID on the connection used to call XAResource.recover. For example, if you specified XATransactionGroup=ACCT200 and called the XAResource.recover method on the same connection, any transactions left in an unprepared state with a transaction group ID of ACCT200 would be rolled back.

If an exception is generated when rolling back a transaction, the exception is written to the sqljdbc.log file, which is located in the same directory as the sqljdbc.dll file.

When using explicit transaction cleanup, distributed transactions associated with orphaned connections, and the locks held by those connections, will persist until the application explicitly invokes them. As a general rule, applications should clean up orphaned connections at startup and when the application is notified that a connection to the server was lost.

---

## Large Object (LOB) Support

Although Microsoft SQL Server does not define a Blob or Clob data type, the SQL Server driver allows you to return and update long data, specifically LONGVARBINARY and LONGVARCHAR 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 advantages as retrieving and updating Blobs and Clobs. For example, using Blobs and Clobs:

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

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

---

## Batch Inserts and Updates

The SQL Server driver implementation for batch Inserts and Updates is JDBC 3.0 compliant. When the SQL Server driver detects an error in a statement or parameter set in a batch Insert or Update, it generates a BatchUpdateException and continues to execute the remaining statements or parameter sets in the batch. The array of update counts contained in the BatchUpdateException contain one entry for each statement or

parameter set. Any entries for statements or parameter sets that failed contain the value Statement.EXECUTE\_FAILED.

---

## Parameter Metadata Support

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

### Insert and Update Statements

The SQL Server driver supports returning parameter metadata for the following forms of Insert and Update statements:

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

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

### Select Statements

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

Parameter metadata can be returned for a Select statement if one of the following conditions is true:

- The statement contains a predicate value expression that can be targeted against the source tables in the associated FROM clause. For example:

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

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

- The statement contains a predicate value expression part that is a nested query. The nested query's metadata must describe a single column. For example:

```
SELECT * FROM foo WHERE (SELECT x FROM y  
WHERE z = 1) < ?
```

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

```
SELECT col1, col2 FROM foo WHERE col1 = ? and col2 > ?
```

```
SELECT ... WHERE colname = (SELECT col2 FROM t2  
WHERE col3 = ?)
```

```
SELECT ... WHERE colname LIKE ?
```

```
SELECT ... WHERE colname BETWEEN ? and ?
```

```
SELECT ... WHERE colname IN (?, ?, ?)
```

```
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE col1 < ?)
```

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

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

Joins are supported. For example:

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

Fully qualified names and aliases are supported. For example:

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

## Stored Procedures

The SQL Server driver does not support returning parameter metadata for stored procedure arguments.

---

## ResultSet MetaData Support

If your application requires table name information, the SQL Server driver can return table name information in ResultSet metadata for Select statements. By setting the ResultSetMetaDataOptions property to 1, the SQL Server 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 SQL Server 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 SQL Server 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 SQL Server 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 SQL Server driver also can return schema name and catalog name information when the `ResultSetMetaData.getSchemaName()` and `ResultSetMetaData.getCatalogName()` methods are called if the driver can determine that information. For example, for the following statement, the SQL Server driver returns "test" for the catalog name, "test1" for the schema name, and "foo" for the table name:

```
SELECT * FROM test.test1.foo
```

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

---

## Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

See <http://www.jcp.org/en/jsr/detail?id=114> for more information about JSR 114.

---

## Auto-Generated Keys Support

The SQL Server driver supports retrieving the values of auto-generated keys. An auto-generated key returned by the SQL Server driver is the value of an identity 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 that contains parameters:

- When using an Insert statement that contains no parameters, the SQL Server 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 that contains parameters,  
the SQL Server 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*)

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.

See “[Retrieving Auto-Generated Keys](#)” on page 709 for  
information about how auto-generated keys can improve  
performance.

---

## Null Values

When the Microsoft SQL Server driver establishes a connection, the driver sets the Microsoft SQL Server database option `ansi_nulls` to on. This action ensures that the driver is compliant with the ANSI SQL standard, which makes developing cross-database applications easier.

By default, Microsoft SQL Server does not evaluate null values in SQL equality (=) or inequality (<>) comparisons or aggregate functions in an ANSI SQL-compliant manner. For example, the ANSI SQL specification defines that `col1=null` as shown in the following Select statement always evaluates to false:

```
SELECT * FROM table WHERE col1 = NULL
```

Using the default database setting (`ansi_nulls=off`), the same comparison evaluates to true instead of false.

Setting `ansi_nulls` to on changes how the database handles null values and forces the use of `IS NULL` instead of `=NULL`. For example, if the value of `col1` in the following Select statement is null, the comparison evaluates to true:

```
SELECT * FROM table WHERE col1 IS NULL
```

In your application, you can restore the default Microsoft SQL Server behavior for a connection in the following ways:

- Use the `InitializationString` property to specify the SQL command `set ANSI_NULLS off`. For example, the following URL ensures that the handling of null values is restored to the Microsoft SQL Server default for the current connection:

```
jdbc:datadirect:sqlserver://server1:1433;  
InitializationString=set ANSI_NULLS off;  
DatabaseName=test
```

- Explicitly execute the following statement after the connection is established:

```
SET ANSI_NULLS OFF
```

---

## Configuring Failover

- 1 Specify the primary and alternate servers:

- Specify your primary server using a connection URL or data source.
- Specify one or multiple alternate servers by setting the AlternateServers property.

See “[Specifying Primary and Alternate Servers](#)” on page 464.

NOTE: If using failover with Microsoft Cluster Server (MSCS), which determines the alternate server for failover instead of the driver, any alternate server specified must be the same as the primary server. For example:

```
jdbc:datadirect:sqlserver://server1:1433;  
DatabaseName=TEST;User=test;Password=secret;  
AlternateServers=(server1:1433;DatabaseName=TEST)
```

- 2 Choose a failover method by setting the FailoverMode connection property. The default method is connection failover (FailoverMode=connect). See “[Using Failover](#)” on page 58 for an overview of each failover method.
- 3 If FailoverMode=extended or FailoverMode=select, set the FailoverGranularity property to specify how you want the driver to behave if exceptions occur while trying to reestablish a lost connection. The default behavior of the driver is to continue with the failover process and post any exceptions on the statement on which they occur (FailoverGranularity=nonAtomic).

- 4 Optionally, configure the connection retry feature. See ["Specifying Connection Retry" on page 467](#).
- 5 Optionally, set the FailoverPreconnect property if you want the driver to establish a connection with the primary and an alternate server at the same time. The default behavior is to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection (FailoverPreconnect=false).

## Specifying Primary and Alternate Servers

Connection information for primary and alternate servers can be specified using either one of the following methods:

- Connection URL through the JDBC Driver Manager
- JDBC data source

For example, the following connection URL for the SQL Server driver specifies connection information for the primary and alternate servers using a connection URL:

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=TEST;User=test;  
Password=secret;AlternateServers=(server2:1433;DatabaseName=TEST2,  
server3:1433;DatabaseName=TEST3)
```

In this example:

```
...server1:1433;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=(server2:1433;DatabaseName=TEST2,  
server3:1433;DatabaseName=TEST3)
```

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

```
SQLServerDataSource mds = new SQLServerDataSource();
mds.setDescription("My SQLServerDataSource");
mds.setServerName("server1");
mds.setPortNumber(1433);
mds.setDatabaseName("TEST");
mds.setUser("test");
mds.setPassword("secret");
mds.setAlternateServers(server2:1433;DatabaseName=TEST2,
    server3:1433;DatabaseName=TEST3)
```

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

The SQL Server driver also allows you to specify connections to named instances, multiple instances of a Microsoft SQL Server database running concurrently on the same server. If specifying named instances for the primary and alternate servers, the connection URL would look like this:

```
jdbc:datadirect:sqlserver://server1\\instance1;User=test;Password=secret;
AlternateServers=(server2\\instance2:1433;DatabaseName=TEST2,
server3\\instance3:1433;DatabaseName=TEST3)
```

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

```
SQLServerDataSource mds = new SQLServerDataSource();
mds.setDescription("My SQLServerDataSource");
mds.setServerName("server1\\instance1");
mds.setPortNumber(1433);
mds.setDatabaseName("TEST");
mds.setUser("test");
mds.setPassword("secret");
```

```
mds.setAlternateServers(server2\\instance2:1433;  
    DatabaseName=TEST2,server3\\instance3:1433;  
    DatabaseName=TEST3)
```

To connect to a named instance using a data source, you specify the named instance on the primary server using the `ServerName` property.

See “[Connecting to Named Instances](#)” on page 420 for more information about connecting to named instances on Microsoft SQL Server.

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

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

or, if connecting to named instances:

```
(servername1\\instance1[;property=value][,servername2\\instance2  
[;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.

`instance1` is the named instance on the first alternate database server, `servername2` is the named instance on the second alternate database server, and so on. If connecting to named instances, the named instance 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. The port number is 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 1433 is used.

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

```
jdbc:datadirect:sqlserver://server1:1433;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:1433;DatabaseName=TEST2,
server3:1433;DatabaseName=TEST3)
```

or, if connecting to named instances:

```
jdbc:datadirect:sqlserver://server1\\instance1:1433;DatabaseName=TEST;
User=test;Password=secret;AlternateServers=(server2\\instance2:1433;
DatabaseName=TEST2,server3\\instance3:1433;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:sqlserver://server1:1433;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:1433,server3:1433)
```

## Specifying Connection Retry

Connection retry allows the SQL Server 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:sqlserver://server1:1433;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:1433;DatabaseName=TEST2,
server3:1433;DatabaseName=TEST3);ConnectionRetryCount=2;
ConnectionRetryDelay=5
```

In this example, if a successful connection is not established on the SQL Server 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.

## Failover Properties

[Table 7-5](#) summarizes the connection properties that control how failover works with the SQL Server driver. See “[Connection Properties](#)” on page 375 for details about configuring each property.

---

**Table 7-5. Summary: Failover Properties for the SQL Server Driver**

---

Property	Characteristic
AlternateServers	One or multiple alternate database servers. An IP address or server name identifying each server is required. Port number and the connection property <code>DatabaseName</code> are optional. If the port number is unspecified, the port number specified for the primary server is used. If a port number is unspecified for the primary server, the default port number of 1433 is used.
ConnectionRetryCount	Number of times the driver retries the primary database server, and if specified, alternate servers until a successful connection is established. The default is 5.
ConnectionRetryDelay	Wait interval, in seconds, between connection retry attempts when the <code>ConnectionRetryCount</code> property is set to a positive integer. The default is 1.
DatabaseName	Name of the database to which you want to connect.

**Table 7-5. Summary: Failover Properties for the SQL Server Driver (cont.)**

<b>Property</b>	<b>Characteristic</b>
FailoverGranularity	Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. The default is nonAtomic (the driver continues with the failover process and posts any exceptions on the statement on which they occur).
FailoverMode	The failover method you want the driver to use. The default is connect (connection failover is used).
FailoverPreconnect	Specifies whether the driver tries to connect to the primary and an alternate server at the same time. The default is false (the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection).
LoadBalancing	Sets whether the driver will use client load balancing in its attempts to connect to the database servers (primary and alternate). If client load balancing is enabled, the driver uses a random pattern instead of a sequential pattern in its attempts to connect. The default is false (client load balancing is disabled).
PortNumber	Port listening for connections on the primary database server. This property is supported only for data source connections. The default port number is 1433.
ServerName	IP address or server name for the primary database server. This property is supported only for data source connections.

See “[Using Failover](#)” on page 58 and “[Using Client Load Balancing](#)” on page 65 for overviews of failover and client load balancing.

## Bulk Load

The driver supports DataDirect Bulk Load, a feature that allows your application to send large numbers of rows of data to the database in a continuous stream instead of in numerous smaller database protocol packets. Similar to batch operations, performance improves because far fewer network round trips are required. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations. See ["Using DataDirect Bulk Load" on page 829](#) for more information.

# 8 The Sybase Driver

The DataDirect Connect *for JDBC* Sybase driver (the "Sybase driver") supports:

- Sybase Adaptive Server Enterprise 15.0
- Sybase Adaptive Server Enterprise 12.0, 12.5, 12.5.1, 12.5.2, 12.5.3, and 12.5.4
- Sybase Adaptive Server 11.5 and 11.9

---

## Data Source and Driver Classes

The data source class for the Sybase driver is:

`com.ddtek.jdbcx.sybase.SybaseDataSource`

See "[Connecting Through Data Sources](#)" on page [48](#) for information about DataDirect Connect *for JDBC* data sources.

The driver class for the Sybase driver is:

`com.ddtek.jdbc.sybase.SybaseDriver`

## Connection URL

The connection URL format for the Sybase driver is:

```
jdbc:datadirect:sybase://hostname:port[;property=value[;...]]
```

where:

*hostname* is the TCP/IP address or TCP/IP host name of the server to which you are connecting. See “[Using IP Addresses](#)” on page 50 for details on using IP addresses.

NOTE: Untrusted applets cannot open a socket to a machine other than the originating host.

*port* is the number of the TCP/IP port.

*property=value* specifies connection properties. For a list of connection properties and their valid values, see “[Connection Properties](#)” on page 473.

For example:

```
jdbc:datadirect:sybase://server2:5000;User=test;Password=secret
```

---

## J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the Sybase resource adapter is:

```
com.ddtek.resource.spi.SybaseManagedConnectionFactory
```

See “[J2EE Connector Architecture Resource Adapters](#)” on page 37 for information about using DataDirect Connect for JDBC drivers as J2EE Connector Architecture resource adapters.

---

# Connection Properties

This section lists the JDBC connection properties supported by the Sybase driver and describes each property. The properties have the form:

*property=value*

You can use these connection properties with either the JDBC Driver Manager or DataDirect Connect for JDBC data sources unless otherwise noted.

## NOTES:

- All connection property names are case-insensitive. For example, Password is the same as password. Required properties are noted as such.
- The data type listed for each connection property is the Java data type used for the property value in a JDBC data source.

## AccountingInfo

Description	Accounting information to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.
Valid Values	<i>string</i> where <i>string</i> is the accounting information.
Default	empty string
Data Type	String
See Also	<a href="#">"Client Information for Connections" on page 521</a>

## AlternateServers

Description	A list of alternate database servers that is used to failover new or lost connections, depending on the failover method selected. See the <a href="#">FailoverMode</a> property for information about choosing a failover method.
Valid Values	<code>(servername1[:port1][;DatabaseName=value])</code> <code>[,servername2[:port2][;DatabaseName=value]]...)</code>
	The server name ( <code>servername1</code> , <code>servername2</code> , and so on) is required for each alternate server entry. Port number ( <code>port1</code> , <code>port2</code> , and so on) and connection properties ( <code>property=value</code> ) are optional for each alternate server entry. If the port is unspecified, the port number of the primary server is used. If the port is unspecified, the port number of the primary server is used.
	The driver only allows one optional connection property, <a href="#">DatabaseName</a> .
Example	The following URL contains alternate server entries for <code>server2</code> and <code>server3</code> . The alternate server entries contain the optional <a href="#">DatabaseName</a> property.
	<code>jdbc:datadirect:sybase://server1:4100;DatabaseName=TEST;</code> <code>User=test;Password=secret;AlternateServers=(server2:4100;</code> <code>DatabaseName=TEST2,server3:4100;DatabaseName=TEST3)</code>
Default	None
Data type	String
See Also	<a href="#">"Configuring Failover" on page 529</a>

## ApplicationName

Description	The name of the application to be stored in the database. This value sets the <code>clientapplname</code> and <code>program_name</code> values in the <code>sysprocesses</code> table. This value is used for database administration/monitoring purposes.
-------------	--

Valid Values *string*

where *string* is the name of the application.

NOTE: Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. See “[Returning MetaData About Client Information Locations](#)” on page 627.

Default empty string

Data Type String

See Also [“Client Information for Connections” on page 521](#)

## AuthenticationMethod

Description 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 kerberos | userIdPassword

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

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

NOTE: The [User](#) property provides the user ID. The [Password](#) property provides the password.

Default clearText

Data Type String

See Also [“Authentication” on page 512](#)

## BatchPerformanceWorkaround

Description Determines the method that is used to execute batch operations.

Valid Values true | false

If set to true, the driver uses the native Sybase batch mechanism. In most cases, using the native Sybase batch functionality provides significantly better performance, but the driver may not always be able to return update counts for the batch.

If set to false, the driver uses the JDBC 3.0-compliant batch mechanism.

Default false

Data Type boolean

See Also ["Batch Inserts and Updates" on page 523](#)

["Performance Considerations" on page 507](#)

## BulkLoadBatchSize

Description 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.

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 set by this property. See ["DDBulkLoad Interface" on page 603](#) for a description of the method.

Valid Values

*x*

where *x* is a positive integer.

Default

2048

Data Type

long

See Also

["Using DataDirect Bulk Load" on page 829](#)

## ClientHostName

Description

The host name of the client machine to be stored in the database. This value sets the clienthostname and hostname values in the sysprocesses table. This value is used for database administration/monitoring purposes.

Valid Values

*string*

where *string* is the host name of the client machine.

NOTE: Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. See ["Returning MetaData About Client Information Locations" on page 627](#).

Default

empty string

Data Type

String

See Also

["Client Information for Connections" on page 521](#)

## ClientUser

Description

The user ID to be stored in the database. This value sets the clientname value in the sysprocesses table in the database. This value is used for database administration/monitoring purposes.

Valid Values

*string*

where *string* is a valid user ID.

NOTE: Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. See “[Returning MetaData About Client Information Locations](#)” on page 627.

Default	empty string
Data Type	String
See Also	<a href="#">“Client Information for Connections” on page 521</a>

## CodePageOverride

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

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.

Valid Values *string*  
where *string* is the name of a valid code page supported by your JVM.

Example CP950

Default None

Data Type String

## ConnectionRetryCount

Description The number of times the driver retries connection attempts to the primary database server, and if specified, alternate servers until a successful connection is established.

NOTE: 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.

Valid Values `0 | x`

where `x` is a positive integer.

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.

NOTE: The [ConnectionRetryDelay](#) property specifies the wait interval, in seconds, to occur between retry attempts.

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.

Default 5 (seconds)

Data Type int

See Also ["Configuring Failover" on page 529](#)

## ConnectionRetryDelay

Description 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.

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.

**Default** 1 (second)

**Data Type** int

**See Also** ["Configuring Failover" on page 529](#)

## ConvertNull

**Description** Controls how data conversions are handled for null values.

**Valid Values** 0 | 1

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

If set to 1, the driver checks the data type being requested against the data type of the table column storing 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 the data type of the column value.

**Default** 1

**Data Type** int

## Database

Description An alias for the [DatabaseName](#) property.

## DatabaseName

Description The name of the database to which you want to connect.

Valid Values *string*

where *string* is the name of a Sybase database.

Default None

Data Type String

Alias [Database](#) property. If both the Database and DatabaseName properties are specified in a connection URL, the last property that is positioned in the connection URL is used. For example, if your application specifies the following connection URL, the value of the Database connection property would be used instead of the value of the DatabaseName connection property.

```
jdbc:datadirect:sybase://server1:1433;DatabaseName=jdbc;  
Database=acct;User=test;Password=secret
```

## EnableBulkLoad

Description Specifies whether the driver uses the native bulk load protocols in the database instead of the batch mechanism for batch inserts. Bulk load bypasses the data parsing that is usually done by the database, providing an additional performance gain over batch operations. This property allows existing applications with batch inserts to take advantage of bulk load without requiring changes to the application code.

Valid Values true | false

If set to true, the driver uses the native bulk load protocols for batch inserts.

If set to false, the driver uses the batch mechanism for batch inserts.

Default false

Data Type boolean

See Also ["Using Bulk Load for Batch Inserts" on page 837](#)

["Performance Considerations" on page 507](#)

## EnableCancelTimeout

Description 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.

Default false

Data Type boolean

## EncryptionMethod

Description	Determines whether data is encrypted and decrypted when transmitted over the network between the driver and database server.  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 <a href="#">LoginTimeout</a> property to avoid problems when connecting to a server that does not support SSL.
Valid Values	<code>noEncryption   SSL</code>  If set to <code>noEncryption</code> , data is not encrypted or decrypted.  If set to <code>SSL</code> , data is encrypted using SSL. If the database server does not support SSL, the connection fails and the driver throws an exception. When SSL is enabled, the following properties also apply:  <a href="#">HostNameInCertificate</a> <a href="#">TrustStore</a> <a href="#">TrustStorePassword</a> <a href="#">ValidateServerCertificate</a>
Default	<code>noEncryption</code>
Data Type	String
See Also	<a href="#">“Data Encryption” on page 520</a> <a href="#">“Performance Considerations” on page 507</a>

## ErrorBehavior

Description	Determines how the driver handles errors that are returned from stored procedures.
Valid Values	<p>exception   warning   raiseerrorwarning</p> <p>If set to exception, the driver throws an exception when it encounters stored procedure errors, including RAISERRORs.</p> <p>If set to warning, the driver returns stored procedure errors, including RAISERRORs, as SQLWarnings.</p> <p>If set to raiseerrorwarning, the driver returns RAISERRORs as SQLWarnings and throws exceptions for other stored procedure errors.</p>
	<p>NOTE: By default, older versions of the Sybase driver converted errors returned from a stored procedure into SQLWarnings. Applications that relied on the driver converting errors to warnings can revert to that behavior by setting ErrorBehavior=warning.</p>
Default	exception
Data Type	String

## FailoverGranularity

Description	Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. This property is ignored if <a href="#">FailoverMode=connect</a> .
Valid Values	<p>nonAtomic   atomic   atomicWithRepositioning   disableIntegrityCheck</p> <p>If set to nonAtomic, the driver continues with the failover process and posts any exceptions on the statement on which they occur.</p>

If set to atomic, the driver fails the entire failover process if an exception is generated as the result of restoring the state of the connection. If an exception is generated as a result of restoring the state of work in progress, the driver continues with the failover process, but generates an exception warning that the Select statement must be reissued.

If set to atomicWithRepositioning, the driver fails the entire failover process if any exception is generated as the result of restoring the state of the connection or the state of work in progress.

If set to disableIntegrityCheck, the driver does not verify that the rows that are restored during the failover process match the original rows. This value is applicable only when FailoverMode=select.

**Default** nonAtomic

**Data Type** String

**See Also** [“Configuring Failover” on page 529](#)

## **FailoverMode**

**Description** Specifies the type of failover method the driver uses.

**Valid Values** connect | extended | select

If set to connect, the driver provides failover protection for new connections only.

If set to extended, the driver provides failover protection for new and lost connections, but not any work in progress.

If set to select, the driver provides failover protection for new and lost connections. In addition, it preserves the state of work performed by the last Select statement executed on the Statement object.

**NOTES:**

- The [AlternateServers](#) property specifies one or multiple alternate servers for failover and is required for all failover methods.
- The [FailoverGranularity](#) property determines which action the driver takes if exceptions occur during the failover process.
- The [FailoverPreconnect](#) property specifies whether the driver tries to connect to multiple database servers (primary and alternate) at the same time.

Default connect

Data Type String

See Also [“Configuring Failover” on page 529](#)

## FailoverPreconnect

Description Specifies whether the driver tries to connect to the primary and an alternate server at the same time. This property is ignored if [FailoverMode=connect](#).

Valid Values true | false

If set to true, the driver tries to connect to the primary and an alternate server at the same time. This can be useful if your application is time-sensitive and cannot absorb the wait for the failover connection to succeed.

If set to false, the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection. This value provides the best performance, but your application typically experiences a short wait while the failover connection is attempted.

NOTE: The [AlternateServers](#) property specifies one or multiple alternate servers for failover.

Default	false
Data Type	boolean
See Also	<a href="#">"Configuring Failover" on page 529</a>

## HostNameInCertificate

**Description** 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.

**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.

**Valid Values** *host\_name* | #SERVERNAME#

where *host\_name* is a valid host name.

If *host\_name* is specified, the driver compares the specified host name to the DNSName value of the SubjectAlternativeName in the certificate. If a DNSName value does not exist in the SubjectAlternativeName or if the certificate does not have a SubjectAlternativeName, the driver compares the host name with the Common Name (CN) part of the certificate's Subject name. 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 specified in the connection URL or data source of the

connection to the DNSName value of the SubjectAlternativeName in the certificate. If a DNSName value does not exist in the SubjectAlternativeName or 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.

Default empty string

Data Type String

## ImportStatementPool

Description 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.

Default empty string

Data Type String

See Also [“Importing Statements into a Statement Pool” on page 820](#)

[“Performance Considerations” on page 507](#)

## InitializationString

Description	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	<p><i>string</i></p> <p>where <i>string</i> is one or multiple SQL commands.</p> <p>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.</p>
Example	The following connection URL sets the handling of null values to the Sybase default and allows delimited identifiers:
	<pre>jdbc:datadirect:sybase://server1:5000; InitializationString=(set ANSINULL off; set QUOTED_IDENTIFIER on);DatabaseName=test</pre>
Default	None
Data Type	String

## InInsensitiveResultSetBufferSize

Description	Determines the amount of memory that is used by the driver to cache insensitive result set data.
Valid Values	<p>-1   0   <i>x</i></p> <p>where <i>x</i> is a positive integer.</p> <p>If set to -1, the driver caches insensitive result set data in memory. If the size of the result set exceeds available memory,</p>

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.

Default 2048

Data Type int

See Also [“Performance Considerations” on page 507](#)

## JavaDoubleToString

Description 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

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.

Default false

Data Type boolean

## JDBCBehavior

Description Determines how the driver describes database data types that map to the following JDBC 4.0 data types: NCHAR, NVARCHAR, NLONGVARCHAR, NCLOB, and SQLXML. In addition, it controls whether the PROCEDURE\_NAME column returned by DatabaseMetadata.getProcedures() and DatabaseMetadata.getProcedureColumns() contains procedure name qualifiers.

This property is applicable only when the application is using Java SE 6.

Valid Values 0 | 1

If set to 0, the driver describes the data types as JDBC 4.0 data types when using Java SE 6. Additionally, the PROCEDURE\_NAME column does not contain procedure name qualifiers in the specific\_name column. For example, for the fully qualified procedure name 1.sp\_productadd, the driver would return sp\_productadd instead of sp\_productadd;1.

If set to 1, the driver describes the data types using JDBC 3.0-equivalent data types, regardless of JVM. This allows your application to continue using JDBC 3.0 types in a Java SE 6 environment. Additionally, the PROCEDURE\_NAME column contains procedure name qualifiers. For example, for the fully

qualified procedure name 1.sp\_productadd, the driver would return sp\_productadd;1.

Default 1

Data Type int

## LoadBalancing

Description 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

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 that they are specified).

Default false

Data Type boolean

See Also ["Configuring Failover" on page 529](#)

## LoginTimeout

Description	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 number of seconds.  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.
Default	0
Data Type	int

## LongDataCacheSize

Description	Determines whether the driver caches long data (images, pictures, long text, or binary data) in result sets. To improve performance, you can disable long data caching if your application retrieves columns in the order in which they are defined in the result set.
Valid Values	-1   0   $x$ where $x$ is a positive integer.  If set to -1, the driver does not cache long data in result sets. It is cached on the server. Use this value only if your application returns columns in the order in which they are defined in the result set.  If set to 0, the driver caches long data in result sets in memory. If the size of the result set data exceeds available memory, the driver pages the result set data to disk.

If set to  $x$ , the driver caches long data in result sets in memory and uses this value to set the size (in KB) of the memory buffer for caching result set data. If the size of the result set data exceeds available memory, the driver pages the result set data to disk.

Default 2048

Data Type int

See Also ["Performance Considerations" on page 507](#)

## MaxPooledStatements

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

Valid Values  $0 \mid x$

where  $x$  is a positive integer.

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

If set to  $x$ , the driver enables the DataDirect Statement Pool Monitor and uses the specified value to cache a certain number of prepared statements that are created by an application. For 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. If the value set for this property is greater than the number of prepared statements that are used by the application, all prepared statements created by the application are cached. Because CallableStatement is a sub-class of PreparedStatement, CallableStatements also are cached.

Default 0

Data Type	int
Alias	<a href="#">MaxStatements</a> property
See Also	<a href="#">“Performance Considerations” on page 507</a> <a href="#">Appendix J “Statement Pool Monitor” on page 811</a>

## MaxStatements

Description	An alias for the <a href="#">MaxPooledStatements</a> property.
-------------	--

## PacketSize

Description	Determines the number of bytes for each database protocol packet that are transferred from the database server to the client machine (Sybase refers to this packet as a network packet).  Adjusting the packet size can improve performance. The optimal value depends on the typical size of data that is inserted, updated, or returned by the application and the environment in which it is running. Typically, larger packet sizes work better for large amounts of data. For example, if an application regularly returns character values that are 10,000 characters in length, using a value of 32 (16 KB) typically results in improved performance.  NOTE: If SSL encryption is enabled using the <a href="#">EncryptionMethod</a> property, any value set for the PacketSize property is ignored.
Valid Values	-1   0   $x$  where $x$ is an integer from 1 to 127.  If set to -1, the driver uses the maximum packet size that is used by the database server.  If set to 0, the driver uses the default packet size that is used by the database server.

If set to *x*, the driver uses a packet size that is a multiple of 512 bytes. For example, `PacketSize=8` means to set the packet size to  $8 * 512$  bytes (4096 bytes).

**NOTE:** If your application sends queries that only retrieve small result sets, you may want to use a packet size smaller than the maximum packet size that is configured on the database server. If a result set that contains only one or two rows of data does not completely fill a larger packet, performance will not improve by setting the value to the maximum packet size.

Default 0

Data Type int

See Also [“Performance Considerations” on page 507](#)

## Password

Description A password that is used to connect to your Sybase database. A password is required if security is enabled on your database. Contact your system administrator to obtain your password.

Valid Values *string*

where *string* is a valid password. The password is case-sensitive.

Default None

Data Type String

## PortNumber

Description REQUIRED. The TCP port of the primary database server that is listening for connections to the Sybase database.

This property is supported only for data source connections.

Valid Values *port*

where *port* is the port number.

Default Varies depending on operating system

Data Type int

## PrepareMethod

Description Determines whether stored procedures are created on the server for prepared statements.

Valid Values StoredProc | StoredProclfParam | Direct}

If set to StoredProc, a stored procedure is created when the statement is prepared and is executed when the prepared statement is executed.

If set to StoredProclfParam, a stored procedure is created only if the prepared statement contains one or multiple parameter markers. In this case, it is created when the statement is prepared and is executed when the prepared statement is executed. If the statement does not contain parameter markers, a stored procedure is not created and the statement is executed directly.

If set to Direct, a stored procedure is not created for the prepared statement and the statement is executed directly. A stored procedure may be created if parameter metadata is requested.

Setting this property to StoredProc or StoredProclfParam can improve performance if your application executes prepared statements multiple times because, once created, executing a stored procedure is faster than executing a single SQL statement. If a prepared statement is only executed once or is never executed, performance can decrease because creating a stored procedure incurs more overhead on the server than simply executing a single SQL statement. Setting this property to Direct

should be used if your application does not execute prepared statements multiple times.

Default	StoredProcIfParam
Data Type	String
See Also	<a href="#">"Performance Considerations" on page 507</a>

## ProgramID

Description	The product and version information of the driver on the client to be stored in the database. This value sets the hostprocess value in the sysprocesses table. This value is used for database administration/monitoring purposes.
Valid Values	<code>DDJVVRM</code> where: <ul style="list-style-type: none"><li>■ <code>DD</code> is an identifier for a DataDirect Connect for JDBC driver.</li><li>■ <code>VV</code> identifies a 2-digit version number (with high-order 0 in the case of a 1-digit version).</li><li>■ <code>RR</code> identifies a 2-digit release number (with high-order 0 in the case of a 1-digit release).</li><li>■ <code>M</code> identifies a 1-character modification level (0-9 or A-Z).</li></ul>
Example	<code>DDJ04100</code>
Default	<code>0000016a</code>
Data Type	String
See Also	<a href="#">"Client Information for Connections" on page 521</a>

## QueryTimeout

Description	Sets the default query timeout (in seconds) for all statements created by a connection.
Valid Values	-1   0   $x$ where $x$ is a number of seconds.  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 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.
Default	0
Data Type	int

## ResultSetMetaDataOptions

Description	Determines whether the driver returns table name information in the ResultSet metadata for Select statements.
Valid Values	0   1  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.

Default 0

Data Type int

See Also [“Performance Considerations” on page 507](#)

## SelectMethod

Description A hint to the driver that determines whether the driver requests a database cursor for Select statements. Performance and behavior of the driver are affected by this property, which is defined as a hint because the driver may not always be able to satisfy the requested method.

Valid Values direct | cursor

If set to direct, the database server sends the complete result set in a single response to the driver when responding to a query. A server-side database cursor is not created if the requested result set type is a forward-only result set. Typically, responses are not cached by the driver. Using this method, the driver must process the entire response to a query before another query is submitted. If another query is submitted (using a different statement on the same connection, for example), the driver caches the response to the first query before submitting the second query. Typically, the direct method performs better than the cursor method.

If set to cursor, a server-side cursor is requested. When returning forward-only result sets, the rows are returned from the server in blocks. The `setFetchSize()` method can be used to control the number of rows that are returned for each request when

forward-only result sets are returned. Performance tests show that, when returning forward-only result sets, the value of `Statement.setFetchSize()` significantly impacts performance. There is no simple rule for determining the `setFetchSize()` value that you should use. We recommend that you experiment with different `setFetchSize()` values to determine which value gives the best performance for your application. The cursor method is useful for queries that produce a large amount of data, particularly if multiple open result sets are used.

**Default** direct

**Data Type** String

**See Also** ["Performance Considerations" on page 507](#)

## ServerName

**Description** 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 SybaseServer

**Default** None

**Data Type** String

## ServicePrincipalName

**Description** Specifies the service principal name to be used by the driver for Kerberos authentication. For Sybase, the service principal name is the name of a server that is configured in your Sybase interfaces file.

If you set this property, you also must set the value of the [AuthenticationMethod](#) property to Kerberos. When Kerberos authentication is not used, this property is ignored.

**Valid Values** *string*

where *string* is a valid service principal name. This name is case-sensitive.

The value can include the Kerberos realm name, but it is optional. If you do not specify the Kerberos realm name, the driver uses the default Kerberos realm.

**Example** If the service principal name, including Kerberos realm name, is server/sybase125ase1@XYZ.COM and the default realm is XYZ.COM, values for this property are:

server/sybase125ase1@XYZ.COM

or

server/sybase125ase1

**Default** None

**Data Type** String

**See Also** ["Authentication" on page 512](#)

## SpyAttributes

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

**Valid Values** (*spy\_attribute*[;*spy\_attribute*]...)

where *spy\_attribute* is any valid DataDirect Spy attribute. See [Appendix H "Tracking JDBC Calls with DataDirect Spy™" on page 783](#) for a list of supported attributes.

NOTE: If coding a path on Windows to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example:

```
log=(file)C:\\temp\\spy.log.
```

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

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

**Default** None

**Data Type** String

## TransactionMode

**Description** Controls how the driver delimits the start of a local transaction.

**Valid Values** implicit | explicit

If set to implicit, the driver uses implicit transaction mode. This means that Sybase, not the driver, automatically starts a transaction when a transactionable statement is executed.

Typically, implicit transaction mode is more efficient than explicit transaction mode because the driver does not have to send commands to start a transaction and a transaction is not started until it is needed. When TRUNCATE TABLE statements are used with implicit transaction mode, Sybase may roll back the transaction if an error occurs. If this occurs, use the explicit value for this property.

If set to explicit, the driver uses explicit transaction mode. This means that the driver, not Sybase, starts a new transaction if the previous transaction was committed or rolled back.

**Default** implicit

**Data Type** String

## TrustStore

Description	<p>Specifies the directory of the truststore file to be used when SSL is enabled (<a href="#">EncryptionMethod=SSL</a>) and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.</p> <p>This value overrides the directory of the truststore file that is specified by the <code>javax.net.ssl.trustStore</code> Java system property. If this property is not specified, the truststore directory is specified by the <code>javax.net.ssl.trustStore</code> Java system property.</p> <p>This property is ignored if <a href="#">ValidateServerCertificate=false</a>.</p>
Valid Values	<p><i>string</i></p> <p>where <i>string</i> is the directory of the truststore file.</p>
Default	None
Data Type	String

## TrustStorePassword

Description	<p>Specifies the password that is used to access the truststore file when SSL is enabled (<a href="#">EncryptionMethod=SSL</a>) and server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts.</p> <p>This value overrides the password of the truststore file that is specified by the <code>javax.net.ssl.trustStorePassword</code> Java system property. If this property is not specified, the truststore password is specified by the <code>javax.net.ssl.trustStorePassword</code> Java system property.</p> <p>This property is ignored if <a href="#">ValidateServerCertificate=false</a>.</p>
Valid Values	<p><i>string</i></p> <p>where <i>string</i> is a valid password for the truststore file.</p>

Default	None
Data Type	String

## UseAlternateProductInfo

Description	Determines if the driver will perform additional processing to return more accurate information for the DatabaseMetaData.getDatabaseProductName() and DatabaseMetaData.getDatabaseProductVersion() methods.
Valid Values	true   false
	If set to true, the driver makes an additional query to select the value of @@version and returns only the product name information from the string it receives when getDatabaseProductName() is called. When getDatabaseProductVersion() is called, the entire string is returned. For example:
	Adaptive Server Enterprise/12.5.1/EBF 11428/P/NT (1X86)/OS 4.0/ase1251/1823/32-bit/OPT/Wed Sep 17 11:10:54 2003
	If set to false, the driver returns the information that it receives from the server during the login process. Previous versions of the driver returned this information.
Default	false
Data Type	boolean

## User

Description	The user name that is used to connect to the Sybase database. A user name is required only if security is enabled on your database. Contact your system administrator to get your user name.
-------------	--

Valid Values *string*

where *string* is a valid user name. The user name is case-insensitive.

Default None

Data Type String

## ValidateServerCertificate

Description 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

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 that 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.

NOTE: Truststore information is specified using the [TrustStore](#) and [TrustStorePassword](#) properties or by using Java system properties.

Default	true
Data Type	boolean

---

## Performance Considerations

You can optimize your application's performance if you set the Sybase driver connection properties as described in this section:

**BatchPerformanceWorkaround:** The driver can use a JDBC 3.0-compliant batch mechanism or the native Sybase batch mechanism to execute batch operations. Performance can be improved by using the native Sybase batch environment, especially when performance-expensive network roundtrips are an issue. When using the native mechanism, be aware that if the execution of the batch results in an error, the driver cannot determine which statement in the batch caused the error. In addition, if the batch contained a statement that called a stored procedure or executed a trigger, multiple update counts for each batch statement or parameter set are generated. The JDBC 3.0-compliant mechanism returns individual update counts for each statement or parameter set in the batch as required by the JDBC 3.0 specification. To use the Sybase native batch mechanism, this property should be set to true.

**EnableBulkLoad:** For batch inserts, the driver can use native bulk load protocols instead of the batch mechanism. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations. Set this property to true to allow existing applications with batch inserts to take advantage of bulk load without requiring changes to the code.

**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.

**LongDataCacheSize:** To improve performance when your application returns images, pictures, long text, or binary data, you can disable caching for long data on the client if your application returns long data column values in the order they are defined in the result set. If your application returns long data column values out of order, long data values must be cached.

**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.

See “[Managing Connections](#)” on page 711 for more information about using prepared statement pooling to optimize performance.

**PacketSize:** Typically, it is optimal for the client to use the maximum packet size that the server allows. This reduces the total number of round trips required to return data to the client, thus improving performance. Therefore, performance can be

improved if this property is set to the maximum packet size of the database server.

**PrepareMethod:** If your application executes prepared statements multiple times, this property should be set to StoredProc to improve performance because, once created, executing a stored procedure is faster than executing a single SQL Statement. If your application does not execute prepared statements multiple times, this property should be set to Direct. In this case, performance decreases if a stored procedure is created because a stored procedure incurs more overhead on the server than executing a single SQL statement.

**SelectMethod:** In most cases, using server-side database cursors impacts performance negatively. However, if the following statements are true for your application, the best setting for this property is cursor, which means use server-side database cursors:

- Your application contains queries that return large amounts of data.
- Your application executes a SQL statement before processing or closing a previous large result set and does this multiple times.
- Large result sets returned by your application use forward-only cursors.

---

## Data Types

[Table 8-1](#) lists the data types supported by the Sybase driver and how they are mapped to the JDBC data types.

---

***Table 8-1. Sybase Data Types***

---

Sybase Data Type	JDBC Data Type
BIGINT <sup>1</sup>	BIGINT
BINARY	BINARY
BIT	BIT
CHAR	CHAR
DATE <sup>2</sup>	DATE
DATETIME	TIMESTAMP
DECIMAL	DECIMAL
FLOAT	FLOAT
IMAGE	LONGVARBINARY
INT	INTEGER
MONEY	DECIMAL
NUMERIC	NUMERIC
REAL	REAL
SMALLDATETIME	TIMESTAMP
SMALLINT	SMALLINT
SMALLMONEY	DECIMAL
SYSNAME	VARCHAR
TEXT	LONGVARCHAR
TIME <sup>2</sup>	TIME
TIMESTAMP	VARBINARY
TINYINT	TINYINT

**Table 8-1. Sybase Data Types (cont.)**

Sybase Data Type	JDBC Data Type
UNICHAR <sup>2</sup>	CHAR NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NCHAR (if using Java SE 6) or CHAR (if using another JVM).
UNITEXT <sup>1</sup>	LONGVARCHAR NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: LONGNVARCHAR (if using Java SE 6) or LONGVARCHAR (if using another JVM).
UNIVARCHAR <sup>2</sup>	VARCHAR NOTE: If JDBCBehavior=0, the data type depends on the JVM used by the application: NVARCHAR (if using Java SE 6) or VARCHAR (if using another JVM).
UNSIGNED BIGINT <sup>1</sup>	DECIMAL
UNSIGNED INT <sup>1</sup>	BIGINT
UNSIGNED SMALLINT <sup>1</sup>	INTEGER
VARBINARY	VARBINARY
VARCHAR	VARCHAR

1. Supported only for Sybase 15.  
2. Supported only for Sybase 12.5 and higher.

**NOTE FOR USERS OF SYBASE 12.5 AND HIGHER:** The Sybase driver supports extended new limits (XNL) for character and binary columns—columns with lengths greater than 255. Refer to your Sybase documentation for more information about XNL for character and binary columns.

See [Appendix D “getTypeInfo” on page 631](#) for more information about data types.

---

## Authentication

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See [“Authentication” on page 67](#) for an overview.

The Sybase driver supports the following methods of authentication:

- User ID/password authentication authenticates the user to the database 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 and MIT Kerberos.

The driver’s AuthenticationMethod connection property controls which authentication mechanism the driver uses when establishing connections. See [“Using the AuthenticationMethod Property” on page 513](#) for information about setting the value for this property.

## Using the AuthenticationMethod Property

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

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

When AuthenticationMethod=userIdPassword (the default), the driver uses user ID/password authentication when establishing a connection. The User property provides the user ID. The Password property provides the password. If a user ID is not specified, the driver throws an exception.

## Configuring User ID/Password Authentication

- 1 Set the AuthenticationMethod property to userIdPassword. See ["Using the AuthenticationMethod Property" on page 513](#) for more information about setting a value for this property.
- 2 Set the User property to provide the user ID.
- 3 Set the Password property to provide the password.

# Configuring Kerberos Authentication

This section provides requirements and instructions for configuring Kerberos authentication for the Sybase driver.

## ***Product Requirements***

Verify that your environment meets the requirements listed in [Table 8-2](#) before you configure the driver for Kerberos authentication.

---

***Table 8-2. Kerberos Authentication Requirements for the Sybase Driver***

---

Component	Requirements
Database server	The database server must be administered by the same domain controller that administers the client and must be running Sybase 12.0 or higher. In addition, the Sybase Security and directory services package, ASE_SECDIR, is required.
Kerberos server	<p>The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos KDC.</p> <p>Network authentication must be provided by one of the following methods:</p> <ul style="list-style-type: none"><li>■ Windows Active Directory on one of the following operating systems: Windows Server 2003 or Windows 2000 Server Service Pack 3 or higher</li><li>■ MIT Kerberos 1.4.2 or higher</li></ul>
Client	The client must be administered by the same domain controller that administers the database server. In addition, J2SE 1.4.2 or higher must be installed.

---

## ***Configuring the Driver***

During installation, DataDirect Connect *for JDBC* installs the following files required for Kerberos authentication in the `/lib` subdirectory of your DataDirect Connect *for JDBC* installation directory:

- `krb5.conf` is a Kerberos configuration file containing values for the Kerberos realm and the KDC name for that realm. DataDirect Connect *for JDBC* installs a generic file that you must modify for your environment.
- `JDBCDriverLogin.conf` file is a configuration file that specifies which Java Authentication and Authorization Service (JAAS) login module to use for Kerberos authentication. This file is configured to load automatically unless the `java.security.auth.login.config` system property is set to load another configuration file. You can modify this file, but the driver must be able to find the `JDBC_DRIVER_01` entry in this file or another specified login configuration file to configure the JAAS login module. Refer to your J2SE documentation for information about setting configuration options in this file

### **To configure the driver:**

- 1 Set the `AuthenticationMethod` property to `kerberos`. See [“Using the `AuthenticationMethod` Property” on page 513](#) for more information about setting a value for this property.
- 2 Set the `ServicePrincipalName` property to the case-sensitive service principal name to be used for Kerberos authentication. For Sybase, the service principal name is the name of a server configured in your Sybase interfaces file.

The value of the `ServicePrincipalName` property can include the Kerberos realm name, but it is optional. If you do not specify the realm name, the default realm is used. For example, if the service principal name, including Kerberos

realm name, is server/sybase125ase1@XYZ.COM and the default realm is XYZ.COM, valid values for this property are:

server/sybase125ase1@XYZ.COM

and

server/sybase125ase1

- 3 Modify the krb5.conf file to contain your Kerberos realm name and the KDC name for that Kerberos realm by editing the file with a text editor or by specifying the system properties, java.security.krb5.realm and java.security.krb5.kdc.

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

For example, if your Kerberos realm name is XYZ.COM and your KDC name is kdc1, your krb5.conf file would look like this:

```
[libdefaults]
    default_realm = XYZ.COM

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

If the krb5.conf file does not contain a valid Kerberos realm and KDC name, the following exception is thrown:

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

The krb5.conf file installed by DataDirect Connect for JDBC is configured to load automatically unless the java.security.krb5.conf system property is set to point to another Kerberos configuration file.

- 4 If using Kerberos authentication with a Security Manager on a Java 2 Platform, you must grant security permissions to the application and driver. See "[Permissions for Kerberos Authentication](#)" on page 79 for an example.

## Specifying User Credentials for Kerberos Authentication (Delegation of Credentials)

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

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

```
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import java.sql.*;

// The following code creates a javax.security.auth.Subject instance
// used for authentication. Refer to the Java Authentication
// and Authorization Service documentation for details on using a
// LoginContext to obtain a Subject.

LoginContext lc = null;
Subject subject = null;
```

```
try {

    lc = new LoginContext("JaasSample", new TextCallbackHandler());
    lc.login();
    subject = lc.getSubject();
}

catch (Exception le) {
    ... // display login error
}

// This application passes the javax.security.auth.Subject
// to the driver by executing the driver code as the subject

Connection con =
    (Connection) Subject.doAs(subject, new PrivilegedExceptionAction() {

        public Object run() {

            Connection con = null;
            try {

                Class.forName("com.ddtek.jdbc.sybase.SybaseDriver");
                String url = "jdbc:datadirect:sybase://myServer:5000";
                con = DriverManager.getConnection(url);
            }
            catch (Exception except) {

                ... //log the connection error
                Return null;
            }

            return con;
        }
    });
}

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

```
ResultSet rs = stmt.executeQuery(sql);  
... // do something with the results
```

## Obtaining a Kerberos Ticket Granting Ticket

To use Kerberos authentication, the application user first must obtain a Kerberos Ticket Granting Ticket (TGT) from the Kerberos server. The Kerberos server verifies the identity of the user and controls access to services using the credentials contained in the TGT.

If the application uses Kerberos authentication from a Windows client and the Kerberos authentication is provided by Windows Active Directory, the application user is not required to log onto the Kerberos server and explicitly obtain a TGT. Windows Active Directory automatically obtains a TGT for the user.

The application user must explicitly obtain a TGT in the following cases:

- If the application uses Kerberos authentication from a UNIX or Linux client
- If the application uses Kerberos authentication from a Windows client and Kerberos authentication is provided by MIT Kerberos

To explicitly obtain a TGT, the user must log onto the Kerberos server using the kinit command. For example, the following command requests a TGT from the server with a lifetime of 10 hours, which is renewable for 5 days:

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

where *user* is the application user.

Refer to your Kerberos documentation for more information about using the kinit command and obtaining TGTs for users.

---

## Data Encryption

The Sybase driver supports SSL encryption for the following databases:

- Sybase Adaptive Server Enterprise 15.0
- Sybase Adaptive Server Enterprise 12.0, 12.5, 12.5.1, 12.5.2, 12.5.3, and 12.5.4
- Sybase Adaptive Server 11.5 and 11.9

In addition, the Sybase Security and Directory Services package, ASE\_SECDIR, is required.

SSL secures the integrity of your data by encrypting information and providing authentication. See ["Data Encryption Across the Network" on page 69](#) for an overview.

**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:**

- 1 Set the EncryptionMethod property to SSL.
- 2 Specify the location and password of the truststore file used for SSL server authentication. Either set the TrustStore and TrustStore properties or their corresponding Java system properties (`javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword`, respectively).

- 3 To validate certificates sent by the database server, set the ValidateServerCertificate property to true.
- 4 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.

---

## Client Information for Connections

The Sybase driver allows applications to store and return the following types of client information associated with a particular connection:

- Name of the application
- User ID
- Host name of the client
- Additional accounting information, such as an accounting ID
- Product name and version of the Sybase driver

This information can be used for database administration and monitoring purposes. See [Appendix C “Client Information for Connections” on page 621](#) for details.

---

## SQL Escape Sequences

See [Appendix F “SQL Escape Sequences for JDBC” on page 717](#) for information about the SQL escape sequences supported by the Sybase driver.

---

## Isolation Levels

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

---

## Using Scrollable Cursors

The Sybase driver supports scroll-sensitive result sets only on result sets returned from tables created with an identity column. The Sybase driver also supports scroll-insensitive result sets and updatable result sets.

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

---

## Large Object (LOB) Support

Although Sybase does not define a Blob or Clob data type, the Sybase driver allows you to return and update long data, specifically LONGVARBINARY and LONGVARCHAR 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 Clob objects provides some of the same advantages as retrieving and updating Blobs and Clob objects. For example, using Blobs and Clob objects:

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

To provide these advantages of Blobs and Clob objects, data must be cached. Because data is cached, you will incur a performance penalty, particularly if the data is read once sequentially. This performance penalty can be severe if the size of the long data is larger than available memory.

---

## Batch Inserts and Updates

The Sybase driver provides the following batch mechanisms:

- A JDBC-compliant mechanism that uses code in the driver to execute batch operations. This is the default mechanism used by the Sybase driver.
- A mechanism that uses the Sybase native batch functionality. This mechanism may be faster than the standard mechanism, particularly when performance-expensive network roundtrips are an issue. Be aware that if the execution of the batch results in an error, the driver cannot determine which statement in the batch caused the error. In addition, if the batch contained a statement that called a stored procedure or executed a trigger, multiple update counts for each batch statement or parameter set are generated.

To use the Sybase native batch mechanism, set the `BatchPerformanceWorkaround` property to true.

## Parameter Metadata Support

The Sybase driver supports returning parameter metadata for all types of SQL statements and stored procedure arguments.

---

## ResultSet MetaData Support

If your application requires table name information, the Sybase driver can return table name information in ResultSet metadata for Select statements. By setting the `ResultSetMetaDataOptions` property to 1, the Sybase 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 Sybase 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 Sybase 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 Sybase 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 Sybase driver also can return schema name and catalog name information when the ResultSetMetaData.getSchemaName() and ResultSetMetaData.getCatalogName() methods are called if the driver can determine that information. For example, for the following statement, the Sybase driver returns "test" for the catalog name, "test1" for the schema name, and "foo" for the table name:

```
SELECT * FROM test.test1.foo
```

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

## Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

See <http://www.jcp.org/en/jsr/detail?id=114> for more information about JSR 114.

---

## Auto-Generated Keys Support

The Sybase driver supports retrieving the values of auto-generated keys. An auto-generated key returned by the Sybase driver is the value of an identity 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 that contains parameters:

- When using an Insert statement that contains no parameters, the Sybase 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 that contains parameters, the Sybase 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)`

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.

See “[Retrieving Auto-Generated Keys](#)” on page 709 for information about how auto-generated keys can improve performance.

---

## Null Values

When the Sybase driver establishes a connection, the driver sets the Sybase database option `ansinull` to on. This action ensures that the driver is compliant with the ANSI SQL standard, which makes developing cross-database applications easier.

By default, Sybase does not evaluate null values in SQL equality (=) or inequality (<>) comparisons or aggregate functions in an ANSI SQL-compliant manner. For example, the

ANSI SQL specification defines that `col1=NULL` as shown in the following Select statement always evaluates to false:

```
SELECT * FROM table WHERE col1 = NULL
```

Using the default database setting `ansinull=off`, the same comparison evaluates to true instead of false.

Setting `ansinull` to on changes how the database handles null values and forces the use of `IS NULL` instead of `=NULL`. For example, if the value of `col1` in the following Select statement is null, the comparison evaluates to true:

```
SELECT * FROM table WHERE col1 IS NULL
```

In your application, you can restore the default Sybase behavior for a connection in the following ways:

- Use the `InitializationString` property to specify the SQL command `set ANSINULL off`. For example, the following URL ensures that the handling of null values is restored to the Sybase default for the current connection:

```
jdbc:datadirect:sybase://server1:5000;  
InitializationString=set ANSINULL off;DatabaseName=test
```

- Explicitly execute the following statement after the connection is established:

```
SET ANSINULL OFF
```

---

# Configuring Failover

- 1 Specify the primary and alternate servers:
  - Specify your primary server using a connection URL or data source.
  - Specify one or multiple alternate servers by setting the AlternateServers property.
- See "[Specifying Primary and Alternate Servers](#)" on page 530.
- 2 Choose a failover method by setting the FailoverMode connection property. The default method is connection failover (FailoverMode=connect). See "[Using Failover](#)" on page 58 for an overview of each failover method.
- 3 If FailoverMode=extended or FailoverMode=select, set the FailoverGranularity property to specify how you want the driver to behave if exceptions occur while trying to reestablish a lost connection. The default behavior of the driver is to continue with the failover process and post any exceptions on the statement on which they occur (FailoverGranularity=nonAtomic).
- 4 Optionally, configure the connection retry feature. See "[Specifying Connection Retry](#)" on page 532.
- 5 Optionally, set the FailoverPreconnect property if you want the driver to establish a connection with the primary and an alternate server at the same time. The default behavior is to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection (FailoverPreconnect=false).

## Specifying Primary and Alternate Servers

Connection information for primary and alternate servers can be specified using either one of the following methods:

- Connection URL through the JDBC Driver Manager
- JDBC data source

For example, the following connection URL for the Sybase driver specifies connection information for the primary and alternate servers using a connection URL:

```
jdbc:datadirect:sybase://server1:4100;DatabaseName=TEST;User=test;  
Password=secret;AlternateServers=(server2:4100;DatabaseName=TEST2,  
server3:4100;DatabaseName=TEST3)
```

In this example:

```
...server1:4100;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=(server2:4100;DatabaseName=TEST2,  
server3:4100;DatabaseName=TEST3)
```

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

```
SybaseDataSource mds = new SybaseDataSource();  
mds.setDescription("My SybaseDataSource");  
mds.setServerName("server1");  
mds.setPortNumber(4100);  
mds.setDatabaseName("TEST");  
mds.setUser("test");  
mds.setPassword("secret");  
AlternateServers=(server2:4100;DatabaseName=TEST2,  
server3:4100;DatabaseName=TEST3)
```

In this example, connection information for the primary server is specified using the `ServerName`, `PortNumber`, and `DatabaseName` properties. Connection information for 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. The port number is optional for each alternate server entry. If unspecified, the port number specified for the primary server is used.

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

```
jdbc:datadirect:sybase://server1:4100;DatabaseName=TEST;User=test;
Password=secret;AlternateServers=(server2:4100;DatabaseName=TEST2,
server3:4100)
```

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:sybase://server1:4100;DatabaseName=TEST;User=test;  
Password=secret;AlternateServers=(server2:4100,server3:4100)
```

## Specifying Connection Retry

Connection retry allows the Sybase 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:sybase://server1:4100;DatabaseName=TEST;User=test;  
Password=secret;AlternateServers=(server2:4100;DatabaseName=TEST2,  
server3:4100;DatabaseName=TEST3);ConnectionRetryCount=2;  
ConnectionRetryDelay=5
```

In this example, if a successful connection is not established on the Sybase 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.

## Failover Properties

[Table 8-3](#) summarizes the connection properties that control how failover works with the Sybase driver. See ["Connection Properties" on page 473](#) for details about configuring each property.

**Table 8-3. Failover Properties for the Sybase Driver**

Property	Characteristic
AlternateServers	One or multiple alternate database servers. An IP address or server name identifying each server is required. Port number and the DatabaseName connection property are optional. If the port number is unspecified, the port number specified for the primary server is used.
ConnectionRetryCount	Number of times the driver retries the primary database server, and if specified, alternate servers until a successful connection is established. The default is 5.
ConnectionRetryDelay	Wait interval, in seconds, between connection retry attempts when the ConnectionRetryCount property is set to a positive integer. The default is 1.
DatabaseName	Name of the database to which you want to connect.
FailoverGranularity	Determines whether the driver fails the entire failover process or continues with the process if exceptions occur while trying to reestablish a lost connection. The default is nonAtomic (the driver continues with the failover process and posts any exceptions on the statement on which they occur).
FailoverMode	The failover method you want the driver to use. The default is connect (connection failover is used).
FailoverPreconnect	Specifies whether the driver tries to connect to the primary and an alternate server at the same time. The default is false (the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection).

---

**Table 8-3. Failover Properties for the Sybase Driver (cont.)**

---

Property	Characteristic
LoadBalancing	Sets whether the driver will use client load balancing in its attempts to connect to the database servers (primary and alternate). If client load balancing is enabled, the driver uses a random pattern instead of a sequential pattern in its attempts to connect. The default is false (client load balancing is disabled).
PortNumber	Port listening for connections on the primary database server. This property is supported only for data source connections. The default port number varies, depending on operating system.
ServerName	IP address or server name of primary database server. This property is supported only for data source connections.

---

See “[Using Failover](#)” on page 58 and “[Using Client Load Balancing](#)” on page 65 for overviews of failover and client load balancing.

---

## Bulk Load

The driver supports DataDirect Bulk Load, a feature that allows your application to send large numbers of rows of data to the database in a continuous stream instead of in numerous smaller database protocol packets. Similar to batch operations, performance improves because far fewer network round trips are required. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations. See “[Using DataDirect Bulk Load](#)” on page 829 for more information.

# A JDBC Support

This appendix provides information about JDBC compatibility and developing JDBC applications for DataDirect Connect *for* JDBC environments.

---

## JDBC/JVM Compatibility

[Table A-1](#) shows the JDBC and JVM versions supported by the DataDirect Connect *for* JDBC drivers.

---

**Table A-1. JDBC/JVM Compatibility**

---

<b>JDBC Version</b>	<b>JVM Version</b>
2.0	J2SE 1.4, J2SE 5, Java SE 6
3.0	J2SE 1.4, J2SE 5, Java SE 6
4.0	J2SE 1.4, J2SE 5, Java SE 6

---

# Supported Functionality

The following tables list functionality supported for each JDBC object.

## Array Object

Array Object	Version		
Methods	Introduced	Supported	Comments
(all)	2.0 Core	No	Array objects are not exposed or used as input.

## Blob Object

Blob Object	Version		
Methods	Introduced	Supported	Comments
void free ()	4.0	Yes	
InputStream getBinaryStream ()	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.

Blob Object (cont.)	Version		
Methods	Introduced	Supported	Comments
byte[] getBytes (long, int)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
long length ()	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
long position (Blob, long)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The Informix driver requires that the pattern parameter (which specifies the Blob object designating the BLOB value for which to search) be less than or equal to a maximum value of 4096 bytes.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.

<b>Blob Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
long position (byte[], long)	2.0 Core	Yes	<p>The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.</p> <p>The Informix driver requires that the pattern parameter (which specifies the byte array for which to search) be less than or equal to a maximum value of 4096 bytes.</p> <p>The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.</p>
OutputStream setBinaryStream (long)	3.0	Yes	<p>The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.</p> <p>The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.</p>
int setBytes (long, byte[])	3.0	Yes	<p>The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.</p> <p>The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.</p>

<b>Blob Object (cont.)</b>		<b>Version</b>		
<b>Methods</b>		<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
<code>int setBytes (long, byte[], int, int)</code>		3.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
<code>void truncate (long)</code>		3.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.

## CallableStatement Object

<b>CallableStatement Object</b>		<b>Version</b>		
<b>Methods</b>		<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
<code>Array getArray (int)</code>		2.0 Core	No	Throws "unsupported method" exception.
<code>Array getArray (String)</code>		3.0	No	Throws "unsupported method" exception.
<code>Reader getCharacterStream (int)</code>		4.0	Yes	
<code>Reader getCharacterStream (String)</code>		4.0	Yes	
<code>BigDecimal getBigDecimal (int)</code>	2.0 Core		Yes	

<b>CallableStatement Object (cont.)</b>	<b>Version</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
BigDecimal getBigDecimal (int, int)	1.0		Yes	
BigDecimal getBigDecimal (String)	3.0		Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Blob getBlob (int)	2.0 Core		Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
Blob getBlob (String)	3.0		Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
boolean getBoolean (int)	1.0		Yes	
boolean getBoolean (String)	3.0		Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
byte getByte (int)	1.0		Yes	
byte getByte (String)	3.0		Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
byte [] getBytes (int)	1.0		Yes	
byte [] getBytes (String)	3.0		Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.

CallableStatement Object (cont.)	Version Introduced	Supported	Comments
Clob getClob (int)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
Clob getClob (String)	3.0	Yes	Supported for the SQL Server driver only using with data types that map to the JDBC LONGVARCHAR data type. All other drivers throw "unsupported method" exception.
Date getDate (int)	1.0	Yes	
Date getDate (int, Calendar)	2.0 Core	Yes	
Date getDate (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Date getDate (String, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
double getDouble (int)	1.0	Yes	
double getDouble (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
float getFloat (int)	1.0	Yes	
float getFloat (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
int getInt (int)	1.0	Yes	

<b>CallableStatement Object (cont.)</b>	<b>Version Introduced</b>	<b>Supported</b>	<b>Comments</b>
int getInt (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
long getLong (int)	1.0	Yes	
long getLong (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Reader getNCharacterStream (int)	4.0	Yes	
Reader getNCharacterStream (String)	4.0	Yes	
NClob getNClob (int)	4.0	Yes	
NClob getNClob (String)	4.0	Yes	
String getNString (int)	4.0	Yes	
String getNString (String)	4.0	Yes	
Object getObject (int)	1.0	Yes	
Object getObject (int, Map)	2.0 Core	Yes	Map ignored.
Object getObject (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Object getObject (String, Map)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception. Map ignored.
Ref getRef (int)	2.0 Core	No	Throws "unsupported method" exception.

<b>CallableStatement Object (cont.)</b>	<b>Version</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
Ref getRef (String)	3.0	No		Throws "unsupported method" exception.
short getShort (int)	1.0	Yes		
short getShort (String)	3.0	Yes		Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
SQLXML getSQLXML (int)	4.0	Yes		
SQLXML getSQLXML (String)	4.0	Yes		
String getString (int)	1.0	Yes		
String getString (String)	3.0	Yes		Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Time getTime (int)	1.0	Yes		
Time getTime (int, Calendar)	2.0 Core	Yes		
Time getTime (String)	3.0	Yes		Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Time getTime (String, Calendar)	3.0	Yes		Supported for SQL Server driver only. All other drivers throw "unsupported method" exception.
Timestamp getTimestamp (int)	1.0	Yes		
Timestamp getTimestamp (int, Calendar)	2.0 Core	Yes		
Timestamp getTimestamp (String)	3.0	Yes		Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.

<b>CallableStatement Object (cont.)</b>	<b>Version Introduced</b>	<b>Supported</b>	<b>Comments</b>
Timestamp getTimestamp (String, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
URL getURL (int)	3.0	No	Throws "unsupported method" exception.
URL getURL (String)	3.0	No	Throws "unsupported method" exception.
boolean isWrapperFor (Class<?> iface)	4.0	Yes	
void registerOutParameter (int, int)	1.0	Yes	
void registerOutParameter (int, int, int)	1.0	Yes	
void registerOutParameter (int, int, String)	2.0 Core	Yes	String/typename ignored.
void registerOutParameter (String, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void registerOutParameter (String, int, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void registerOutParameter (String, int, String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception. String/typename ignored.
void setArray (int, Array)	2.0 Core	No	Throws "unsupported method" exception.

<b>CallableStatement Object (cont.)</b>	<b>Version Introduced</b>	<b>Supported</b>	<b>Comments</b>
void setAsciiStream (String, InputStream)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setAsciiStream (String, InputStream, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setAsciiStream (String, InputStream, long)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setBigDecimal (String, BigDecimal)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setBinaryStream (String, InputStream)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setBinaryStream (String, InputStream, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setBinaryStream (String, InputStream, long)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setBlob (String, Blob)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.

<b>CallableStatement Object (cont.)</b>	<b>Version Introduced</b>	<b>Supported</b>	<b>Comments</b>
void setBlob (String, InputStream)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setBlob (String, InputStream, long)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setBoolean (String, boolean)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setByte (String, byte)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setBytes (String, byte [])	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setCharacterStream (String, Reader, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setCharacterStream (String, InputStream, long)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setClob (String, Clob)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.

<b>CallableStatement Object (cont.)</b>	<b>Version</b>		
	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void setClob (String, Reader)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setClob (String, Reader, long)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setDate (String, Date)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setDate (String, Date, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setDouble (String, double)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setFloat (String, float)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setInt (String, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setLong (String, long)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setNCharacterStream (String, Reader, long)	4.0	Yes	

<b>CallableStatement Object (cont.)</b>	<b>Version</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void setNClob (String, NClob)	4.0	Yes		
void setNClob (String, Reader)	4.0	Yes		
void setNClob (String, Reader, long)	4.0	Yes		
void setNString (String, String)	4.0	Yes		
void setNull (String, int)	3.0	Yes		Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setNull (String, int, String)	3.0	Yes		Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setObject (String, Object)	3.0	Yes		Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setObject (String, Object, int)	3.0	Yes		Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setObject (String, Object, int, int)	3.0	Yes		Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setShort (String, short)	3.0	Yes		Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setSQLXML (String, SQLXML)	4.0	Yes		

<b>CallableStatement Object (cont.)</b>	<b>Version</b>		
	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void setString (String, String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setTime (String, Time)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setTime (String, Time, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setTimestamp (String, Timestamp)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setTimestamp (String, Timestamp, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
<T> T unwrap(Class<T> iface)	4.0	Yes	
void setURL (String, URL)	3.0	No	Throws "unsupported method" exception.
boolean wasNull ()	1.0	Yes	

## Clob Object

<b>Clob Object Methods</b>	<b>Version</b>		
	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void free ()	4.0	Yes	

<b>Clob Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
InputStream getAsciiStream ()	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
Reader getCharacterStream ()	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
Reader getCharacterStream (long, long)	4.0	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
String getSubString (long, int)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
long length ()	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
long position (Clob, long)	2.0 Core	Yes	The Informix driver requires that the searchStr parameter be less than or equal to a maximum value of 4096 bytes. The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.

Clob Object (cont.)	Version			
Methods	Introduced	Supported	Comments	
long position (String, long)	2.0 Core	Yes	The Informix driver requires that the searchStr parameter be less than or equal to a maximum value of 4096 bytes. The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.	
OutputStream setAsciiStream (long)	3.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.	
Writer setCharacterStream (long)	3.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.	
int setString (long, String)	3.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.	
int setString (long, String, int, int)	3.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.	
void truncate (long)	3.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.	

## Connection Object

Connection Object Methods	Version Introduced	Supported	Comments
void clearWarnings ()	1.0	Yes	
void close ()	1.0	Yes	When a connection is closed while a transaction is still active, that transaction is rolled back.
void commit ()	1.0	Yes	
Blob createBlob ()	4.0	Yes	
Clob createClob ()	4.0	Yes	
NClob createNClob ()	4.0	Yes	
SQLXML createSQLXML ()	4.0	Yes	
Statement createStatement ()	1.0	Yes	
Statement createStatement (int, int)	2.0 Core	Yes	ResultSet.TYPE_SCROLL_SENSITIVE downgraded to TYPE_SCROLL_INSENSITIVE for the DB2 driver.
Statement createStatement (int, int, int)	3.0	No	Throws "unsupported method" exception.
boolean getAutoCommit ()	1.0	Yes	
String getCatalog ()	1.0	Yes	Supported for all drivers except Oracle, which does not have the concept of a catalog. The Oracle driver returns an empty string.
String getClientInfo ()	4.0	Yes	See <a href="#">Appendix C "Client Information for Connections" on page 621</a> for more information.

<b>Connection Object (cont.)</b>		<b>Version</b>		
<b>Methods</b>		<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
<code>String getClientInfo (String)</code>	4.0	Yes		See <a href="#">Appendix C "Client Information for Connections" on page 621</a> for more information.
<code>int getHoldability ()</code>	3.0	Yes		
<code>DatabaseMetaData getMetaData ()</code>	1.0	Yes		
<code>int getTransactionIsolation ()</code>	1.0	Yes		
<code>Map getTypeMap ()</code>	2.0 Core	Yes		Always returns empty <code>java.util.HashMap</code> .
<code>SQLWarning getWarnings ()</code>	1.0	Yes		
<code>boolean isClosed ()</code>	1.0	Yes		
<code>boolean isReadOnly ()</code>	1.0	Yes		
<code>boolean isValid ()</code>	4.0	Yes		
<code>boolean isWrapperFor (Class&lt;?&gt; iface)</code>	4.0	Yes		
<code>String nativeSQL (String)</code>	1.0	Yes		Always returns same String as passed in.
<code>CallableStatement prepareCall (String)</code>	1.0	Yes		
<code>CallableStatement prepareCall (String, int, int)</code>	2.0 Core	Yes		ResultSet.TYPE_SCROLL_SENSITIVE downgraded to TYPE_SCROLL_INSENSITIVE for the DB2 driver.
<code>CallableStatement prepareCall (String, int, int, int)</code>	3.0	No		Throws "unsupported method" exception.
<code>PreparedStatement prepareStatement (String)</code>	1.0	Yes		
<code>PreparedStatement prepareStatement (String, int)</code>	3.0	Yes		

<b>Connection Object (cont.) Methods</b>	<b>Version Introduced</b>	<b>Supported</b>	<b>Comments</b>
PreparedStatement prepareStatement (String, int, int)	2.0 Core	Yes	ResultSet.TYPE_SCROLL_SENSITIVE downgraded to TYPE_SCROLL_INSENSITIVE for the DB2 driver.
PreparedStatement prepareStatement (String, int, int, int)	3.0	No	Throws "unsupported method" exception.
PreparedStatement prepareStatement (String, int[])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws "unsupported method" exception.
PreparedStatement prepareStatement (String, String [])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws "unsupported method" exception.
void releaseSavepoint (Savepoint)	3.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
void rollback ()	1.0	Yes	
void rollback (Savepoint)	3.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
void setAutoCommit (boolean)	1.0	Yes	
void setCatalog (String)	1.0	Yes	Supported for all drivers except Oracle, which does not have the concept of a catalog. The Oracle driver ignores any value set by the catalog parameter.

<b>Connection Object (cont.)</b>		<b>Version</b>		
<b>Methods</b>		<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
<code>String setClientInfo (Properties)</code>	4.0	Yes		See <a href="#">Appendix C “Client Information for Connections” on page 621</a> for more information.
<code>String setClientInfo (String, String)</code>	4.0	Yes		See <a href="#">Appendix C “Client Information for Connections” on page 621</a> for more information.
<code>void setHoldability (int)</code>	3.0	Yes		Holdability parameter value is ignored.
<code>void setReadOnly (boolean)</code>	1.0	Yes		
<code>Savepoint setSavepoint ()</code>	3.0	Yes		The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  In addition, the DB2 driver only supports multiple nested savepoints for DB2 v8.2 and higher for Linux/UNIX/Windows.
<code>Savepoint setSavepoint (String)</code>	3.0	Yes		The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  In addition, the DB2 driver only supports multiple nested savepoints for DB2 v8.2 and higher for Linux/UNIX/Windows.
<code>void setTransactionIsolation (int)</code>	1.0	Yes		
<code>void setTypeMap (Map)</code>	2.0 Core	Yes		Ignored.

<b>Connection Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
<T> T unwrap(Class<T> iface)	4.0	Yes	

## ConnectionEventListener Object

<b>ConnectionEventListener</b>	<b>Version</b>		
<b>Object</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void connectionClosed (event)	3.0	Yes	
void connectionErrorOccurred (event)	3.0	Yes	

## ConnectionPoolDataSource Object

<b>ConnectionPoolDataSource</b>	<b>Version</b>		
<b>Object</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
int getLoginTimeout ()	2.0	Optional	Yes
PrintWriter getLogWriter ()	2.0	Optional	Yes
PooledConnection getPooledConnection ()	2.0	Optional	Yes
PooledConnection getPooledConnection (String, String)	2.0	Optional	Yes
void setLoginTimeout (int)	2.0	Optional	Yes
void setLogWriter (PrintWriter)	2.0	Optional	Yes

## DatabaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
boolean autoCommitFailureClosesAllResultSets ()	4.0	Yes	
boolean allProceduresAreCallable ()	1.0	Yes	
boolean allTablesAreSelectable ()	1.0	Yes	
boolean dataDefinitionCausesTransaction Commit ()	1.0	Yes	
boolean dataDefinitionIgnoredInTransactions ()	1.0	Yes	
boolean deletesAreDetected (int)	2.0 Core	Yes	
boolean doesMaxRowSizeIncludeBlobs ()	1.0	Yes	Not supported by the SQL Server and Sybase drivers.
getAttributes (String, String, String, String)	3.0	Yes	Empty result set is returned.
ResultSet getBestRowIdentifier (String, String, String, int, boolean)	1.0	Yes	
ResultSet getCatalogs ()	1.0	Yes	
String getCatalogSeparator ()	1.0	Yes	
String getCatalogTerm ()	1.0	Yes	
String getClientInfoProperties ()	4.0	Yes	See <a href="#">Appendix C "Client Information for Connections"</a> on page 621 for more information.
ResultSet getColumnPrivileges (String, String, String, String)	1.0	Yes	
ResultSet getColumns (String, String, String, String)	1.0	Yes	

<b>DatabaseMetaData Object (cont.)</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
Connection getConnection ()	2.0 Core	Yes		
ResultSet getCrossReference (String, String, String, String, String, String)	1.0	Yes		
ResultSet getFunctions ()	4.0	Yes		
ResultSet getFunctionColumns ()	4.0	Yes		
int getDatabaseMajorVersion ()	3.0	Yes		
int getDatabaseMinorVersion ()	3.0	Yes		
String getDatabaseProductName ()	1.0	Yes		
String getDatabaseProductVersion ()	1.0	Yes		
int getDefaultTransactionIsolation ()	1.0	Yes		
int getDriverMajorVersion ()	1.0	Yes		
int getDriverMinorVersion ()	1.0	Yes		
String getDriverName ()	1.0	Yes		
String getDriverVersion ()	1.0	Yes		
ResultSet getExportedKeys (String, String, String)	1.0	Yes		
String getExtraNameCharacters ()	1.0	Yes		
String getIdentifierQuoteString ()	1.0	Yes		
ResultSet getImportedKeys (String, String, String)	1.0	Yes		
ResultSet getIndexInfo (String, String, String, boolean, boolean)	1.0	Yes		
int getJDBCMajorVersion ()	3.0	Yes		
int getJDBCMinorVersion ()	3.0	Yes		
int getMaxBinaryLiteralLength ()	1.0	Yes		
int getMaxCatalogNameLength ()	1.0	Yes		

<b>DatabaseMetaData Object (cont.)</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
int getMaxCharLiteralLength ()	1.0	Yes		
int getMaxColumnNameLength ()	1.0	Yes		
int getMaxColumnsInGroupBy ()	1.0	Yes		
int getMaxColumnsInIndex ()	1.0	Yes		
int getMaxColumnsInOrderBy ()	1.0	Yes		
int getMaxColumnsInSelect ()	1.0	Yes		
int getMaxColumnsInTable ()	1.0	Yes		
int getMaxConnections ()	1.0	Yes		
int getMaxCursorNameLength ()	1.0	Yes		
int getMaxIndexLength ()	1.0	Yes		
int getMaxProcedureNameLength ()	1.0	Yes		
int getMaxRowSize ()	1.0	Yes		
int getMaxSchemaNameLength ()	1.0	Yes		
int getMaxStatementLength ()	1.0	Yes		
int getMaxStatements ()	1.0	Yes		
int getMaxTableNameLength ()	1.0	Yes		
int getMaxTablesInSelect ()	1.0	Yes		
int getMaxUserNameLength ()	1.0	Yes		
String getNumericFunctions ()	1.0	Yes		
ResultSet getPrimaryKeys (String, String, String)	1.0	Yes		
ResultSet getProcedureColumns (String, String, String, String)	1.0	Yes		
ResultSet getProcedures (String, String, String)	1.0	Yes		

<b>DatabaseMetaData Object (cont.)</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
String getProcedureTerm ()	1.0	Yes		
int getResultSetHoldability ()	3.0	Yes		
ResultSet getSchemas ()	1.0	Yes		
ResultSet getSchemas (catalog, pattern)	4.0	Yes		
String getSchemaTerm ()	1.0	Yes		
String getSearchStringEscape ()	1.0	Yes		
String getSQLKeywords ()	1.0	Yes		
int getSQLStateType ()	3.0	Yes		
String getStringFunctions ()	1.0	Yes		
ResultSet getSuperTables (String, String, String)	3.0	Yes	Empty result set is returned.	
ResultSet getSuperTypes (String, String, String)	3.0	Yes	Empty result is returned.	
String getSystemFunctions ()	1.0	Yes		
ResultSet getTablePrivileges (String, String, String)	1.0	Yes		
ResultSet getTables (String, String, String, String [])	1.0	Yes		
ResultSet getTableTypes ()	1.0	Yes		
String getTimeDateFunctions ()	1.0	Yes		
ResultSet getTypeInfo ()	1.0	Yes		
ResultSet getUDTs (String, String, String, int [])	2.0 Core	No	Always returns empty ResultSet.	
String getURL ()	1.0	Yes		
String getUserName ()	1.0	Yes		
ResultSet getVersionColumns (String, String, String)	1.0	Yes		

<b>DatabaseMetaData Object (cont.)</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
boolean insertsAreDetected (int)	2.0 Core	Yes		
boolean isCatalogAtStart ()	1.0	Yes		
boolean isReadOnly ()	1.0	Yes		
boolean isWrapperFor (Class<?> iface)	4.0	Yes		
boolean locatorsUpdateCopy ()	3.0	Yes		
boolean nullPlusNonNullIsNull ()	1.0	Yes		
boolean nullsAreSortedAtEnd ()	1.0	Yes		
boolean nullsAreSortedAtStart ()	1.0	Yes		
boolean nullsAreSortedHigh ()	1.0	Yes		
boolean nullsAreSortedLow ()	1.0	Yes		
boolean othersDeletesAreVisible (int)	2.0 Core	Yes		
boolean othersInsertsAreVisible (int)	2.0 Core	Yes		
boolean othersUpdatesAreVisible (int)	2.0 Core	Yes		
boolean ownDeletesAreVisible (int)	2.0 Core	Yes		
boolean ownInsertsAreVisible (int)	2.0 Core	Yes		
boolean ownUpdatesAreVisible (int)	2.0 Core	Yes		
boolean storesLowerCaselIdentifiers ()	1.0	Yes		
boolean storesLowerCaseQuoted Identifiers ()	1.0	Yes		
boolean storesMixedCaselIdentifiers ()	1.0	Yes		
boolean storesMixedCaseQuoted Identifiers ()	1.0	Yes		
boolean storesUpperCaselIdentifiers ()	1.0	Yes		
boolean storesUpperCaseQuoted Identifiers ()	1.0	Yes		

<b>DatabaseMetaData Object (cont.)</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
boolean supportsAlterTableWith AddColumn ()	1.0	Yes		
boolean supportsAlterTableWith DropColumn ()	1.0	Yes		
boolean supportsANSI92EntryLevelSQL ()	1.0	Yes		
boolean supportsANSI92FullSQL ()	1.0	Yes		
boolean supportsANSI92Intermediate SQL ()	1.0	Yes		
boolean supportsBatchUpdates ()	2.0 Core	Yes		
boolean supportsCatalogsInData Manipulation ()	1.0	Yes		
boolean supportsCatalogsInIndex Definitions ()	1.0	Yes		
boolean supportsCatalogsInPrivilege Definitions ()	1.0	Yes		
boolean supportsCatalogsInProcedure Calls ()	1.0	Yes		
boolean supportsCatalogsInTable Definitions ()	1.0	Yes		
boolean supportsColumnAliasing ()	1.0	Yes		
boolean supportsConvert ()	1.0	Yes		
boolean supportsConvert (int, int)	1.0	Yes		
boolean supportsCoreSQLGrammar ()	1.0	Yes		
boolean supportsCorrelatedSubqueries ()	1.0	Yes		
boolean supportsDataDefinitionAndData ManipulationTransactions ()	1.0	Yes		
boolean supportsDataManipulation TransactionsOnly ()	1.0	Yes		

<b>DatabaseMetaData Object (cont.)</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
boolean supportsDifferentTableCorrelationNames ()	1.0	Yes		
boolean supportsExpressionsIn OrderBy ()	1.0	Yes		
boolean supportsExtendedSQLGrammar ()	1.0	Yes		
boolean supportsFullOuterJoins ()	1.0	Yes		
boolean supportsGetGeneratedKeys ()	3.0	Yes		
boolean supportsGroupBy ()	1.0	Yes		
boolean supportsGroupByBeyondSelect ()	1.0	Yes		
boolean supportsGroupByUnrelated ()	1.0	Yes		
boolean supportsIntegrityEnhancementFacility ()	1.0	Yes		
boolean supportsLikeEscapeClause ()	1.0	Yes		
boolean supportsLimitedOuterJoins ()	1.0	Yes		
boolean supportsMinimumSQLGrammar ()	1.0	Yes		
boolean supportsMixedCaseIdentifiers ()	1.0	Yes		
boolean supportsMixedCaseQuotedIdentifiers ()	1.0	Yes		
boolean supportsMultipleOpenResults ()	3.0	Yes		
boolean supportsMultipleResultSets ()	1.0	Yes		
boolean supportsMultipleTransactions ()	1.0	Yes		
boolean supportsNamedParameters ()	3.0	Yes		
boolean supportsNonNullableColumns ()	1.0	Yes		

<b>DatabaseMetaData Object (cont.)</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
boolean supportsOpenCursorsAcross Commit ()	1.0	Yes		
boolean supportsOpenCursorsAcross Rollback ()	1.0	Yes		
boolean supportsOpenStatementsAcross Commit ()	1.0	Yes		
boolean supportsOpenStatementsAcross Rollback ()	1.0	Yes		
boolean supportsOrderByUnrelated ()	1.0	Yes		
boolean supportsOuterJoins ()	1.0	Yes		
boolean supportsPositionedDelete ()	1.0	Yes		
boolean supportsPositionedUpdate ()	1.0	Yes		
boolean supportsResultSetConcurrency (int, int)	2.0 Core	Yes		
boolean supportsResultSetHoldability (int)	3.0	Yes		
boolean supportsResultSetType (int)	2.0 Core	Yes		
boolean supportsSavePoints ()	3.0	Yes		
boolean supportsSchemasInData Manipulation ()	1.0	Yes		
boolean supportsSchemasInIndex Definitions ()	1.0	Yes		
boolean supportsSchemasIn PrivilegeDefinitions ()	1.0	Yes		
boolean supportsSchemasInProcedure Calls ()	1.0	Yes		
boolean supportsSchemasInTable Definitions ()	1.0	Yes		
boolean supportsSelectForUpdate ()	1.0	Yes		

<b>DatabaseMetaData Object (cont.)</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
boolean supportsStoredFunctionsUsingCallSyntax ()	4.0	Yes		
boolean supportsStoredProcedures ()	1.0	Yes		
boolean supportsSubqueriesIn Comparisons ()	1.0	Yes		
boolean supportsSubqueriesInExists ()	1.0	Yes		
boolean supportsSubqueriesInIns ()	1.0	Yes		
boolean supportsSubqueriesIn Quantifieds ()	1.0	Yes		
boolean supportsTableCorrelationNames ()	1.0	Yes		
boolean supportsTransactionIsolationLevel (int)	1.0	Yes		
boolean supportsTransactions ()	1.0	Yes		
boolean supportsUnion ()	1.0	Yes		
boolean supportsUnionAll ()	1.0	Yes		
<T> T unwrap(Class<T> iface)	4.0	Yes		
boolean updatesAreDetected (int)	2.0 Core	Yes		
boolean usesLocalFilePerTable ()	1.0	Yes		
boolean usesLocalFiles ()	1.0	Yes		

## DataSource Object

<b>DataSource Object *</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
Connection getConnection ()	2.0	Optional	Yes	
Connection getConnection (String, String)	2.0	Optional	Yes	
int getLoginTimeout ()	2.0	Optional	Yes	
PrintWriter getLogWriter ()	2.0	Optional	Yes	
boolean isWrapperFor (Class<?> iface)	4.0		Yes	
void setLoginTimeout (int)	2.0	Optional	Yes	
void setLogWriter (PrintWriter)	2.0	Optional	Yes	Enables DataDirect Spy, which traces JDBC information into the specified PrintWriter.
<T> T unwrap(Class<T> iface)	4.0		Yes	
* The DataSource object implements the javax.naming.Referenceable and java.io.Serializable interfaces.				

## Driver Object

Driver Object Methods	Version Introduced	Supported	Comments
boolean acceptsURL (String)	1.0	Yes	
Connection connect (String, Properties)	1.0	Yes	
int getMajorVersion ()	1.0	Yes	
int getMinorVersion ()	1.0	Yes	
DriverPropertyInfo [] getPropertyInfo (String, Properties)	1.0	Yes	

## ParameterMetaData Object

ParameterMetaData Object Methods	Version Introduced	Supported	Comments
String getParameterClassName (int)	3.0	Yes	The DB2 driver supports parameter metadata for stored procedures for DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
int getParameterCount ()	3.0	Yes	
int getParameterMode (int)	3.0	Yes	The DB2 driver supports parameter metadata for stored procedures for DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.

<b>ParameterMetaData Object (cont.)</b>	<b>Version</b>			
	<b>Introduced</b>		<b>Supported</b>	
int getParameterType (int)	3.0	Yes		The DB2 driver supports parameter metadata for stored procedures for DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
String getParameterTypeName (int)	3.0	Yes		The DB2 driver supports parameter metadata for stored procedures for DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
int getPrecision (int)	3.0	Yes		The DB2 driver supports parameter metadata for stored procedures for DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
int getScale (int)	3.0	Yes		The DB2 driver supports parameter metadata for stored procedures for DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
int isNullable (int)	3.0	Yes		The DB2 driver supports parameter metadata for stored procedures for DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.

ParameterMetaData Object (cont.)	Version Introduced	Supported	Comments
boolean isSigned (int)	3.0	Yes	The DB2 driver supports parameter metadata for stored procedures for DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
boolean isWrapperFor (Class<?> iface)	4.0	Yes	
boolean jdbcCompliant ()	1.0	Yes	
<T> T unwrap(Class<T> iface)	4.0	Yes	

## PooledConnection Object

PooledConnection Object Methods	Version Introduced	Supported	Comments
void addConnectionEventListener (ConnectionEventListener)	2.0 Optional	Yes	
void addStatementEventListener (listener)	4.0	Yes	
void close()	2.0 Optional	Yes	

PooledConnection Object (cont.)	Version Introduced	Supported	Comments
Connection getConnection()	2.0 Optional	Yes	A pooled connection object can have only one Connection object open (the one most recently created). The purpose of allowing the server (PoolManager implementation) to invoke this a second time is to give an application server a way to take a connection away from an application and give it to another user (a rare occurrence). The drivers do not support the "reclaiming" of connections and will throw an exception.
void removeConnectionEvent Listener (ConnectionEventListener)	2.0 Optional	Yes	
void removeStatementEventListener (listener)	4.0	Yes	

## PreparedStatement Object

PreparedStatement Object Methods	Version Introduced	Supported	Comments
void addBatch ()	2.0 Core	Yes	
void clearParameters ()	1.0	Yes	
boolean execute ()	1.0	Yes	
ResultSet executeQuery ()	1.0	Yes	
int executeUpdate ()	1.0	Yes	

<b>PreparedStatement Object (cont.)</b>	<b>Version Introduced</b>	<b>Supported</b>	<b>Comments</b>
ResultSetMetaData getMetaData ()	2.0 Core	Yes	
ParameterMetaData getParameterMetaData ()	3.0	Yes	
boolean isWrapperFor (Class<?> iface)	4.0	Yes	
void setArray (int, Array)	2.0 Core	No	Throws "unsupported method" exception.
void setAsciiStream (int, InputStream)	4.0	Yes	
void setAsciiStream (int, InputStream, int)	1.0	Yes	
void setAsciiStream (int, InputStream, long)	4.0	Yes	
void setBigDecimal (int, BigDecimal)	1.0	Yes	
void setBinaryStream (int, InputStream)	4.0	Yes	When used with Blobs, the DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
void setBinaryStream (int, InputStream, int)	1.0	Yes	When used with Blobs, the DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.

<b>PreparedStatement Object (cont.)</b>	<b>Version Introduced</b>	<b>Supported</b>	<b>Comments</b>
void setBinaryStream (int, InputStream, long)	4.0	Yes	When used with Blobs, the DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
void setBlob (int, Blob)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
void setBlob (int, InputStream)	4.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
void setBlob (int, InputStream, long)	4.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.

<b>PreparedStatement Object (cont.)</b>	<b>Version Introduced</b>	<b>Supported</b>	<b>Comments</b>
void setBoolean (int, boolean)	1.0	Yes	
void setByte (int, byte)	1.0	Yes	
void setBytes (int, byte [])	1.0	Yes	When used with Blobs, the DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
void setCharacterStream (int, Reader)	4.0	Yes	
void setCharacterStream (int, Reader, int)	2.0 Core	Yes	
void setCharacterStream (int, Reader, long)	4.0	Yes	
void setClob (int, Clob)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
void setClob (int, Reader)	4.0	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
void setClob (int, Reader, long)	4.0	Yes	
void setDate (int, Date)	1.0	Yes	
void setDate (int, Date, Calendar)	2.0 Core	Yes	
void setDouble (int, double)	1.0	Yes	
void setFloat (int, float)	1.0	Yes	
void setInt (int, int)	1.0	Yes	

<b>PreparedStatement Object (cont.)</b>	<b>Version</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void setLong (int, long)	1.0		Yes	
void setNCharacterStream (int, Reader)	4.0		Yes	
void setNCharacterStream (int, Reader, long)	4.0		Yes	
void setNClob (int, NClob)	4.0		Yes	
void setNClob (int, Reader)	4.0		Yes	
void setNClob (int, Reader, long)	4.0		Yes	
void setNull (int, int)	1.0		Yes	
void setNull (int, int, String)	2.0 Core		Yes	
void setNString (int, String)	4.0		Yes	
void setObject (int, Object)	1.0		Yes	
void setObject (int, Object, int)	1.0		Yes	
void setObject (int, Object, int, int)	1.0		Yes	

PreparedStatement Object (cont.)	Version Introduced	Supported	Comments
void setQueryTimeout (int)	1.0	Yes	The DB2 driver supports setting a timeout value, in seconds, for a statement with DB2 v8.x and higher for Linux/UNIX/Windows and DB2 v8.1 for z/OS. If the execution of the statement exceeds the timeout value, the statement is timed out by the database server, and the driver throws an exception indicating that the statement was timed out. The DB2 driver throws an "unsupported method" exception with other DB2 versions.
			The Informix driver throws an "unsupported method" exception.
			The Oracle, SQL Server, and Sybase drivers support setting a timeout value, in seconds, for a statement. If the execution of the statement exceeds the timeout value, the statement is timed out by the database server, and the driver throws an exception indicating that the statement was timed out.
void setRef (int, Ref)	2.0 Core	No	Throws "unsupported method" exception.
void setShort (int, short)	1.0	Yes	
void setSQLXML (int, SQLXML)	4.0	Yes	
void setString (int, String)	1.0	Yes	

<b>PreparedStatement Object (cont.)</b>	<b>Version</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void setTime (int, Time)	1.0	Yes		
void setTime (int, Time, Calendar)	2.0 Core	Yes		
void setTimestamp (int, Timestamp)	1.0	Yes		
void setTimestamp (int, Timestamp, Calendar)	2.0 Core	Yes		
void setUnicodeStream (int, InputStream, int)	1.0	No		Throws "unsupported method" exception. This method was deprecated in JDBC 2.0.
<T> T unwrap(Class<T> iface)	4.0	Yes		
void setURL (int, URL)	3.0	No		Throws "unsupported method" exception.

## Ref Object

<b>Ref Object</b>	<b>Version</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
Methods				
(all)	2.0 Core	No		

## ResultSet Object

<b>ResultSet Object</b>	<b>Version</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
Methods				
boolean absolute (int)	2.0 Core	Yes		
void afterLast ()	2.0 Core	Yes		

<b>ResultSet Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void beforeFirst ()	2.0 Core	Yes	
void cancelRowUpdates ()	2.0 Core	Yes	
void clearWarnings ()	1.0	Yes	
void close ()	1.0	Yes	
void deleteRow ()	2.0 Core	Yes	
int findColumn (String)	1.0	Yes	
boolean first ()	2.0 Core	Yes	
Array getArray (int)	2.0 Core	No	Throws "unsupported method" exception.
Array getArray (String)	2.0 Core	No	Throws "unsupported method" exception.
InputStream getAsciiStream (int)	1.0	Yes	
InputStream getAsciiStream (String)	1.0	Yes	
BigDecimal getBigDecimal (int)	2.0 Core	Yes	
BigDecimal getBigDecimal (int, int)	1.0	Yes	
BigDecimal getBigDecimal (String)	2.0 Core	Yes	
BigDecimal getBigDecimal (String, int)	1.0	Yes	

<b>ResultSet Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
InputStream getBinaryStream (int)	1.0	Yes	The DB2 driver supports for all DB2 versions when retrieving BINARY, VARBINARY, and LONGVARBINARY data. The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries when retrieving BLOB data.
InputStream getBinaryStream (String)	1.0	Yes	The DB2 driver supports for all DB2 versions when retrieving BINARY, VARBINARY, and LONGVARBINARY data. The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries when retrieving BLOB data.
Blob getBlob (int)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.

<b>ResultSet Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
Blob getBlob (String)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
boolean getBoolean (int)	1.0	Yes	
boolean getBoolean (String)	1.0	Yes	
byte getByte (int)	1.0	Yes	
byte getByte (String)	1.0	Yes	
byte [] getBytes (int)	1.0	Yes	The DB2 driver supports for all DB2 versions when retrieving BINARY, VARBINARY, and LONGVARBINARY data. The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries when retrieving BLOB data.
byte [] getBytes (String)	1.0	Yes	The DB2 driver supports for all DB2 versions when retrieving BINARY, VARBINARY, and LONGVARBINARY data. The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries when retrieving BLOB data.

<b>ResultSet Object (cont.)</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
Reader getCharacterStream (int)	2.0 Core	Yes		
Reader getCharacterStream (String)	2.0 Core	Yes		
Clob getClob (int)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.	
Clob getClob (String)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.	
int getConcurrency ()	2.0 Core	Yes		
String getCursorName ()	1.0	No	Throws "unsupported method" exception.	
Date getDate (int)	1.0	Yes		
Date getDate (int, Calendar)	2.0 Core	Yes		
Date getDate (String)	1.0	Yes		
Date getDate (String, Calendar)	2.0 Core	Yes		
double getDouble (int)	1.0	Yes		
double getDouble (String)	1.0	Yes		
int getFetchDirection ()	2.0 Core	Yes		
int getFetchSize ()	2.0 Core	Yes		
float getFloat (int)	1.0	Yes		
float getFloat (String)	1.0	Yes		
int getHoldability ()	4.0	Yes		
int getInt (int)	1.0	Yes		

<b>ResultSet Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
int getInt (String)	1.0	Yes	
long getLong (int)	1.0	Yes	
long getLong (String)	1.0	Yes	
ResultSetMetaData getMetaData ()	1.0	Yes	
Reader getNCharacterStream (int)	4.0	Yes	
Reader getNCharacterStream (String)	4.0	Yes	
NClob getNClob (int)	4.0	Yes	
NClob getNClob (String)	4.0	Yes	
String getNString (int)	4.0	Yes	
String getNString (String)	4.0	Yes	
Object getObject (int)	1.0	Yes	Returns a Long object when called on DB2 BigInt columns.
Object getObject (int, Map)	2.0 Core	Yes	
Object getObject (String)	1.0	Yes	
Object getObject (String, Map)	2.0 Core	Yes	Map ignored.
Ref getRef (int)	2.0 Core	No	Throws "unsupported method" exception.
Ref getRef (String)	2.0 Core	No	Throws "unsupported method" exception.
int getRow ()	2.0 Core	Yes	
short getShort (int)	1.0	Yes	
short getShort (String)	1.0	Yes	
SQLXML getSQLXML (int)	4.0	Yes	
SQLXML getSQLXML (String)	4.0	Yes	

<b>ResultSet Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
Statement getStatement ()	2.0 Core	Yes	
String getString (int)	1.0	Yes	
String getString (String)	1.0	Yes	
Time getTime (int)	1.0	Yes	
Time getTime (int, Calendar)	2.0 Core	Yes	
Time getTime (String)	1.0	Yes	
Time getTime (String, Calendar)	2.0 Core	Yes	
Timestamp getTimestamp (int)	1.0	Yes	
Timestamp getTimestamp (int, Calendar)	2.0 Core	Yes	
Timestamp getTimestamp (String)	1.0	Yes	
Timestamp getTimestamp (String, Calendar)	2.0 Core	Yes	
int getType ()	2.0 Core	Yes	
InputStream getUnicodeStream (int)	1.0	No	Throws "unsupported method" exception. This method was deprecated in JDBC 2.0.
InputStream getUnicodeStream (String)	1.0	No	Throws "unsupported method" exception. This method was deprecated in JDBC 2.0.
URL getURL (int)	3.0	No	Throws "unsupported method" exception.
URL getURL (String)	3.0	No	Throws "unsupported method" exception.
SQLWarning getWarnings ()	1.0	Yes	
void insertRow ()	2.0 Core	Yes	

<b>ResultSet Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
boolean isAfterLast ()	2.0 Core	Yes	
boolean isBeforeFirst ()	2.0 Core	Yes	
boolean isClosed ()	4.0	Yes	
boolean isFirst ()	2.0 Core	Yes	
boolean isLast ()	2.0 Core	Yes	
boolean isWrapperFor (Class<?> iface)	4.0	Yes	
boolean last ()	2.0 Core	Yes	
void moveToCurrentRow ()	2.0 Core	Yes	
void moveToInsertRow ()	2.0 Core	Yes	
boolean next ()	1.0	Yes	
boolean previous ()	2.0 Core	Yes	
void refreshRow ()	2.0 Core	Yes	
boolean relative (int)	2.0 Core	Yes	
boolean rowDeleted ()	2.0 Core	Yes	
boolean rowInserted ()	2.0 Core	Yes	
boolean rowUpdated ()	2.0 Core	Yes	
void setFetchDirection (int)	2.0 Core	Yes	
void setFetchSize (int)	2.0 Core	Yes	
<T> T unwrap(Class<T> iface)	4.0	Yes	
void updateArray (int, Array)	3.0	No	Throws "unsupported method" exception.
void updateArray (String, Array)	3.0	No	Throws "unsupported method" exception.
void updateAsciiStream (int, InputStream, int)	2.0 Core	Yes	

<b>ResultSet Object (cont.)</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
void updateAsciiStream (int, InputStream, long)	4.0	Yes		
void updateAsciiStream (String, InputStream)	4.0	Yes		
void updateAsciiStream (String, InputStream, int)	2.0 Core	Yes		
void updateAsciiStream (String, InputStream, long)	4.0	Yes		
void updateBigDecimal (int, BigDecimal)	2.0 Core	Yes		
void updateBigDecimal (String, BigDecimal)	2.0 Core	Yes		
void updateBinaryStream (int, InputStream)	4.0	Yes		
void updateBinaryStream (int, InputStream, int)	2.0 Core	Yes		
void updateBinaryStream (int, InputStream, long)	4.0	Yes		
void updateBinaryStream (String, InputStream)	4.0	Yes		
void updateBinaryStream (String, InputStream, int)	2.0 Core	Yes		
void updateBinaryStream (String, InputStream, long)	4.0	Yes		

<b>ResultSet Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void updateBlob (int, Blob)	3.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
void updateBlob (int, InputStream)	4.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
void updateBlob (int, InputStream, long)	4.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.

<b>ResultSet Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void updateBlob (String, Blob)	3.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
void updateBlob (String, InputStream)	4.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
void updateBlob (String, InputStream, long)	4.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.  The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
void updateBoolean (int, boolean)	2.0 Core	Yes	
void updateBoolean (String, boolean)	2.0 Core	Yes	
void updateByte (int, byte)	2.0 Core	Yes	

<b>ResultSet Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void updateByte (String, byte)	2.0 Core	Yes	
void updateBytes (int, byte [])	2.0 Core	Yes	
void updateBytes (String, byte [])	2.0 Core	Yes	
void updateCharacterStream (int, Reader)	4.0	Yes	
void updateCharacterStream (int, Reader, int)	2.0 Core	Yes	
void updateCharacterStream (int, Reader, long)	4.0	Yes	
void updateCharacterStream (String, Reader)	4.0	Yes	
void updateCharacterStream (String, Reader, int)	2.0 Core	Yes	
void updateCharacterStream (String, Reader, long)	4.0	Yes	
void updateClob (int, Clob)	3.0	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
void updateClob (int, Reader)	4.0	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
void updateClob (int, Reader, long)	4.0	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.

<b>ResultSet Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void updateClob (String, Clob)	3.0	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
void updateClob (String, Reader)	4.0	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
void updateClob (String, Reader, long)	4.0	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.
void updateDate (int, Date)	2.0 Core	Yes	
void updateDate (String, Date)	2.0 Core	Yes	
void updateDouble (int, double)	2.0 Core	Yes	
void updateDouble (String, double)	2.0 Core	Yes	
void updateFloat (int, float)	2.0 Core	Yes	
void updateFloat (String, float)	2.0 Core	Yes	
void updateInt (int, int)	2.0 Core	Yes	
void updateInt (String, int)	2.0 Core	Yes	
void updateLong (int, long)	2.0 Core	Yes	
void updateLong (String, long)	2.0 Core	Yes	
void updateNCharacterStream (int, Reader)	4.0	Yes	
void updateNCharacterStream (int, Reader, long)	4.0	Yes	

<b>ResultSet Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void updateNCharacterStream (String, Reader)	4.0	Yes	
void updateNCharacterStream (String, Reader, long)	4.0	Yes	
void updateNClob (int, NClob)	4.0	Yes	
void updateNClob (int, Reader)	4.0	Yes	
void updateNClob (int, Reader, long)	4.0	Yes	
void updateNClob (String, NClob)	4.0	Yes	
void updateNClob (String, Reader)	4.0	Yes	
void updateNClob (String, Reader, long)	4.0	Yes	
void updateNString (int, String)	4.0	Yes	
void updateNString (String, String)	4.0	Yes	
void updateNull (int)	2.0 Core	Yes	
void updateNull (String)	2.0 Core	Yes	
void updateObject (int, Object)	2.0 Core	Yes	
void updateObject (int, Object, int)	2.0 Core	Yes	
void updateObject (String, Object)	2.0 Core	Yes	
void updateObject (String, Object, int)	2.0 Core	Yes	
void updateRef (int, Ref)	3.0	No	Throws "unsupported method" exception.

<b>ResultSet Object (cont.)</b> <b>Methods</b>	<b>Version</b> <b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void updateRef (String, Ref)	3.0	No	Throws "unsupported method" exception.
void updateRow ()	2.0 Core	Yes	
void updateShort (int, short)	2.0 Core	Yes	
void updateShort (String, short)	2.0 Core	Yes	
void updateSQLXML (int, SQLXML)	4.0	Yes	
void updateSQLXML (String, SQLXML)	4.0	Yes	
void updateString (int, String)	2.0 Core	Yes	
void updateString (String, String)	2.0 Core	Yes	
void updateTime (int, Time)	2.0 Core	Yes	
void updateTime (String, Time)	2.0 Core	Yes	
void updateTimestamp (int, Timestamp)	2.0 Core	Yes	
void updateTimestamp (String, Timestamp)	2.0 Core	Yes	
boolean wasNull ()	1.0	Yes	

## ResultSetMetaData Object

<b>ResultSetMetaData Object</b> <b>Methods</b>	<b>Version</b> <b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
String getCatalogName (int)	1.0	Yes	
String getColumnClassName (int)	2.0 Core	Yes	

<b>ResultSetMetaData Object (cont.)</b>	<b>Version</b>		
	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
int getColumnCount ()	1.0	Yes	
int getColumnDisplaySize (int)	1.0	Yes	
String getColumnLabel (int)	1.0	Yes	
String getColumnName (int)	1.0	Yes	
int getColumnType (int)	1.0	Yes	
String getColumnTypeName (int)	1.0	Yes	
int getPrecision (int)	1.0	Yes	
int getScale (int)	1.0	Yes	
String getSchemaName (int)	1.0	Yes	
String getTableName (int)	1.0	Yes	
boolean isAutoIncrement (int)	1.0	Yes	
boolean isCaseSensitive (int)	1.0	Yes	
boolean isCurrency (int)	1.0	Yes	
boolean isDefinitelyWritable (int)	1.0	Yes	
int isNullable (int)	1.0	Yes	
boolean isReadOnly (int)	1.0	Yes	
boolean isSearchable (int)	1.0	Yes	
boolean isSigned (int)	1.0	Yes	
boolean isWrapperFor (Class<?> iface)	4.0	Yes	
boolean isWritable (int)	1.0	Yes	
<T> T unwrap(Class<T> iface)	4.0	Yes	

## RowSet Object

RowSet Object	Version			
Methods	Introduced	Supported	Comments	
(all)	2.0	Optional	No	

## SavePoint Object

SavePoint Object	Version			
Methods	Introduced	Supported	Comments	
(all)	3.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.	

## Statement Object

Statement Object	Version			
Methods	Introduced	Supported	Comments	
void addBatch (String)	2.0 Core	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.	

<b>Statement Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void cancel ()	1.0	Yes	The DB2 driver cancels the execution of the statement with DB2 v8.x and higher for Linux/UNIX/Windows and DB2 v8.1 for z/OS. If the statement is cancelled by the database server, the driver throws an exception indicating that it was cancelled. The DB2 driver throws an "unsupported method" exception with other DB2 versions.  The Informix driver throw an "unsupported method" exception.
			The Oracle, SQL Server, and Sybase drivers cancel the execution of the statement. If the statement is cancelled by the database server, the driver throws an exception indicating that it was cancelled.
void clearBatch ()	2.0 Core	Yes	
void clearWarnings ()	1.0	Yes	
void close ()	1.0	Yes	
boolean execute (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.
boolean execute (String, int)	3.0	Yes	
boolean execute (String, int [])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws "unsupported method" exception.

<b>Statement Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
boolean execute (String, String [])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws "unsupported method" exception.
int [] executeBatch ()	2.0 Core	Yes	
ResultSet executeQuery (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.
int executeUpdate (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.
int executeUpdate (String, int)	3.0	Yes	
int executeUpdate (String, int [])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws "unsupported method" exception.
int executeUpdate (String, String [])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws "unsupported method" exception.
Connection getConnection ()	2.0 Core	Yes	
int getFetchDirection ()	2.0 Core	Yes	
int getFetchSize ()	2.0 Core	Yes	

<b>Statement Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
ResultSet getGeneratedKeys ()	3.0	Yes	The DB2, SQL Server, and Sybase drivers return the last value inserted into an identity column. If an identity column does not exist in the table, the drivers return an empty result set.  The Informix driver returns the last value inserted into a Serial or Serial8 column. If a Serial or Serial8 column does not exist in the table, the driver returns an empty result set.  The Oracle driver returns the ROWID of the last row inserted.
int getMaxFieldSize ()	1.0	Yes	
int getMaxRows ()	1.0	Yes	
boolean getMoreResults ()	1.0	Yes	
boolean getMoreResults (int)	3.0	Yes	
int getQueryTimeout ()	1.0	Yes	The DB2 driver returns the timeout value, in seconds, set for the statement with DB2 v8.x and higher for Linux/UNIX/Windows and DB2 v8.1 for z/OS. The DB2 driver returns 0 with other DB2 versions.  The Informix driver return 0.  The Oracle, SQL Server, and Sybase drivers return the timeout value, in seconds, set for the statement.
ResultSet getResultSet ()	1.0	Yes	

<b>Statement Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
int getResultSetConcurrency ()	2.0 Core	Yes	
int getResultSetHoldability ()	3.0	Yes	
int getResultSetType ()	2.0 Core	Yes	
int getUpdateCount ()	1.0	Yes	
SQLWarning getWarnings ()	1.0	Yes	
boolean isClosed ()	4.0	Yes	
boolean isPoolable ()	4.0	Yes	
boolean isWrapperFor (Class<?> iface)	4.0	Yes	
void setCursorName (String)	1.0	No	Throws "unsupported method" exception.
void setEscapeProcessing (boolean)	1.0	Yes	Ignored.
void setFetchDirection (int)	2.0 Core	Yes	
void setFetchSize (int)	2.0 Core	Yes	
void setMaxFieldSize (int)	1.0	Yes	
void setMaxRows (int)	1.0	Yes	
void setPoolable (boolean)	4.0	Yes	

<b>Statement Object (cont.)</b>	<b>Version</b>		
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>
void setQueryTimeout (int)	1.0	Yes	<p>The DB2 driver supports setting a timeout value, in seconds, for a statement with DB2 v8.x and higher for Linux/UNIX/Windows and DB2 v8.1 for z/OS. If the execution of the statement exceeds the timeout value, the statement is timed out by the database server, and the driver throws an exception indicating that the statement was timed out. The DB2 driver throws an "unsupported method" exception with other DB2 versions.</p> <p>The Informix driver throw an "unsupported method" exception.</p> <p>The Oracle, SQL Server, and Sybase drivers support setting a timeout value, in seconds, for a statement. If the execution of the statement exceeds the timeout value, the statement is timed out by the database server, and the driver throws an exception indicating that the statement was timed out.</p>
<T> T unwrap(Class<T> iface)	4.0	Yes	

## StatementEventListener Object

<b>StatementEventListener</b>				
<b>Object</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
void statementClosed (event)	4.0	Yes		
void statementErrorOccurred (event)	4.0	Yes		

## Struct Object

<b>Struct Object</b>				
<b>Object</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
(all)	2.0	No		

## XAConnection Object

<b>XAConnection Object</b>				
<b>Object</b>	<b>Version</b>			
<b>Methods</b>	<b>Introduced</b>	<b>Supported</b>	<b>Comments</b>	
(all)	2.0	Optional	Yes	Supported for all drivers, except for DB2 v7.x for Linux/UNIX/Windows and DB2 v7.x and v8.1 for z/OS.

## XADatasource Object

XADatasource Object	Version		
Methods	Introduced	Supported	Comments
(all)	2.0 Optional	Yes	Supported for all drivers, except for DB2 v7.x for Linux/UNIX/Windows and DB2 v7.x and v8.1 for z/OS.

## XAResource Object

XAResource Object	Version		
Methods	Introduced	Supported	Comments
(all)	2.0 Optional	Yes	Supported for all drivers, except for DB2 v7.x for Linux/UNIX/Windows and DB2 v7.x and v8.1 for z/OS.



# B JDBC Extensions

This appendix describes the JDBC extensions provided by the com.ddtek.jdbc.extensions package. The interfaces in this package are:

- [DDBulkLoad Interface](#) (see “[DDBulkLoad Interface](#)” on page 603). The methods in this interface allow your application to perform bulk load operations. See [Appendix K “Using DataDirect Bulk Load” on page 829](#) for information about using DataDirect Bulk Load.
- [ExtConnection Interface](#) (see “[ExtConnection Interface](#)” on page 613). The methods in this interface allow you to perform the following actions:
  - Store and return client information. See [Appendix C “Client Information for Connections” on page 621](#) for more information.
  - Switch the user associated with a connection to another user to minimize the number of connections that are required in a connection pool. See “[Using Reauthentication](#)” on page 55 for more information.
  - Access the DataDirect Statement Pool Monitor from a connection. See [Appendix J “Statement Pool Monitor” on page 811](#) for information about using the DataDirect Statement Pool Monitor.
- [ExtDatabaseMetaData Interface](#) (see “[ExtDatabaseMetaData Interface](#)” on page 619). The methods in this interface allow you to obtain property information that is useful for storing and retrieving client information. See [Appendix C “Client Information for Connections” on page 621](#) for more information.

- ExtLogControl (see “[ExtLogControl Class](#)” on page 620). The methods in this interface are useful for determining if DataDirect Spy logging is enabled and turning on and off DataDirect Spy logging if enabled. See [Appendix H “Tracking JDBC Calls with DataDirect Spy™”](#) on page 783 for more information.

---

## Using JDBC Wrapper Methods to Access the JDBC Extensions

The Wrapper methods allow an application to access vendor-specific classes. The following example shows how to access the DataDirect-specific ExtConnection class using the Wrapper methods:

```
ExtStatementPoolMonitor monitor = null;
Class<ExtConnection> cls = ExtConnection.class;
if (con.isWrapperFor(cls)) {
    ExtConnection extCon = con.unwrap(cls);
    extCon.setClientUser("Joe Smith");
    monitor = extCon.getStatementPoolMonitor();
}
...
if(monitor != null) {
    long hits = monitor.getHitCount();
    long misses = monitor.getMissCount();
}
...
```

---

# DDBulkLoad Interface

The methods of this interface are supported by the DB2, Oracle, SQL Server, and Sybase drivers.

<b>DDBulkLoad Interface</b>	
<b>Methods</b>	<b>Description</b>
void clearWarnings()	Clears all warnings that were generated by this DDBulkLoad object.
void close()	Releases a DDBulkLoad object's resources immediately instead of waiting for this to occur when the connection is closed.
long export(File)	<p>Exports all rows from the database table into the specified CSV file specified by a file reference. The database table is specified using the setTableName() method. If the CSV file does not already exist, the driver creates it when the export() method is executed. In addition, the driver creates a bulk load configuration file matching the CSV file. See <a href="#">"Exporting Data to a CSV File" on page 834</a> for more information.</p> <p>This method also returns the number of rows that were successfully exported from the database table.</p>
long export(ResultSet, File)	<p>Exports all rows from the specified ResultSet into the CSV file specified by a file reference. If the CSV file does not already exist, the driver creates it when the export() method is executed. In addition, the driver creates a bulk load configuration file matching the CSV file. See <a href="#">"Exporting Data to a CSV File" on page 834</a> for more information.</p> <p>This method also returns the number of rows that were successfully exported from the ResultSet object.</p>

<b>DDBulkLoad Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
long export(String)	<p>Exports all rows from the database table into the CSV file specified by name. The database table is specified using the <code>setTableName()</code> method. If the CSV file does not already exist, the driver creates it when the <code>export()</code> method is executed. In addition, the driver creates a bulk load configuration file matching the CSV file. See <a href="#">"Exporting Data to a CSV File" on page 834</a> for more information.</p> <p>This method also returns the number of rows that were successfully exported from the database table.</p>
long getBatchSize()	Returns the number of rows that the driver sends to the database at a time when bulk loading data.
long getBinaryThreshold()	Returns the maximum size (in bytes) of binary data that can be exported from the database to the CSV file. Once this size is reached, binary data is written to one or multiple external overflow files. See <a href="#">"External Overflow Files" on page 842</a> for more information.
long getCharacterThreshold()	Returns the maximum size (in bytes) of character data that can be exported from the database to the CSV file. Once this size is reached, character data is written to one or multiple external overflow files. See <a href="#">"External Overflow Files" on page 842</a> for more information.
String getCodePage()	Returns the code page that the driver uses for the CSV file. See <a href="#">"Character Set Conversions" on page 840</a> for more information.
String getConfigFile()	Returns the name of the bulk load configuration file. See <a href="#">"Bulk Load Configuration File" on page 838</a> for more information.
String getDiscardFile()	Returns the name of the discard file. The discard file contains rows that were unable to be loaded into the database as the result of a bulk load operation. See <a href="#">"Discard File" on page 843</a> for more information.

<b>DDBulkLoad Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
long getErrorTolerance()	Returns the number of errors that can occur before this DDBulkLoad object will end the bulk load operation.
String getLogFile()	Returns the name of the log file. The log file records information about each bulk load operation. <a href="#">"Logging" on page 836</a> for more information.
long getNumRows()	Returns the maximum number of rows from the CSV file or ResultSet object the driver will load when the load() method is executed.
Properties getProperties()	Returns the properties specified for a DDBulkLoad object. Properties are specified using the setProperties() method.
long getReadBufferSize()	Returns the size (in KB) of the buffer that is used to read the CSV file.
long getStartPosition()	Returns the position (number of the row) in a CSV file or ResultSet object from which the driver will start loading. The position is specified using the setStartPosition() method.
void getTableName()	Returns the name of the database table the data will be loaded into or exported from. See <a href="#">"Loading Data" on page 835</a> and <a href="#">"Exporting Data to a CSV File" on page 834</a> for more information.
long getTimeout()	Returns the number of seconds for the bulk load operation to complete before it times out. The timeout is specified using the setTimeout() method.
SQLWarning getWarnings()	Returns any warnings generated by this DDBulkLoad object.
long getWarningTolerance()	Returns the maximum number of warnings that can occur. Once the maximum number is reached, the bulk load operation will end.

<b>DDBulkLoad Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
long load(File)	<p>Loads data from the CSV file specified by a file reference into a database table. The database table is specified using the <code>setTableName()</code> method. This method also returns the number of rows successfully loaded.</p> <p>If logging is enabled using the <code>setLogFile()</code> method, information about the bulk load operation is recorded in the log file. If a discard file is created using the <code>setDiscardFile()</code> method, rows that were unable to be loaded are recorded in the discard file. See “<a href="#">Logging</a>” on page 836 and “<a href="#">Discard File</a>” on page 843 for more information.</p> <p>Before the bulk load operation is performed, your application can verify that the data in the CSV file is compatible with the structure of the target database using the <code>validateTableFromFile()</code> method.</p>
long load(String)	<p>Loads data from the CSV file specified by file name into a database table. The database table is specified using the <code>setTableName()</code> method. This method also returns the number of rows successfully loaded.</p> <p>If logging is enabled using the <code>setLogFile()</code> method, information about the bulk load operation is recorded in the log file. If a discard file is created using the <code>setDiscardFile()</code> method, rows that were unable to be loaded are recorded in the discard file. See “<a href="#">Logging</a>” on page 836 and “<a href="#">Discard File</a>” on page 843 for more information.</p> <p>Before the bulk load operation is performed, your application can verify that the data in the CSV file is compatible with the structure of the target database using the <code>validateTableFromFile()</code> method.</p>

DDBulkLoad Interface (cont.)	
Methods	Description
long load(ResultSet)	<p>Loads data from a ResultSet object into the database table specified using the setTableName() method. This method also returns the number of rows successfully loaded.</p> <p>If logging is enabled using the setLogFile() method, information about the bulk load operation is recorded in the log file. See <a href="#">"Logging" on page 836</a> for more information.</p> <p>The structure of the database that produced the ResultSet object must match the structure of the target database. If not, the driver throws an exception.</p>
void setBatchSize(long)	<p>Specifies the number of rows that the driver sends 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.</p> <p>If unspecified, the driver uses a value of 2048.</p>

<b>DDBulkLoad Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
<code>void setBinaryThreshold(long)</code>	<p>Specifies the maximum size (in bytes) of binary data to be exported to the CSV file. Any column with data over this threshold is exported into individual external overflow files and a marker of the format <code>{DD LOBFILE "filename"}</code> is placed in the CSV file to signify that the data for this column is located in an external file. The format for overflow file names is:</p> <p><code>csv_filename_xxxxxx.lob</code></p> <p>where:</p> <p><code>csv_filename</code> is the name of the CSV file.</p> <p><code>xxxxxx</code> is a 6-digit number that increments the overflow file.</p> <p>For example, if multiple overflow files are created for a CSV file named CSV1, the file names would look like this:</p> <p><code>CSV1.000001.lob</code>  <code>CSV1.000002.lob</code>  <code>CSV1.000003.lob</code>  <code>...</code></p> <p>If set to -1, this method tells the driver to never overflow binary data to external files.</p> <p>If unspecified, the driver uses a value of 4096.</p> <p>See "<a href="#">External Overflow Files</a>" on page 842 for more information.</p>

<b>DDBulkLoad Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
<code>void setCharacterThreshold(long)</code>	<p>Specifies the maximum size (in bytes) of character data to be exported to the CSV file. Any column with data over this threshold is exported into individual external overflow files and a marker of the format {DD LOBFILE "filename"} is placed in the CSV file to signify that the data for this column is located in an external file.</p> <p>The format for overflow file names is:</p> <p><code>csv_filename_xxxxxx.lob</code></p> <p>where:</p> <p><code>csv_filename</code> is the name of the CSV file.</p> <p><code>xxxxxx</code> is a 6-digit number that increments the overflow file.</p> <p>For example, if multiple overflow files are created for a CSV file named CSV1, the file names would look like this:</p> <p>CSV1.000001.lob      CSV1.000002.lob      CSV1.000003.lob      ...</p> <p>If set to -1, this method tells the driver to never overflow character data to external files.</p> <p>If unspecified, the driver uses a value of 4096.</p> <p>See "<a href="#">External Overflow Files</a>" on page 842 for more information.</p>
<code>void setCodePage(String)</code>	Specifies the code page the driver uses for the CSV file. See " <a href="#">Character Set Conversions</a> " on page 840 for more information.

<b>DDBulkLoad Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
<code>void setConfigFile(String)</code>	<p>Specifies the fully qualified directory and file name of the bulk load configuration file. If the Column Info section in the bulk load configuration file is specified, the driver uses it to map the columns in the CSV file to the columns in the target database table when performing a bulk load operation.</p> <p>If unspecified, the name of the bulk load configuration file is assumed to be <code>csv_filename.xml</code>, where <code>csv_filename</code> is the file name of the CSV file.</p> <p>If set to an empty string, the driver does not try to use the bulk load configuration file and reads all data from the CSV file as character data.</p> <p>See “<a href="#">Bulk Load Configuration File</a>” on page 838 for more information.</p>
<code>void setDiscardFile(String)</code>	<p>Specifies the fully qualified directory and file name of the discard file. The discard file contains rows that were unable to be loaded from a CSV file into the database as the result of a bulk load operation. After fixing the reported issues in the discard file, the bulk load can be reissued, using the discard file as the CSV file.</p> <p>If unspecified, a discard file is not created.</p> <p>See “<a href="#">Discard File</a>” on page 843 for more information.</p>
<code>void setErrorTolerance(long)</code>	<p>Specifies the maximum number of errors that can occur. Once the maximum number is reached, the bulk load operation will end. Errors are written to the log file.</p> <p>If set to 0, no errors are tolerated; the bulk load operation fails if any error is encountered. Any rows that were processed before the error occurred are loaded.</p> <p>If unspecified or set to -1, an infinite number of errors are tolerated.</p>

<b>DDBulkLoad Interface (cont.)</b>															
<b>Methods</b>	<b>Description</b>														
<code>void setLogFile(String)</code>	<p>Specifies the fully qualified directory and file name of the log file. The log file records information about each bulk load operation.</p> <p>If unspecified, a log file is not created.</p> <p>See "<a href="#">Logging</a>" on page 836 for more information.</p>														
<code>void setNumRows()</code>	Specifies the maximum number of rows from the CSV file or ResultSet object the driver will load.														
<code>void setProperties(Properties)</code>	<p>Specifies one or more of the following properties for a DDBulkLoad object:</p> <table> <tr> <td>tableName</td> <td>numRows</td> </tr> <tr> <td>codePage</td> <td>binaryThreshold</td> </tr> <tr> <td>timeout</td> <td>characterThreshold</td> </tr> <tr> <td>logFile</td> <td>errorTolerance</td> </tr> <tr> <td>discardFile</td> <td>warningTolerance</td> </tr> <tr> <td>configFile</td> <td>readBufferSize</td> </tr> <tr> <td>startPosition</td> <td>batchSize</td> </tr> </table> <p>These properties also can be set using the corresponding setxxx() methods.</p>	tableName	numRows	codePage	binaryThreshold	timeout	characterThreshold	logFile	errorTolerance	discardFile	warningTolerance	configFile	readBufferSize	startPosition	batchSize
tableName	numRows														
codePage	binaryThreshold														
timeout	characterThreshold														
logFile	errorTolerance														
discardFile	warningTolerance														
configFile	readBufferSize														
startPosition	batchSize														
<code>void setReadBufferSize(long)</code>	<p>Specifies the size (in KB) of the buffer that is used to read the CSV file.</p> <p>If unspecified, the driver uses a value of 2048.</p>														
<code>void setStartPosition()</code>	Specifies the position (number of the row) in a CSV file or ResultSet object from which the bulk load operation will start. For example, if a value of 10 is specified, the first 9 rows of the CSV file are skipped and the first row loaded is row 10.														
<code>void setTableName(String)</code>	<p>When loading data into a database table, specifies the name of the table into which the data is loaded.</p> <p>When exporting data from a database table, specifies the name of the table the data from which the data is exported. If the specified table does not exist, the driver throws an exception. See "<a href="#">Loading Data</a>" on page 835 and "<a href="#">Exporting Data to a CSV File</a>" on page 834 for more information.</p>														

<b>DDBulkLoad Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
<code>void setTimeout(long)</code>	Sets the maximum number of seconds that can elapse for this bulk load operation to complete. Once this number is reached, the bulk load operation times out.
<code>void setWarningTolerance(long)</code>	Specifies the maximum number of warnings that can occur. Once the maximum is reached, the bulk load operation will end. Warnings are written to the log file.  If set to 0, no warnings are tolerated; the bulk load operation fails if any warning is encountered.  If unspecified or set to -1, an infinite number of warnings are tolerated.
<code>Properties validateTableFromFile()</code>	Verifies the metadata in the bulk load configuration file against the structure of the database table to which the data is loaded. This method is used to ensure that the data in a CSV file is compatible with the structure of the target database table before the actual bulk load operation is performed. The driver performs checks to detect mismatches of the following types:  Data types Column sizes Code pages Column info  This method returns a Properties object with an entry for each of these checks: <ul style="list-style-type: none"><li>■ If no mismatches are found, the Properties object does not contain any messages.</li><li>■ If minor mismatches are found, the Properties object lists the problems.</li><li>■ If problems are detected that would prevent a successful bulk load operation, for example, if the target table does not exist, the driver throws an exception.</li></ul> See " <a href="#">"Verifying the Bulk Load Configuration File"</a> on page 840 for more information.

---

# ExtConnection Interface

<b>ExtConnection Interface</b>	
<b>Methods</b>	<b>Description</b>
public void abortConnection()	Closes the current connection and marks the connection as closed. This method does not attempt to obtain any locks when closing the connection. If subsequent operations are performed on the connection, the driver throws an exception.
String getClientAccountingInfo()	Returns the accounting client information on the connection or an empty string if the accounting client information value or the connection has not been set. If getting accounting client information is supported by the database and this operation fails, the driver throws an exception.
String getClientApplicationName()	Returns the name of the client application on the connection or an empty string if the client name value for the connection has not been set. If getting client name information is supported by the database and this operation fails, the driver throws an exception.
String getClientHostname()	Returns the name of the host used by the client application on the connection or an empty string if the client hostname value in the database has not been set. If getting host name information is supported by the database and this operation fails, the driver throws an exception.
String getClientUser()	Returns the user ID of the client on the connection or an empty string if the client user ID value for the connection has not been set. The user ID may be different from the user ID establishing the connection. If getting user ID application information is supported by the database and this operation fails, the driver throws an exception.

<b>ExtConnection Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
<code>String getCurrentUser()</code>	<p>Returns the current user of the connection. If reauthentication was performed on the connection, the current user may be different than the user that created the connection. For the DB2 and Oracle drivers, the current user is the same as the user reported by <code>DatabaseMetaData.getUserName()</code>. For the SQL Server driver, the current user is the login user name. <code>DatabaseMetaData.getUserName()</code> reports the user name the login user name is mapped to in the database.</p> <p>See “<a href="#">Using Reauthentication</a>” on page 55 for more information.</p>
<code>ExtStatementPoolMonitor getStatementPoolMonitor()</code>	<p>Returns an <code>ExtStatementPoolMonitor</code> object for the statement pool associated with the connection. If the connection does not have a statement pool, this method returns null. See “<a href="#">Using DataDirect-Specific Methods</a>” on page 815 for more information.</p>
<code>void resetUser(String)</code>	<p>Specifies a non-null string that resets the current user on the connection to the user that created the connection. It also restores the current schema, current path, or current database to the original value used when the connection was created. If reauthentication was performed on the connection, this method is useful to reset the connection to the original user.</p> <p>For the SQL Server driver, the current user is the login user name.</p> <p>The driver throws an exception in the following circumstances:</p> <ul style="list-style-type: none"> <li>■ The driver cannot change the current user to the initial user.</li> <li>■ A transaction is active on the connection.</li> </ul>

<b>ExtConnection Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
void setClientAccountingInfo(String)	<p>Specifies a non-null string that sets the accounting client information on the connection. Some databases include this information in their usage reports. The maximum length allowed for accounting information for a particular database can be determined by calling the ExtDatabaseMetaData.getClientAccountingInfoLength method. If the length of the information specified is longer than the maximum length allowed, the information is truncated to the maximum length, and the driver generates a warning.</p> <p>If setting accounting client information is supported by the database and this operation fails, the driver throws an exception.</p>
void setClientApplicationName(String)	<p>Specifies a non-null string that sets the name of the client application on the connection. The maximum client name length allowed for a particular database can be determined by calling the ExtDatabaseMetaData.getClientApplicationNameLength method. If the length of the client application name specified is longer than the maximum name length allowed, the name is truncated to the maximum length allowed, and the driver generates a warning.</p> <p>If setting client name information is supported by the database and this operation fails, the driver throws an exception.</p>

<b>ExtConnection Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
<code>void setClientHostname(String)</code>	<p>Specifies a non-null string that sets the name of the host used by the client application on the connection. The maximum hostname length allowed for a particular database can be determined by calling the <code>ExtDatabaseMetaData.getClientHostnameLength</code> method. If the length of the hostname specified is longer than the maximum hostname length allowed, the hostname is truncated to the maximum hostname length, and the driver generates a warning.</p> <p>If setting hostname information is supported by the database and this operation fails, the driver throws an exception.</p>
<code>void setClientUser(String)</code>	<p>Specifies a non-null string that sets the user ID of the client on the connection. This user ID may be different from the user ID establishing the connection. The maximum user ID length allowed for a particular database can be determined by calling the <code>ExtDatabaseMetaData.getClientUserLength</code> method. If the length of the user ID specified is longer than the maximum length allowed, the user ID is truncated to the maximum user ID length, and the driver generates a warning.</p> <p>If setting user ID information is supported by the database and this operation fails, the driver throws an exception.</p>
<code>void setCurrentUser(String)</code>	<p>Specifies a non-null string that sets the current user on the connection. This method is used to perform reauthentication on a connection.</p> <p>For the SQL Server driver, the current user is the login user name.</p> <p>The driver throws an exception in the following circumstances:</p> <ul style="list-style-type: none"> <li>■ The driver is connected to a database server that does not support reauthentication.</li> <li>■ The database server rejects the request to change the user on the connection.</li> <li>■ A transaction is active on the connection.</li> </ul>

<b>ExtConnection Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
<code>void setCurrentUser(String, Properties)</code>	<p>Specifies a non-null string that sets the current user on the connection. This method is used to perform reauthentication on a connection. In addition, this method sets options that control how the driver handles reauthentication. The options that are supported depend on the driver. See the DB2 driver, Oracle driver, and SQL Server driver chapters for information on which options are supported by each driver.</p> <p>For the SQL Server driver, the current user is the login user name.</p> <p>The driver throws an exception in the following circumstances:</p> <ul style="list-style-type: none"> <li>■ The driver is connected to a database server that does not support reauthentication.</li> <li>■ The database server rejects the request to change the user on the connection.</li> <li>■ A transaction is active on the connection.</li> </ul>
<code>void setCurrentUser(javax.security.auth.Subject)</code>	<p>Specifies a non-null string that sets the current user on the connection to the user specified by the javax.security.auth.Subject object. This method is used to perform reauthentication on a connection.</p> <p>For the SQL Server driver, the current user is the login user name.</p> <p>The driver throws an exception in the following circumstances:</p> <ul style="list-style-type: none"> <li>■ The driver does not support reauthentication.</li> <li>■ The driver is connected to a database server that does not support reauthentication.</li> <li>■ The database server rejects the request to change the user on the connection.</li> <li>■ A transaction is active on the connection.</li> </ul>

<b>ExtConnection Interface (cont.)</b>	
<b>Methods</b>	<b>Description</b>
<code>void setCurrentUser (javax.security.auth.Subject, Properties)</code>	<p>Specifies a non-null string that sets the current user on the connection to the user specified by the <code>javax.security.auth.Subject</code> object. This method is used to perform reauthentication on a connection. In addition, this method sets options that control how the driver handles reauthentication. The options that are supported depend on the driver. See the DB2 driver, Oracle driver, and SQL Server driver chapters for information on which options are supported by each driver.</p> <p>For the SQL Server driver, the current user is the login user name.</p> <p>The driver throws an exception in the following circumstances:</p> <ul style="list-style-type: none"> <li>■ The driver does not support reauthentication.</li> <li>■ The driver is connected to a database server that does not support reauthentication.</li> <li>■ The database server rejects the request to change the user on the connection.</li> <li>■ A transaction is active on the connection.</li> </ul>
<code>boolean supportsReauthentication()</code>	Indicates whether the database connection supports reauthentication. If true is returned, you can perform reauthentication on the connection. If false is returned, any attempt to perform reauthentication on the connection throws an exception.

---

# ExtDatabaseMetaData Interface

<b>ExtDatabaseMetaData Interface</b>	
<b>Methods</b>	<b>Description</b>
int getClientApplicationNameLength()	Returns the maximum length of the client application name. A value of 0 indicates that the client application name is stored locally in the driver, not in the database. There is no maximum length if the application name is stored locally.
int getClientUserLength()	Returns the maximum length of the client user ID. A value of 0 indicates that the client user ID is stored locally in the driver, not in the database. There is no maximum length if the client user ID is stored locally.
int getClientHostnameLength()	Returns the maximum length of the hostname. A value of 0 indicates that the hostname is stored locally in the driver, not in the database. There is no maximum length if the hostname is stored locally.
int getClientAccountingInfoLength()	Returns the maximum length of the accounting information. A value of 0 indicates that the accounting information is stored locally in the driver, not in the database. There is no maximum length if the hostname is stored locally.

## ExtLogControl Class

<b>ExtLogControl Class</b>	
<b>Methods</b>	<b>Description</b>
void setEnableLogging(boolean enable disable)	If DataDirect Spy was enabled when the connection was created, you can turn on or off DataDirect Spy logging at runtime using this method. If true, logging is turned on. If false, logging is turned off. If DataDirect Spy logging was not enabled when the connection was created, calling this method has no effect.
boolean getEnableLogging()	Indicates whether DataDirect Spy logging was enabled when the connection was created and whether logging is turned on. If the returned value is true, logging is turned on. If the returned value is false, logging is turned off.

# C Client Information for Connections

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

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

For DB2 V9.5 and V9.7 for Linux/UNIX/Windows and DB2 for z/OS, this information can feed directly into the Workload Manager (WLM) for workload management and monitoring purposes. See “[DB2 Workload Manager \(WLM\) Attributes](#)” on [page 628](#) for more information about using the WLM.

---

## How Databases Store Client Information

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

For example, let's assume that the following code returns a pooled connection to a DB2 V9.1 for Linux/UNIX/Windows database and sets a client application name for that connection. In this example, the application sets the application name SALES157 using the driver property ApplicationName.

```
// Get Database Connection  
Connection con = DriverManager.getConnection  
( "jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc", "TEST", "secret",  
    ApplicationName=SALES157 );  
...  
SALES157 is stored by the DB2 database in the CURRENT  
CLIENT_APPLNAME register, the location that DB2 reserves for  
this information. When the connection to the database is closed,  
the connection is returned to the connection pool as usual and  
the client information on the connection is reset to an empty  
string. If we used this same scenario against a MySQL database,  
which does not support storing client information, the driver  
would store the application name locally.
```

---

# Storing Client Information

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

- Using the driver properties listed in [Table C-1](#). See the specific driver chapters for a description of each property.
- Using the following JDBC methods:
  - `Connection.setClientInfo(properties)`
  - `Connection.setClientInfo(property_name, value)`
- See [“Connection Object” on page 552](#) for more information about these JDBC methods.
- Using the JDBC extension methods provided in the `com.ddtek.jdbc.extensions` package. See [Appendix B “JDBC Extensions” on page 601](#) for more information about the `com.ddtek.jdbc.extensions` package.

[Table C-1](#) shows the driver properties your application can use to store client information and where that client information is stored for each database.

**Table C-1. Database Locations for Storing Client Information**

<b>Property</b>	<b>Description</b>	<b>Database</b>	<b>Location</b>
AccountingInfo	Additional information that may be used for accounting or troubleshooting purposes, such as an accounting ID	DB2	CURRENT CLIENT_ACCTNG register (DB2 for Linux/UNIX/Windows) or CLIENT_ACCTNG register (DB2 for z/OS and DB2 for iSeries).
		Informix	Local cache.
		MySQL	Local cache.
		Oracle	CLIENT_INFO value in the V\$SESSION table.
		Microsoft SQL Server	Local cache.
		Sybase	Local cache.
ApplicationName	Name of the application currently using the connection	DB2	CURRENT CLIENT_APPLNAME register (DB2 for Linux/UNIX/Windows) or CLIENT_APPLNAME register (DB2 for z/OS and DB2 for iSeries). For DB2 V9.1 and higher for Linux/UNIX/Windows, this value is also stored in the APPL_NAME value in the SYSIBMADM.APPLICATIONS table.
		Informix	Local cache.
		MySQL	Local cache.
		Oracle	CLIENT_IDENTIFIER attribute. In addition, this value is also stored in the PROGRAM value in the V\$SESSION table.
		Microsoft SQL Server	For Microsoft SQL Server 2000 and higher, program_name value in the sysprocesses table. For Microsoft SQL Server 7, local cache.
		Sybase	clientapplname and program_name value in sysprocesses table.

**Table C-1. Database Locations for Storing Client Information (cont.)**

<b>Property</b>	<b>Description</b>	<b>Database</b>	<b>Location</b>
ClientHostName	Host name of the client on which the application using the connection is running	DB2	CURRENT CLIENT_WRKSTNNNAME register (DB2 for Linux/UNIX/Windows) or CLIENT WRKSTNNNAME register (DB2 for z/OS and DB2 for iSeries).
		Informix	Local cache.
		MySQL	Local cache.
		Oracle	MACHINE value in the V\$SESSION table.
		Microsoft SQL Server	For Microsoft SQL Server 2000 and higher, hostname value in the sysprocesses table. For Microsoft SQL Server 7, local cache.
		Sybase	clienthostname and hostname value in sysprocesses table.
ClientUser	User ID for whom the application using the connection is performing work	DB2	CURRENT CLIENT_USERID register (DB2 for Linux/UNIX/Windows) or CLIENT USERID register (DB2 for z/OS and DB2 for iSeries).
		Informix	Local cache.
		MySQL	Local cache.
		Oracle	OSUSER value in the V\$SESSION table.
		Microsoft SQL Server	Local cache.
		Sybase	clientname value in sysprocesses table.

**Table C-1. Database Locations for Storing Client Information (cont.)**

<b>Property</b>	<b>Description</b>	<b>Database</b>	<b>Location</b>
ProgramID	Product name and version of the driver on the client	DB2	CLIENT_PR DID value. For DB2 V9.1 and higher for Linux/UNIX/Windows, the CLIENT_PR DID value is located in the SYSIBMADM.APPLICATIONS table.
		Informix	Local cache.
		MySQL	Local cache.
		Oracle	PROCESS value in the V\$SESSION table.
		Microsoft SQL Server	For Microsoft SQL Server 2000 and higher, hostprocess value in the sysprocesses table. For Microsoft SQL Server 7, local cache.
		Sybase	hostprocess value in the sysprocesses table.

## Returning Client Information

Your application can return client information in the following ways:

- Using the following JDBC methods:
  - `Connection.getClientInfo()`
  - `Connection.getClientInfo(property_name)`
  - `DatabaseMetaData.getClientInfoProperties()`
 See “[Connection Object](#)” on page 552 for more information about these JDBC methods.
- Using the JDBC extension methods provided in the `com.ddtek.jdbc.extensions` package. See [Appendix B “JDBC](#)

[Extensions](#) on page 601 for more information about the com.ddtek.jdbc.extensions package.

---

## Returning MetaData About Client Information Locations

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

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

```
// Get Database Connection
Connection con = DriverManager.getConnection
  ("jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc", "test", "secret");
DatabaseMetaData metaData = con.getMetaData();
metaData.getClientInfoProperties();
...
...
```

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

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

---

## DB2 Workload Manager (WLM) Attributes

The Workload Manager (WLM) is a priority and resource manager within DB2 V9.5 and V9.7 for Linux/UNIX/Windows. On z/OS, the WLM is part of the operating system. When you set client information using the drivers, the WLM can access and use that information for workload management and monitoring purposes as described in ["Workload Manager \(WLM\)" on page 160](#).

## DB2 V9.5 and V9.7 for Linux/UNIX/Windows

[Table C-2](#) lists the WLM attributes for DB2 V9.5 and V9.7 for Linux/UNIX/Windows that map to information set by driver properties. Refer to your DB2 documentation for information about using these WLM attributes.

---

**Table C-2. WLM Attributes for DB2 V9.5 and V9.7 for Linux/UNIX/Windows**

---

WLM Attribute	Driver Property	Description
APPLNAME	ApplicationName	Name of the application currently using the connection
CURRENT CLIENT_ACCTNG	AccountingInfo	Additional information that may be used for accounting or troubleshooting purposes, such as an accounting ID
CLIENT_PRDID	ProgramID	Product name and version of the driver on the client
CURRENT CLIENT_USERID	ClientUser	User ID for whom the application using the connection is performing work.
CURRENT CLIENT_WRKSTNNNAME	ClientHostName	Host name of the client on which the application using the connection is running

---

## DB2 for z/OS

**Table C-3** lists the WLM attributes for DB2 for z/OS that map to information set by driver properties. Refer to your DB2 documentation for information about using these WLM attributes.

---

**Table C-3. WLM Attributes for DB2 z/OS**

---

<b>WLM Attribute</b>	<b>Driver Property</b>	<b>Description</b>
Accounting Info (AI)	AccountingInfo	Additional information that may be used for accounting or troubleshooting purposes, such as an accounting ID
Correlation Info (CI)	ProgramID	Product name and version of the driver on the client
Collection Name (CN)	PackageCollection	Name of the collection or library (group of packages) to which DB2 packages are bound
Process Name (PC)	ApplicationName	Name of the application currently using the connection
Userid (UI)	ClientUser	User ID for whom the application using the connection is performing work.

---



# D `getTypeInfo`

The following tables provide results returned from the `DataBaseMetaData.getTypeInfo` method for all DataDirect Connect for JDBC drivers. The `getTypeInfo()` method returns information about data types supported by a particular database. These tables are organized by driver, and within each table, the results are organized alphabetically for each `TYPE_NAME` column.

## DB2 Driver

[Table D-1](#) provides `getTypeInfo` results for all DB2 databases supported by the DB2 driver (see [Chapter 3 “The DB2 Driver” on page 85](#)).

**Table D-1. `getTypeInfo` for DB2**

<code>TYPE_NAME</code> = <code>bigint</code> *	
<code>AUTO_INCREMENT</code> = false	
<code>CASE_SENSITIVE</code> = false	
<code>CREATE_PARAMS</code> = NULL	
<code>DATA_TYPE</code> = -5 (BIGINT)	
<code>FIXED_PREC_SCALE</code> = false	
<code>LITERAL_PREFIX</code> = NULL	
<code>LITERAL_SUFFIX</code> = NULL	
<code>LOCAL_TYPE_NAME</code> = <code>bigint</code>	
<code>MAXIMUM_SCALE</code> = 0	

<code>MINIMUM_SCALE</code> = 0	
<code>NULLABLE</code> = 1	
<code>NUM_PREC_RADIX</code> = NULL	
<code>PRECISION</code> = 19	
<code>SEARCHABLE</code> = 2	
<code>SQL_DATA_TYPE</code> = NULL	
<code>SQL_DATETIME_SUB</code> = NULL	
<code>UNSIGNED_ATTRIBUTE</code> = false	

\* Supported only for DB2 v9.1 for z/OS.

**Table D-1.** *getTypeInfo* for DB2 (cont.)**TYPE\_NAME = binary \***

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION = 255
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = BINARY(X'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ')	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = binary	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for DB2 v9.1 for z/OS

**TYPE\_NAME = blob \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = ( <i>length</i> )	NUM_PREC_RADIX = NULL
DATA_TYPE = 2004 (BLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = BLOB	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for DB2 v8.1 and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.

**TYPE\_NAME = char**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = ( <i>length</i> )	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION =
FIXED_PREC_SCALE = false	254 (DB2 for Linux/UNIX/Windows),
LITERAL_PREFIX = '	255 (DB2 for z/OS),
LITERAL_SUFFIX = '	32765 (DB2 for iSeries)
LOCAL_TYPE_NAME = char	SEARCHABLE = 3
MAXIMUM_SCALE = NULL	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

---

**Table D-1. *getTypeInfo* for DB2 (cont.)**

---

**TYPE\_NAME = char() for bit data**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = ( <i>length</i> )	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION =
FIXED_PREC_SCALE = false	254 (DB2 for Linux/UNIX/Windows),
LITERAL_PREFIX = X'	254 (DB2 for z/OS),
LITERAL_SUFFIX = '	32765 (DB2 for iSeries)
LOCAL_TYPE_NAME = char() for bit data	SEARCHABLE = 3
MAXIMUM_SCALE = NULL	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

---

**TYPE\_NAME = clob**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = ( <i>length</i> )	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = clob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = date**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {d '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**Table D-1.** `getTypeInfo` for DB2 (cont.)**TYPE\_NAME = dbblob \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS =	NUM_PREC_RADIX = NULL
$(length)$ (DB2 for Linux/UNIX/Windows and DB2 for z/OS),	PRECISION = 2147483647
$(length)$ CCSID 13488 (DB2 V5R2 and higher for iSeries)	SEARCHABLE = 1
DATA_TYPE = 2005 (CLOB) **	SQL_DATA_TYPE = NULL
FIXED_PREC_SCALE = false	SQL_DATETIME_SUB = NULL
LITERAL_PREFIX = '	UNSIGNED_ATTRIBUTE = NULL
LITERAL_SUFFIX = '	
LOCAL_TYPE_NAME = dbblob	
MAXIMUM_SCALE = NULL	

\* Supported only for DB2 v8.1 and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.

\*\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application:  
2011 (NCLOB) (if using Java SE 6) or 2005 (CLOB) (if using another JVM).

**TYPE\_NAME = decfloat \***

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 34
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = NULL	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

\* Supported only for DB2 V9.5 and V9.7 for Linux/UNIX/Windows, DB2 v9.1 for z/OS, and DB2 V6R1 for iSeries.

---

**Table D-1. *getTypeInfo* for DB2 (cont.)**

---

**TYPE\_NAME = decimal**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = ( <i>precision,scale</i> )	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 31
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = decimal	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 31	

---

**TYPE\_NAME = double**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 8 (DOUBLE)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = double	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = float**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 6 (FLOAT)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

**Table D-1.** *getTypeInfo* for DB2 (cont.)**TYPE\_NAME = graphic**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR) *	PRECISION =
FIXED_PREC_SCALE = false	127 (DB2 for Linux/UNIX/Windows),
LITERAL_PREFIX = G'	127 (DB2 for z/OS),
LITERAL_SUFFIX = '	16352 (DB2 for iSeries)
LOCAL_TYPE_NAME = char	SEARCHABLE = 3
MAXIMUM_SCALE = NULL	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application:  
15 (NCHAR) (if using Java SE 6) or 1 (CHAR) (if using another JVM).

**TYPE\_NAME = integer**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = integer	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

**TYPE\_NAME = long varchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION =
FIXED_PREC_SCALE = false	32700 (DB2 for Linux/UNIX/Windows),
LITERAL_PREFIX = ''	32704 (DB2 for z/OS),
LITERAL_SUFFIX = ''	32700 (DB2 for iSeries)
LOCAL_TYPE_NAME = long varchar	SEARCHABLE = 1
MAXIMUM_SCALE = NULL	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

---

**Table D-1. *getTypeInfo* for DB2 (cont.)**

---

**TYPE\_NAME = long varchar for bit data**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION =
FIXED_PREC_SCALE = false	32700 (DB2 for Linux/UNIX/Windows),
LITERAL_PREFIX = X'	32698 (DB2 for z/OS),
LITERAL_SUFFIX = '	32739 (DB2 for iSeries)
LOCAL_TYPE_NAME = long varchar for bit data	SEARCHABLE = 1
MAXIMUM_SCALE = NULL	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

---

**TYPE\_NAME = long vargraphic**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR) *	PRECISION = 16352
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = G'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = longvarchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application:  
15 (LONGNVARCHAR) (if using Java SE 6) or -1 (LONGVARCHAR) (if using another JVM).

---

**TYPE\_NAME = numeric**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = ( <i>precision,scale</i> )	NUM_PREC_RADIX = 10
DATA_TYPE = 2 (NUMERIC)	PRECISION = 31
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = numeric	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 31	

---

**Table D-1.** *getTypeInfo* for DB2 (cont.)**TYPE\_NAME = real**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float(4)	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

**TYPE\_NAME = rowid \***

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = not null generated always	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (Binary)	PRECISION = 40
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = rowid	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = NULL	

\* Supported only for DB2 for z/OS and DB2 V5R2 and higher for iSeries.

**TYPE\_NAME = smallint**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 5 (SMALLINT)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**Table D-1. *getTypeInfo* for DB2 (cont.)**

---

**TYPE\_NAME = time**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 92 (TIME)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {t '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = time	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = timestamp**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 6
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 26
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 6	

---

**TYPE\_NAME = varbinary\***

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION = 32703
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = VARBINARY(X'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ')	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varbinary	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

\* Supported only for DB2 v9.1 for z/OS

**Table D-1.** `getTypeInfo` for DB2 (cont.)**TYPE\_NAME = varchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = ( <i>max length</i> )	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION =
FIXED_PREC_SCALE = false	32704 (DB2 v7.x for Linux/UNIX/Windows),
LITERAL_PREFIX = '	32762 (DB2 v8.x and higher for Linux/UNIX/Windows),
LITERAL_SUFFIX = '	32698 (DB2 for z/OS),
LOCAL_TYPE_NAME = varchar	32739 (DB2 for iSeries)
MAXIMUM_SCALE = NULL	SEARCHABLE =
	3 (DB2 for Linux/UNIX/Windows),
	1 (DB2 for z/OS),
	1 (DB2 for iSeries)
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

**TYPE\_NAME = varchar() for bit data**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = ( <i>max length</i> )	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION =
FIXED_PREC_SCALE = false	32704 (DB2 v7.x for Linux/UNIX/Windows),
LITERAL_PREFIX = X'	32762 (DB2 v8.x and higher for Linux/UNIX/Windows),
LITERAL_SUFFIX = '	32698 (DB2 for z/OS),
LOCAL_TYPE_NAME = varchar() for bit data	32739 (DB2 for iSeries)
MAXIMUM_SCALE = NULL	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

---

**Table D-1. *getTypeInfo* for DB2 (cont.)**

---

**TYPE\_NAME = vargraphic**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR) *	PRECISION = 16352
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = G'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application:  
-9 (NVARCHAR) (if using Java SE 6) or 12 (VARCHAR) (if using another JVM).

---

**TYPE\_NAME = xml\***

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB) **	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX =NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = xml	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for DB2 V9.1 and higher for Linux/UNIX/Windows and DB2 v9.1 for z/OS.

\*\*If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application:  
2008 (SQLXML) (if using Java SE 6) or 2005 (CLOB) (if using another JVM). In addition, the XMLDescribeType property can override driver mappings.

---

---

## Informix Driver

[Table D-2](#) provides `getTypeInfo` results for all Informix databases supported by the Informix driver (see [Chapter 4 “The Informix Driver” on page 177](#)).

---

**Table D-2. `getTypeInfo` for Informix**

---

**TYPE\_NAME = blob**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2004 (BLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = blob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = boolean**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -7 (BIT)	PRECISION = 1
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = boolean	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = byte**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = byte	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**Table D-2.** *getTypeInfo* for Informix (cont.)

---

**TYPE\_NAME = char**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 32766
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = char	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = clob**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = clob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = date**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {d '}	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**Table D-2.** *getTypeInfo* for Informix (cont.)**TYPE\_NAME = datetime hour to second**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 92 (TIME)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {t '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime hour to second	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

**TYPE\_NAME = datetime year to day**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {d '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime year to day	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**TYPE\_NAME = datetime year to fraction(5)**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 5
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 25
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime year to	UNSIGNED_ATTRIBUTE = NULL
fraction(5)	
MAXIMUM_SCALE = 5	

---

**Table D-2.** *getTypeInfo* for Informix (cont.)

---

**TYPE\_NAME = datetime year to second**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime year to second	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = decimal**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision,scale</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 32
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = decimal	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 32	

---

**TYPE\_NAME = float**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 6 (FLOAT)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

**Table D-2.** *getTypeInfo* for Informix (cont.)**TYPE\_NAME = int8**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = int8	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

**TYPE\_NAME = integer**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = integer	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

**TYPE\_NAME = lvarchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS =	NUM_PREC_RADIX = NULL
NULL (Informix 9.2, 9.3), <i>max length</i> (Informix 9.4, 10)	PRECISION = 2048 (Informix 9.2, 9.3), 32739 (Informix 9.4, 10)
DATA_TYPE = 12 (VARCHAR)	SEARCHABLE = 3
FIXED_PREC_SCALE = false	SQL_DATA_TYPE = NULL
LITERAL_PREFIX = '	SQL_DATETIME_SUB = NULL
LITERAL_SUFFIX = '	UNSIGNED_ATTRIBUTE = NULL
LOCAL_TYPE_NAME = lvarchar	
MAXIMUM_SCALE = NULL	

---

**Table D-2. *getTypeInfo* for Informix (cont.)**

---

**TYPE\_NAME = money**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision,scale</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 32
FIXED_PREC_SCALE = true	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = money	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 32	

---

**TYPE\_NAME = nchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR) *	PRECISION = 32766
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: -15 (NCHAR) (if using Java SE 6) or 1 (CHAR) (if using another JVM).

---

**TYPE\_NAME = nvarchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>max length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)*	PRECISION = 254
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nvarchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: -9 (NVARCHAR) (if using Java SE 6) or 12 (VARCHAR) (if using another JVM).

---

**Table D-2.** *getTypeInfo* for Informix (cont.)

---

**TYPE\_NAME = serial**

AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = start	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = serial	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = serial8**

AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = serial8	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = smallfloat**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallfloat	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

---

**Table D-2.** *getTypeInfo* for Informix (cont.)

---

**TYPE\_NAME = smallint**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 5 (SMALLINT)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = text**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = text	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = varchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>max length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 254
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

---

# MySQL Driver

[Table D-3](#) provides `getTypeInfo()` results for MySQL 5.0.x and 5.1 (see [Chapter 5 “The MySQL Driver” on page 217](#)).

---

**Table D-3. `getTypeInfo` for MySQL**

---

**TYPE\_NAME = bigint**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = BIGINT	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = bigint unsigned**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 20
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = BIGINT UNSIGNED	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = binary**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION = 255
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = BINARY	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**Table D-3.** *getTypeInfo* for MySQL (cont.)

---

**TYPE\_NAME = bit**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION = 64
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = b'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = BIT	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = blob**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 65535
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = BLOB	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = char**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 255
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = CHAR	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**Table D-3.** *getTypeInfo* for MySQL (cont.)**TYPE\_NAME = date**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = DATE	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**TYPE\_NAME = datetime**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = DATETIME	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

**TYPE\_NAME = decimal**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision,scale</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 65
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = DECIMAL	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 30	

**Table D-3.** *getTypeInfo* for MySQL (cont.)**TYPE\_NAME = decimal unsigned**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision,scale</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 65
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = DECIMAL UNSIGNED	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 30	

**TYPE\_NAME = double**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 8 (DOUBLE)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = DOUBLE	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

**TYPE\_NAME = double unsigned**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 8 (DOUBLE)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = DOUBLE UNSIGNED	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = NULL	

---

**Table D-3.** *getTypeInfo* for MySQL (cont.)

---

**TYPE\_NAME = float**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = FLOAT	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = float unsigned**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = FLOAT UNSIGNED	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = integer**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = INTEGER	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**Table D-3.** *getTypeInfo* for MySQL (cont.)

---

**TYPE\_NAME = integer unsigned**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = INTEGER UNSIGNED	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = longblob**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = LONGBLOB	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = longtext**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = LONGTEXT	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**Table D-3.** *getTypeInfo* for MySQL (cont.)

---

**TYPE\_NAME = mediumblob**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 16777215
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = MEDIUMBLOB	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = mediumint**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = MEDIUMINT	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = mediumint unsigned**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = MEDIUMINT UNSIGNED	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

**Table D-3.** *getTypeInfo* for MySQL (cont.)**TYPE\_NAME = mediumtext**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 16777215
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = MEDIUMTEXT	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**TYPE\_NAME = smallint**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 5 (SMALLINT)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = SMALLINT	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

**TYPE\_NAME = smallint unsigned**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 5 (SMALLINT)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = SMALLINT UNSIGNED	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

---

**Table D-3.** *getTypeInfo* for MySQL (cont.)

---

**TYPE\_NAME = text**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 65535
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = TEXT	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = time**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 92 (TIME)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = TIME	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = timestamp**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = TIMESTAMP	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

---

**Table D-3.** *getTypeInfo* for MySQL (cont.)

---

**TYPE\_NAME = tinyblob**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 255
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = TINYBLOB	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = tinyint**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = -6 (TINYINT)	PRECISION = 3
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = TINYINT	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = tinyint unsigned**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = -6 (TINYINT)	PRECISION = 3
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = TINYINT UNSIGNED	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

---

**Table D-3.** *getTypeInfo* for MySQL (cont.)

---

**TYPE\_NAME = tinytext**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 255
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = TINYTEXT	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = varbinary**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION = 255
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = VARBINARY	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = varchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 255
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = VARCHAR	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**Table D-3.** *getTypeInfo* for MySQL (cont.)

---

**TYPE\_NAME = year**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 5 (SMALLINT)	PRECISION = 4
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = YEAR	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

---

---

## Oracle Driver

[Table D-4](#) provides `getTypeInfo` results for all Oracle databases supported by the Oracle driver (see [Chapter 6 “The Oracle Driver” on page 265](#)).

---

**Table D-4.** `getTypeInfo` for Oracle

---

**TYPE\_NAME = bfile**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2004 (BLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bfile	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = binary\_float \***

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = binary_float	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

\* Supported only for Oracle 10g and higher.

---

**Table D-4. *getTypeInfo* for Oracle (cont.)**

---

**TYPE\_NAME = binary\_double \***

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 8 (DOUBLE)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = binary_double	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

\* Supported only for Oracle 10g and higher.

---

**TYPE\_NAME = blob**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2004 (BLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = blob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**TYPE\_NAME = char**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 2000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = char	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**Table D-4.** *getTypeInfo* for Oracle (cont.)**TYPE\_NAME = clob**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = clob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**TYPE\_NAME = date**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

**TYPE\_NAME = long**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = long	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**Table D-4.** *getTypeInfo* for Oracle (cont.)**TYPE\_NAME = long raw**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = long raw	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**TYPE\_NAME = nchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)*	PRECISION = 32766
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = N'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: -15 (NCHAR) (if using Java SE 6) or 1 (CHAR) (if using another JVM).

**TYPE\_NAME = nclob**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB) *	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = N'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nclob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: 2001 (NCLOB) (if using Java SE 6) or 2005 (CLOB) (if using another JVM).

**Table D-4.** *getTypeInfo* for Oracle (cont.)**TYPE\_NAME = number**

AUTO_INCREMENT = false	MINIMUM_SCALE = -84
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision,scale</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = number	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 127	

**TYPE\_NAME = number**

AUTO_INCREMENT = false	MINIMUM_SCALE = -84
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = number	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 127	

**TYPE\_NAME = nvarchar2**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>max length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR) *	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = N'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nvarchar2	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: -9 (NVARCHAR) (if using Java SE 6) or 12 (VARCHAR) (if using another JVM).

---

**Table D-4. *getTypeInfo* for Oracle (cont.)**

---

**TYPE\_NAME = raw**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>max length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION = 2000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = raw	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = timestamp \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>fractional_seconds_precision</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 9	

---

\* Supported only for Oracle 9*i* and higher.

**TYPE\_NAME = timestamp with local time****zone \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>fractional_seconds_precision</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp with local	UNSIGNED_ATTRIBUTE = NULL
time zone	
MAXIMUM_SCALE = 9	

---

\* Supported only for Oracle 9*i* and higher.

**Table D-4.** *getTypeInfo* for Oracle (cont.)**TYPE\_NAME = timestamp with time zone \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>fractional_seconds_precision</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp with time zone	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 9	

\* Supported only for Oracle 9*i* and higher.

**TYPE\_NAME = urowid\***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>max_length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ''	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ''	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = urowid	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for Oracle 9*i* and higher.

**TYPE\_NAME = varchar2**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>max_length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ''	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ''	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar2	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**Table D-4. *getTypeInfo* for Oracle (cont.)**

---

**TYPE\_NAME = xmlobj\***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)**	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = xmlobj('	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ')	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = xmlobj	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for Oracle 9*i* (R2) and higher.

\*\*If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: 2009 (SQLXML) (if using Java SE 6) or 2005 (CLOB) (if using another JVM).

---

---

## SQL Server Driver

[Table D-5](#) provides *getTypeInfo* results for all Microsoft SQL Server databases supported by the SQL Server driver (see [Chapter 7 “The Microsoft SQL Server Driver” on page 373](#)).

---

**Table D-5.** *getTypeInfo* for SQL Server

---

**TYPE\_NAME = bigint \***

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

\* Supported only for Microsoft SQL Server 2000 and higher.

---

**TYPE\_NAME = bigint identity \***

AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bigint identity	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

\* Supported only for Microsoft SQL Server 2000 and higher.

---

**Table D-5. *getTypeInfo* for SQL Server (cont.)**

---

**TYPE\_NAME = binary**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = binary	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = bit**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -7 (BIT)	PRECISION = 1
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bit	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = char**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = char	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

---

**Table D-5.** *getTypeInfo* for SQL Server (cont.)

---

**TYPE\_NAME = date**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = datetime**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 3
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 23
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 3	

---

**TYPE\_NAME = datetime2**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 27
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime2	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

**Table D-5.** *getTypeInfo* for SQL Server (cont.)**TYPE\_NAME = datetimeoffset**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 34
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetimeoffset	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

**TYPE\_NAME = decimal**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision,scale</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION =
FIXED_PREC_SCALE = false	28 (SQL Server 7), 38 (SQL Server 2000 and higher) *
LITERAL_PREFIX = NULL	SEARCHABLE = 2
LITERAL_SUFFIX = NULL	SQL_DATA_TYPE = NULL
LOCAL_TYPE_NAME = decimal	SQL_DATETIME_SUB = NULL
MAXIMUM_SCALE =	UNSIGNED_ATTRIBUTE = false
28 (SQL Server 7),	
38 (SQL Server 2000 and higher) *	

\* Configurable server option for Microsoft SQL Server 2000 and higher.

**TYPE\_NAME = decimal() identity**

AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION =
FIXED_PREC_SCALE = false	28 (SQL Server 7), 38 (SQL Server 2000 and higher)
LITERAL_PREFIX = NULL	SEARCHABLE = 2
LITERAL_SUFFIX = NULL	SQL_DATA_TYPE = NULL
LOCAL_TYPE_NAME = decimal() identity	SQL_DATETIME_SUB = NULL
MAXIMUM_SCALE = 0	UNSIGNED_ATTRIBUTE = false

---

**Table D-5.** *getTypeInfo* for SQL Server (cont.)

---

**TYPE\_NAME = float**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 2
DATA_TYPE = 6 (FLOAT)	PRECISION = 53
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = image**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = image	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = int**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = int	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

**Table D-5.** *getTypeInfo* for SQL Server (cont.)**TYPE\_NAME = int identity**

AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = int identity	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

**TYPE\_NAME = money**

AUTO_INCREMENT = false	MINIMUM_SCALE = 4
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 19
FIXED_PREC_SCALE = true	SEARCHABLE = 2
LITERAL_PREFIX = \$	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = money	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 4	

**TYPE\_NAME = nchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR) *	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = N'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: -15 (NCHAR) (if using Java SE 6) or 1 (CHAR) (if using another JVM).

**Table D-5.** *getTypeInfo* for SQL Server (cont.)**TYPE\_NAME = ntext**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR) *	PRECISION = 1073741823
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = N'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = ntext	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: -16 (LONGNVARCHAR) (if using Java SE 6) or -1 (LONGVARCHAR) (if using another JVM).

**TYPE\_NAME = numeric**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision,scale</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 2 (NUMERIC)	PRECISION =
FIXED_PREC_SCALE = false	28 (SQL Server 7), 38 (SQL Server 2000 and higher) *
LITERAL_PREFIX = NULL	SEARCHABLE = 2
LITERAL_SUFFIX = NULL	SQL_DATA_TYPE = NULL
LOCAL_TYPE_NAME = numeric	SQL_DATETIME_SUB = NULL
MAXIMUM_SCALE =	UNSIGNED_ATTRIBUTE = false
28 (SQL Server 7), 38 (SQL Server 2000 and higher) *	

\* Configurable server option for Microsoft SQL Server 2000 and higher.

**TYPE\_NAME = numeric() identity**

AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = <i>precision</i>	NUM_PREC_RADIX = 10
DATA_TYPE = 2 (NUMERIC)	PRECISION =
FIXED_PREC_SCALE = false	28 (SQL Server 7.0), 38 (SQL Server 2000 and higher)
LITERAL_PREFIX = NULL	SEARCHABLE = 2
LITERAL_SUFFIX = NULL	SQL_DATA_TYPE = NULL
LOCAL_TYPE_NAME = numeric() identity	SQL_DATETIME_SUB = NULL
MAXIMUM_SCALE = 0	UNSIGNED_ATTRIBUTE = false

**Table D-5.** *getTypeInfo* for SQL Server (cont.)**TYPE\_NAME = nvarchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>max length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR) *	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = N'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nvarchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: -9 (NVARCHAR) (if using Java SE 6) or 12 (VARCHAR) (if using another JVM).

**TYPE\_NAME = nvarchar(max) \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR) *	PRECISION = 1073741823
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = N'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nvarchar(max)	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for Microsoft SQL Server 2005 and higher.

\*\*If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: -16 (LONGNVARCHAR) (if using Java SE 6) or -1 (LONGVARCHAR) (if using another JVM).

**TYPE\_NAME = real**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 2
DATA_TYPE = 7 (REAL)	PRECISION = 24
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = real	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

---

**Table D-5.** *getTypeInfo* for SQL Server (cont.)

---

**TYPE\_NAME = smalldatetime**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 16
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smalldatetime	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = smallint**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 5 (SMALLINT)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = smallint identity**

AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 5 (SMALLINT)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallint identity	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**Table D-5. *getTypeInfo* for SQL Server (cont.)**

---

**TYPE\_NAME = smallmoney**

AUTO_INCREMENT = false	MINIMUM_SCALE = 4
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 10
FIXED_PREC_SCALE = true	SEARCHABLE = 2
LITERAL_PREFIX = \$	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallmoney	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 4	

---

**TYPE\_NAME = sql\_variant \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 12 (VARCHAR)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = sql_variant	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

---

\* Supported only for Microsoft SQL Server 2000 and higher.

---

**TYPE\_NAME = sysname**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 128
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = N'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = sysname	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

---

**Table D-5. *getTypeInfo* for SQL Server (cont.)**

---

**TYPE\_NAME = text**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = text	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = time**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 16
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = time	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = timestamp**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**Table D-5. *getTypeInfo* for SQL Server (cont.)**

---

**TYPE\_NAME = tinyint**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -6 (TINYINT)	PRECISION = 3
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = tinyint	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = tinyint identity**

AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -6 (TINYINT)	PRECISION = 3
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = tinyint identity	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = uniqueidentifier**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 1(CHAR)	PRECISION = 36
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = uniqueidentifier	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**Table D-5.** *getTypeInfo* for SQL Server (cont.)**TYPE\_NAME = varbinary**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>max length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varbinary	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**TYPE\_NAME = varbinary(max) \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varbinary(max)	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for Microsoft SQL Server 2005 and higher.

**TYPE\_NAME = varchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>max length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**Table D-5. *getTypeInfo* for SQL Server (cont.)**

---

**TYPE\_NAME = varchar(max) \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar(max)	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for Microsoft SQL Server 2005 and higher.

---

**TYPE\_NAME = xml \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR) **	PRECISION = 1073741823
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = N'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = xml	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for Microsoft SQL Server 2005 and higher.

\*\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: 2009 (SQLXML) (if using Java SE 6) or -1 (LONGVARCHAR) (if using another JVM). In addition, the XMLDescribeType property can override driver mappings.

---

---

## Sybase Driver

[Table D-6](#) provides `getTypeInfo` results for all Sybase databases supported by the Sybase driver (see [Chapter 8 “The Sybase Driver” on page 471](#)).

---

**Table D-6. `getTypeInfo` for Sybase**

---

**TYPE\_NAME = bigint \***

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

\* Supported only for Sybase 15.

---

**TYPE\_NAME = binary**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION =
FIXED_PREC_SCALE = false	255 (Sybase 11.x, 12.0), *
LITERAL_PREFIX = 0x	2048 (Sybase 12.5 and higher) *
LITERAL_SUFFIX = NULL	SEARCHABLE = 2
LOCAL_TYPE_NAME = binary	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE = NULL	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

\* For Sybase 12.5.1 and higher, precision is determined by the server page size.

---

**Table D-6. *getTypeInfo* for Sybase (cont.)**

---

**TYPE\_NAME = bit**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -7 (BIT)	PRECISION = 1
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bit	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = char**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION =
FIXED_PREC_SCALE = false	255 (Sybase 11.x, 12.0), *
LITERAL_PREFIX = '	2048 (Sybase 12.5 and higher) *
LITERAL_SUFFIX = '	SEARCHABLE = 3
LOCAL_TYPE_NAME = char	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE = NULL	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

---

\* For Sybase 12.5.1 and higher, precision is determined by the server page size.

**TYPE\_NAME = date \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

\* Supported only for Sybase 12.5.1 and higher.

**Table D-6.** *getTypeInfo* for Sybase (cont.)**TYPE\_NAME = datetime**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 3
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 23
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 3	

**TYPE\_NAME = decimal**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision,scale</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 3 (DECIMAL)	PRECISION = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = decimal	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 38	

**TYPE\_NAME = float**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 6 (FLOAT)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

---

**Table D-6. *getTypeInfo* for Sybase (cont.)**

---

**TYPE\_NAME = image**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = image	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**TYPE\_NAME = int**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = int	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = money**

AUTO_INCREMENT = false	MINIMUM_SCALE = 4
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 3 (DECIMAL)	PRECISION = 19
FIXED_PREC_SCALE = true	SEARCHABLE = 2
LITERAL_PREFIX = \$	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = money	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 4	

**Table D-6.** *getTypeInfo* for Sybase (cont.)**TYPE\_NAME = numeric**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>precision,scale</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 2 (NUMERIC)	PRECISION = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = numeric	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 38	

**TYPE\_NAME = real**

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = real	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

**TYPE\_NAME = smalldatetime**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 3
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 16
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smalldatetime	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 3	

---

**Table D-6. *getTypeInfo* for Sybase (cont.)**

---

**TYPE\_NAME = smallint**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 5 (SMALLINT)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = smallmoney**

AUTO_INCREMENT = false	MINIMUM_SCALE = 4
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 3 (DECIMAL)	PRECISION = 10
FIXED_PREC_SCALE = true	SEARCHABLE = 2
LITERAL_PREFIX = \$	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallmoney	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 4	

---

**TYPE\_NAME = sysname**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>max length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 30
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = sysname	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**Table D-6.** *getTypeInfo* for Sybase (cont.)**TYPE\_NAME = text**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = text	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

**TYPE\_NAME = time \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 3
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 92 (TIME)	PRECISION = 12
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = time	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 3	

\* Supported only for Sybase 12.5.1 and higher.

**TYPE\_NAME = timestamp**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX =0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

---

**Table D-6. *getTypeInfo* for Sybase (cont.)**

---

**TYPE\_NAME = tinyint**

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -6 (TINYINT)	PRECISION = 3
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = tinyint	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

---

**TYPE\_NAME = unsigned bigint \***

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 3 (DECIMAL)	PRECISION = 20
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = unsigned bigint	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

---

\* Supported only for Sybase 15.

**TYPE\_NAME = unsigned int \***

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -5 (BIGINT)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = unsigned int	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

---

\* Supported only for Sybase 15.

**Table D-6.** `getTypeInfo` for Sybase (cont.)**TYPE\_NAME = unsigned smallint \***

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 4 (INTEGER)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = unsigned smallint	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

\* Supported only for Sybase 15.

**TYPE\_NAME = unichar \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR) **	PRECISION = 2048
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = unichar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for Sybase 12.5 and higher.

\*\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: -15 (NCHAR) (if using Java SE 6) or 1 (CHAR) (if using another JVM).

**TYPE\_NAME = unitext \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR) **	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = unitext	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for Sybase 15.

\*\* If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: 2011 (LONGNVARCHAR) (if using Java SE 6) or -1 (LONGVARCHAR) (if using another JVM).

---

**Table D-6. *getTypeInfo* for Sybase (cont.)**

---

**TYPE\_NAME = univarchar \***

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>max length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR) **	PRECISION = 2048
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = univarchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

\* Supported only for Sybase 12.5 and higher.

\*\*If JDBCBehavior=0, the value returned for DATA\_TYPE depends on the JVM used by the application: -9 (NVARCHAR) (if using Java SE 6) or 12 (VARCHAR) (if using another JVM).

---

**TYPE\_NAME = varbinary**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = <i>max length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION =
FIXED_PREC_SCALE = false	255 (Sybase 11.x, 12.0), *
LITERAL_PREFIX = 0x	2048 (Sybase 12.5 and higher) *
LITERAL_SUFFIX = NULL	SEARCHABLE = 2
LOCAL_TYPE_NAME = varbinary	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE = NULL	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

\* For Sybase 12.5.1 and higher, precision is determined by the server page size.

---

**TYPE\_NAME = varchar**

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = <i>max length</i>	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION =
FIXED_PREC_SCALE = false	255 (Sybase 11.x, 12.0), *
LITERAL_PREFIX = '	2048 (Sybase 12.5 and higher) *
LITERAL_SUFFIX = '	SEARCHABLE = 3
LOCAL_TYPE_NAME = varchar	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE = NULL	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

\* For Sybase 12.5.1 and higher, precision is determined by the server page size.



# E Designing JDBC Applications for Performance Optimization

Developing performance-oriented JDBC applications is not easy. JDBC drivers do not throw exceptions to tell you when your code is running too slow. This appendix presents some general guidelines for improving JDBC application performance that have been compiled by examining the JDBC implementations of numerous shipping JDBC applications. These guidelines include:

- Use DatabaseMetaData methods appropriately
- Return only required data
- Select functions that optimize performance
- Manage connections and updates

Following these general guidelines can help you solve some common JDBC system performance problems, such as those listed in the following table.

Problem	Solution	See guidelines in...
Network communication is slow.	Reduce network traffic.	<a href="#">"Using Database Metadata Methods" on page 696</a>
Evaluation of complex SQL queries on the database server is slow and can reduce concurrency.	Simplify queries.	<a href="#">"Using Database Metadata Methods" on page 696</a> <a href="#">"Selecting JDBC Objects and Methods" on page 703</a>
Excessive calls from the application to the driver slow performance.	Optimize application-to-driver interaction.	<a href="#">"Returning Data" on page 700</a> <a href="#">"Selecting JDBC Objects and Methods" on page 703</a>
Disk I/O is slow.	Limit disk I/O.	<a href="#">"Managing Connections and Updates" on page 711</a>

In addition, most JDBC drivers provide options that improve performance, often with a trade-off in functionality. If your application is not affected by functionality that is modified by setting a particular option, significant performance improvements can be realized.

---

## Using Database Metadata Methods

Because database metadata methods that generate `ResultSet` objects are slow compared to other JDBC methods, their frequent use can impair system performance. The guidelines in this section will help you optimize system performance when selecting and using database metadata.

### Minimizing the Use of Database Metadata Methods

Compared to other JDBC methods, database metadata methods that generate `ResultSet` objects are relatively slow. Applications should cache information returned from result sets that generate database metadata methods so that multiple executions are not needed.

Although almost no JDBC application can be written without database metadata methods, you can improve system performance by minimizing their use. To return all result column information *mandated* by the JDBC specification, a JDBC driver may have to perform complex queries or multiple queries to return the necessary result set for a single call to a database metadata method. These particular elements of the SQL language are performance-expensive.

Applications should cache information from database metadata methods. For example, call `getTypeInfo` once in the application

and cache the elements of the result set that your application depends on. It is unlikely that any application uses all elements of the result set generated by a database metadata method, so the cache of information should not be difficult to maintain.

## Avoiding Search Patterns

Using null arguments or search patterns in database metadata methods results in generating time-consuming queries. In addition, network traffic potentially increases due to unwanted results. Always supply as many non-null arguments as possible to result sets that generate database metadata methods.

Because database metadata methods are slow, invoke them in your applications as efficiently as possible. Many applications pass the fewest non-null arguments necessary for the function to return success. For example:

```
ResultSet WSrs = WSdbmd.getTables (null, null, "WSTable",  
        null);
```

In this example, an application uses the `getTables` method to determine if the `WSTable` table exists. A JDBC driver interprets the request as: return all tables, views, system tables, synonyms, temporary tables, and aliases named "`WSTable`" that exist in any database schema inside the database catalog.

In contrast, the following request provides non-null arguments as shown:

```
String[] tableTypes = {"TABLE"}; WSdbmd.getTables ("cat1",  
        "johng", "WSTable", "tableTypes");
```

Clearly, a JDBC driver can process the second request more efficiently than it can process the first request.

Sometimes, little information is known about the object for which you are requesting information. Any information that the

application can send the driver when calling database metadata methods can result in improved performance and reliability.

## Using a Dummy Query to Determine Table Characteristics

Avoid using the `getColumns` method to determine characteristics about a database table. Instead, use a dummy query with `getMetadata`.

Consider an application that allows the user to choose the columns to be selected. Should the application use `getColumns` to return information about the columns to the user or instead prepare a dummy query and call `getMetadata`?

### ***Case 1: GetColumns Method***

```
ResultSet WSrc = WSc.getColumns (... "UnknownTable" ...);  
// This call to getColumns will generate a query to  
// the system catalogs... possibly a join  
// which must be prepared, executed, and produce  
// a result set  
.  
.  
.  
WSrc.next();  
string Cname = getString(4);  
.  
. .  
// user must return N rows from the server  
// N = # result columns of UnknownTable  
// result column information has now been obtained
```

### ***Case 2: GetMetadata Method***

```
// prepare dummy query  
PreparedStatement WSps = WSc.prepareStatement  
("SELECT * FROM UnknownTable WHERE 1 = 0");
```

```
// query is never executed on the server - only prepared
ResultSetMetaData WSsmd=WSps.getMetaData();
int numcols = WSrsmd.getColumnCount();
...
int ctype = WSrsmd.getColumnType(n)
...
// result column information has now been obtained
// Note we also know the column ordering within the
// table! This information cannot be
// assumed from the getColumns example.
```

In both cases, a query is sent to the server. However, in Case 1, the potentially complex query must be prepared and executed, result description information must be formulated, and a result set of rows must be sent to the client. In Case 2, we prepare a simple query where we only return result set information. Clearly, Case 2 is the better performing model.

To somewhat complicate this discussion, let us consider a DBMS server that does not natively support preparing a SQL statement. The performance of Case 1 does not change but the performance of Case 2 improves slightly because the dummy query must be evaluated in addition to being prepared. Because the Where clause of the query always evaluates to FALSE, the query generates no result rows and should execute without accessing table data. For this situation, Case 2 still outperforms Case 1.

In summary, always use result set metadata to return table column information, such as column names, column data types, and column precision and scale. Only use the `getColumns` method when the requested information cannot be obtained from result set metadata (for example, using the table column default values).

---

# Returning Data

To return data efficiently, return only the data that you need and choose the most efficient method of doing so. The guidelines in this section will help you optimize system performance when retrieving data with JDBC applications.

## Returning Long Data

Because retrieving long data across a network is slow and resource intensive, applications should not request long data unless it is necessary.

Most users do not want to see long data. If the user does want to see these result items, then the application can query the database again, specifying only the long columns in the Select list. This method allows the average user to return the result set without having to pay a high performance penalty for network traffic.

Although the best method is to exclude long data from the Select list, some applications do not formulate the Select list before sending the query to the JDBC driver (that is, some applications SELECT \* FROM *table\_name* ...). If the Select list contains long data, most drivers are forced to return that long data at fetch time, even if the application does not ask for the long data in the result set. When possible, the designer should attempt to implement a method that does not return all columns of the table.

For example, consider the following code:

```
ResultSet rs = stmt.executeQuery (  
    "SELECT * FROM Employees WHERE SSID = '999-99-2222'" );  
rs.next();  
String name = rs.getString(1);
```

Remember that a JDBC driver cannot interpret an application's final intention. When a query is executed, the driver has no way to know which result columns an application will use. A driver anticipates that an application can request any of the result columns that are returned. When the JDBC driver processes the rs.next request, it will probably return at least one, if not more, result rows from the database server across the network. In this case, a result row contains all the column values for each row, including an employee photograph if the Employees table contains such a column. If you limit the Select list to contain only the employee name column, it results in decreased network traffic and a faster performing query at runtime. For example:

```
ResultSet rs = stmt.executeQuery (   
    "SELECT name FROM Employees WHERE SSID = '999-99-2222'" );  
rs.next();  
String name = rs.getString(1);
```

Additionally, although the getClob() and getBlob() methods allow the application to control how long data is returned in the application, the designer must realize that in many cases, the JDBC driver emulates these methods due to the lack of true Large Object (LOB) locator support in the DBMS. In such cases, the driver must return all the long data across the network before exposing the getClob and getBlob methods.

## Reducing the Size of Returned Data

Sometimes long data must be returned. When this is the case, remember that most users do not want to see 100 KB, or more, of text on the screen.

To reduce network traffic and improve performance, you can reduce the size of any data being returned to some manageable limit by calling setMaxRows, setMaxFieldSize, and the driver-specific setFetchSize. Another method of reducing the size of the data being returned is to decrease the column size.

In addition, be careful to return only the rows you need. If you return five columns when you only need two columns, performance is decreased, especially if the unnecessary rows include long data.

## Choosing the Right Data Type

Retrieving and sending certain data types can be expensive. When you design a schema, select the data type that can be processed most efficiently. For example, integer data is processed faster than floating-point data. Floating-point data is defined according to internal database-specific formats, usually in a compressed format. The data must be decompressed and converted into a different format so that it can be processed by the database wire protocol.

## Retrieving Result Sets

Most JDBC drivers cannot implement scrollable cursors because of limited support for scrollable cursors in the database system. Unless you are certain that the database supports using a scrollable result set, `rs`, for example, do not call `rs.last` and `rs.getRow` methods to find out how many rows the result set contains. For JDBC drivers that emulate scrollable cursors, calling `rs.last` results in the driver retrieving all results across the network to reach the last row. Instead, you can either count the rows by iterating through the result set or get the number of rows by submitting a query with a `Count` column in the `Select` clause.

In general, do not write code that relies on the number of result rows from a query because drivers must fetch all rows in a result set to know how many rows the query will return.

---

# Selecting JDBC Objects and Methods

The guidelines in this section will help you to select which JDBC objects and methods will give you the best performance.

## Using Parameter Markers as Arguments to Stored Procedures

When calling stored procedures, always use parameter markers for argument markers instead of using literal arguments. JDBC drivers can call stored procedures on the database server either by executing the procedure as a SQL query or by optimizing the execution by invoking a Remote Procedure Call (RPC) directly on the database server. When you execute a stored procedure as a SQL query, the database server parses the statement, validates the argument types, and converts the arguments into the correct data types.

Remember that SQL is always sent to the database server as a character string, for example, “{call getUserName (12345)}”. In this case, even though the application programmer may have assumed that the only argument to getUserName was an integer, the argument is actually passed inside a character string to the server. The database server parses the SQL query, isolates the single argument value 12345, and then converts the string '12345' into an integer value before executing the procedure as a SQL language event.

By invoking a RPC on the database server, the overhead of using a SQL character string is avoided. Instead, the JDBC driver constructs a network packet that contains the parameters in their native data type formats and executes the procedure remotely.

## ***Case 1: Not Using a Server-Side RPC***

In this example, the stored procedure `getCustName` cannot be optimized to use a server-side RPC. The database server must treat the SQL request as a normal language event, which includes parsing the statement, validating the argument types, and converting the arguments into the correct data types before executing the procedure.

```
CallableStatement cstmt = conn.prepareCall ("call getCustName (12345)");
ResultSet rs = cstmt.executeQuery ();
```

## ***Case 2: Using a Server-Side RPC***

In this example, the stored procedure `getCustName` can be optimized to use a server-side RPC. Because the application avoids literal arguments and calls the procedure by specifying all arguments as parameters, the JDBC driver can optimize the execution by invoking the stored procedure directly on the database as an RPC. The SQL language processing on the database server is avoided and execution time is greatly improved.

```
CallableStatement cstmt = conn.prepareCall ("call getCustName (?)}");
cstmt.setLong (1,12345);
ResultSet rs = cstmt.executeQuery();
```

## **Using the Statement Object Instead of the PreparedStatement Object**

JDBC drivers are optimized based on the perceived use of the functions that are being executed. Choose between the `PreparedStatement` object and the `Statement` object depending on how you plan to use the object. The `Statement` object is optimized for a single execution of a SQL statement. In contrast,

the PreparedStatement object is optimized for SQL statements to be executed two or more times.

The overhead for the initial execution of a PreparedStatement object is high. The advantage comes with subsequent executions of the SQL statement. For example, suppose we are preparing and executing a query that returns employee information based on an ID. Using a PreparedStatement object, a JDBC driver would process the prepare request by making a network request to the database server to parse and optimize the query. The execute results in another network request. If the application will only make this request once during its life span, using a Statement object instead of a PreparedStatement object results in only a single network roundtrip to the database server. Reducing network communication typically provides the most performance gains.

This guideline is complicated by the use of prepared statement pooling because the scope of execution is longer. When using prepared statement pooling, if a query will only be executed once, use the Statement object. If a query will be executed infrequently, but may be executed again during the life of a statement pool inside a connection pool, use a PreparedStatement object. Under similar circumstances without statement pooling, use the Statement object.

## Using Batches Instead of Prepared Statements

Updating large amounts of data typically is done by preparing an Insert statement and executing that statement multiple times, resulting in numerous network roundtrips. To reduce the number of JDBC calls and improve performance, you can send multiple queries to the database at a time using the addBatch method of the PreparedStatement object. For example, let us compare the following examples, Case 1 and Case 2.

## ***Case 1: Executing Prepared Statement Multiple Times***

```
PreparedStatement ps = conn.prepareStatement(
    "INSERT INTO employees VALUES (?, ?, ?)");
for (n = 0; n < 100; n++) {
    ps.setString(name[n]);
    ps.setLong(id[n]);
    ps.setInt(salary[n]);
    ps.executeUpdate();
}
```

## ***Case 2: Using a Batch***

```
PreparedStatement ps = conn.prepareStatement(
    "INSERT INTO employees VALUES (?, ?, ?)");
for (n = 0; n < 100; n++) {
    ps.setString(name[n]);
    ps.setLong(id[n]);
    ps.setInt(salary[n]);
    ps.addBatch();
}
ps.executeBatch();
```

In Case 1, a prepared statement is used to execute an Insert statement multiple times. In this case, 101 network roundtrips are required to perform 100 Insert operations: one roundtrip to prepare the statement and 100 additional roundtrips to execute its iterations. When the addBatch method is used to consolidate 100 Insert operations, as demonstrated in Case 2, only two network roundtrips are required—one to prepare the statement and another to execute the batch. Although more database CPU cycles are involved by using batches, performance is gained through the reduction of network roundtrips. Remember that the biggest gain in performance is realized by reducing network communication between the JDBC driver and the database server.

## Choosing the Right Cursor

Choosing the appropriate type of cursor allows maximum application flexibility. This section summarizes the performance issues of three types of cursors: forward-only, insensitive, and sensitive.

A forward-only cursor provides excellent performance for sequential reads of all rows in a table. For retrieving table data, there is no faster way to return result rows than using a forward-only cursor. However, forward-only cursors cannot be used when the rows to be returned are not sequential.

Insensitive cursors used by JDBC drivers are ideal for applications that require high levels of concurrency on the database server and require the ability to scroll forwards and backwards through result sets. The first request to an insensitive cursor fetches all of the rows and stores them on the client. In most cases, the first request to an insensitive cursor fetches all the rows and stores them on the client. If a driver uses "lazy" fetching (fetch-on-demand) implementation, the first request may include many rows, if not all rows. Therefore, the initial request is very slow, especially when long data is returned. Subsequent requests do not require any network traffic (or, when a driver uses "lazy" fetching, requires limited network traffic) and are processed quickly.

Because the first request is processed slowly, insensitive cursors should not be used for a single request of one row. Developers should also avoid using insensitive cursors when long data or large result sets are returned because memory can be exhausted. Some insensitive cursor implementations cache the data in a temporary table on the database server and avoid the performance issue, but most cache the information local to the application.

Sensitive cursors, sometimes called keyset-driven cursors, use identifiers, such as a ROWID, that already exist in the database. When you scroll through the result set, the data for these

identifiers is returned. Because each request generates network traffic, performance can be very slow. However, returning non-sequential rows does not further affect performance.

To illustrate this point further, consider an application that would normally return 1000 rows to an application. At execute time, or when the first row is requested, a JDBC driver does not execute the Select statement that was provided by the application. Instead, the JDBC driver replaces the Select list of the query with the key identifier, for example, ROWID. This modified query is then executed by the driver and all 1000 key values are returned by the database server and cached for use by the driver. Each request from the application for a result row directs the JDBC driver to look up the key value for the appropriate row in its local cache, construct an optimized query that contains a Where clause similar to WHERE ROWID=? , execute the modified query, and return the single result row from the server.

Sensitive cursors are the preferred scrollable cursor model for dynamic situations when the application cannot afford to buffer the data associated with an insensitive cursor.

## Using get Methods Effectively

JDBC provides a variety of methods to return data from a result set, such as getInt, getString, and getObject. The getObject method is the most generic and provides the worst performance when the non-default mappings are specified. This is because the JDBC driver must perform extra processing to determine the type of the value being returned and generate the appropriate mapping. Always use the specific method for the data type.

To further improve performance, provide the column number of the column being returned, for example, getString(1), getLong(2), and getInt(3), instead of the column name. If the column names are not specified, network traffic is unaffected,

but costly conversions and lookups increase. For example, suppose you use:

```
getString("foo")...
```

The JDBC driver may have to convert foo to uppercase (if necessary), and then compare foo with all the columns in the column list. That is costly. If, instead, the driver was able to go directly to result column 23, then a lot of processing would be saved.

For example, suppose you have a result set that has 15 columns and 100 rows, and the column names are not included in the result set. You are interested in three columns, EMPLOYEEENAME (a string), EMPLOYEENUMBER (a long integer), and SALARY (an integer). If you specify `getString("EmployeeName")`, `getLong("EmployeeNumber")`, and `getInt("Salary")`, each column name must be converted to the appropriate case of the columns in the database metadata and lookups would increase considerably. Performance would improve significantly if you specify `getString(1)`, `getLong(2)`, and `getInt(15)`.

## Retrieving Auto-Generated Keys

Many databases have hidden columns (called pseudo-columns) that represent a unique key over every row in a table. Typically, using these types of columns in a query is the fastest way to access a row because the pseudo-columns usually represent the physical disk address of the data. Prior to JDBC 3.0, an application could only return the value of the pseudo-columns by executing a Select statement immediately after inserting the data. For example:

```
//insert row
int rowcount = stmt.executeUpdate (
    "INSERT INTO LocalGeniusList (name)
     VALUES ('Karen')");
```

```
// now get the disk address - rowid -
// for the newly inserted row
ResultSet rs = stmt.executeQuery (
    "SELECT rowid FROM LocalGeniusList
     WHERE name = 'Karen'");
```

Retrieving pseudo-columns this way has two major flaws. First, retrieving the pseudo-column requires a separate query to be sent over the network and executed on the server. Second, because there may not be a primary key over the table, the search condition of the query may be unable to uniquely identify the row. In the latter case, multiple pseudo-column values can be returned, and the application may not be able to determine which value is actually the value for the most recently inserted row.

An optional feature of the JDBC 3.0 specification is the ability to return auto-generated key information for a row when the row is inserted into a table. For example:

```
int rowCount = stmt.executeUpdate (
    "INSERT INTO LocalGeniusList (name) VALUES ('Karen')",
    // insert row AND return key
    Statement.RETURN_GENERATED_KEYS);
ResultSet rs = stmt.getGeneratedKeys ();
// key is automatically available
```

Now, the application contains a value that can be used in a search condition to provide the fastest access to the row and a value that uniquely identifies the row, even when a primary key doesn't exist on the table.

The ability to return auto-generated keys provides flexibility to the JDBC developer and creates performance boosts when accessing data.

---

# Managing Connections and Updates

The guidelines in this section will help you to manage connections and updates to improve system performance for your JDBC applications.

## Managing Connections

Connection management is important to application performance. Optimize your application by connecting once and using multiple Statement objects, instead of performing multiple connections. Avoid connecting to a data source after establishing an initial connection.

Although gathering driver information at connect time is a good practice, it is often more efficient to gather it in one step rather than two steps. For example, some applications establish a connection and then call a method in a separate component that reattaches and gathers information about the driver. Applications that are designed as separate entities should pass the established connection object to the data collection routine instead of establishing a second connection.

Another bad practice is to connect and disconnect several times throughout your application to perform SQL statements. Connection objects can have multiple Statement objects associated with them. Statement objects, which are defined to be memory storage for information about SQL statements, can manage multiple SQL statements.

You can improve performance significantly with connection pooling, especially for applications that connect over a network or through the World Wide Web. Connection pooling lets you reuse connections. Closing connections does not close the physical connection to the database. When an application requests a connection, an active connection is reused, thus

avoiding the network round trips needed to create a new connection.

Typically, you can configure a connection pool to provide scalability for connections. The goal is to maintain a reasonable connection pool size while ensuring that each user who needs a connection has one available within an acceptable response time. To achieve this goal, you can configure the minimum and maximum number of connections that are in the pool at any given time, and how long idle connections stay in the pool. In addition, to help minimize the number of connections required in a connection pool, you can switch the user associated with a connection to another user, a process known as *reauthentication*. Not all databases support reauthentication.

In addition to connection pooling tuning options, JDBC also specifies semantics for providing a prepared statement pool. Similar to connection pooling, a prepared statement pool caches PreparedStatement objects so that they can be re-used from a cache without application intervention. For example, an application may create a PreparedStatement object similar to the following SQL statement:

```
SELECT name, address, dept, salary FROM personnel  
WHERE empid = ? or name = ? or address = ?
```

When the PreparedStatement object is created, the SQL query is parsed for semantic validation and a query optimization plan is produced. The process of creating a prepared statement is extremely expensive in terms of performance with some database systems such as DB2. Once the prepared statement is closed, a JDBC 3.0-compliant driver places the prepared statement into a local cache instead of discarding it. If the application later attempts to create a prepared statement with the same SQL query, a common occurrence in many applications, the driver can simply retrieve the associated statement from the local cache instead of performing a network roundtrip to the server and an expensive database validation.

Connection and statement handling should be addressed before implementation. Thoughtfully handling connections and statements improves application performance and maintainability.

## Managing Commits in Transactions

Committing transactions is slow because of the amount of disk I/O and potentially network round trips that are required. Always turn off Autocommit by using the `WSConnection.setAutoCommit(false)` setting.

What does a commit actually involve? The database server must flush back to disk every data page that contains updated or new data. This is usually a sequential write to a journal file, but nevertheless, it involves disk I/O. By default, Autocommit is on when connecting to a data source, and Autocommit mode usually impairs performance because of the significant amount of disk I/O needed to commit every operation.

Furthermore, most database servers do not provide a native Autocommit mode. For this type of server, the JDBC driver must explicitly issue a `COMMIT` statement and a `BEGIN TRANSACTION` for every operation sent to the server. In addition to the large amount of disk I/O required to support Autocommit mode, a performance penalty is paid for up to three network requests for every statement issued by an application.

Although using transactions can help application performance, do not take this tip too far. Leaving transactions active can reduce throughput by holding locks on rows for longer than necessary, preventing other users from accessing the rows. Commit transactions in intervals that allow maximum concurrency.

## Choosing the Right Transaction Model

Many systems support distributed transactions; that is, transactions that span multiple connections. Distributed transactions are at least four times slower than normal transactions due to the logging and network round trips necessary to communicate between all the components involved in the distributed transaction (the JDBC driver, the transaction monitor, and the DBMS). Unless distributed transactions are required, avoid using them. Instead, use local transactions when possible. Many Java application servers provide a default transaction behavior that uses distributed transactions.

For the best system performance, design the application to run using a single Connection object.

## Using updateXXX Methods

Although programmatic updates do not apply to all types of applications, developers should attempt to use programmatic updates and deletes. Using the updateXXX methods of the ResultSet object allows the developer to update data without building a complex SQL statement. Instead, the developer simply supplies the column in the result set that is to be updated and the data that is to be changed. Then, before moving the cursor from the row in the result set, the updateRow method must be called to update the database as well.

In the following code fragment, the value of the Age column of the ResultSet object rs is returned using the method getInt(), and the method updateInt() is used to update the column with an int value of 25. The method updateRow is called to update the row in the database contain the modified value.

```
int n = rs.getInt("Age");
// n contains value of Age column in the resultset rs
. . .
```

```
rs.updateInt("Age", 25);
rs.updateRow();
```

In addition to making the application more easily maintainable, programmatic updates usually result in improved performance. Because the database server is already positioned on the row for the Select statement in process, performance-expensive operations to locate the row that needs to be changed are unnecessary. If the row must be located, the server usually has an internal pointer to the row available (for example, ROWID).

## Using getBestRowIdentifier

Use `getBestRowIdentifier` to determine the optimal set of columns to use in the Where clause for updating data. Pseudo-columns often provide the fastest access to the data, and these columns can only be determined by using `getBestRowIdentifier`.

Some applications cannot be designed to take advantage of positioned updates and deletes. Some applications formulate the Where clause by calling `getPrimaryKeys` to use all searchable result columns or by calling `getIndexInfo` to find columns that may be part of a unique index. These methods usually work, but can result in fairly complex queries.

Consider the following example:

```
ResultSet WSrs = WSs.executeQuery
    ("SELECT first_name, last_name, ssn, address, city, state, zip
     FROM emp");
// fetchdata
...
WSs.executeQuery ("UPDATE emp SET address = ?
                  WHERE first_name = ? and last_name = ? and ssn = ?
                  and address = ? and city = ? and state = ? and zip = ?");
// fairly complex query
```

Applications should call `getBestRowIdentifier` to return the optimal set of columns (possibly a pseudo-column) that identifies a specific record. Many databases support special columns that are not explicitly defined by the user in the table definition but are "hidden" columns of every table (for example, ROWID and TID). These pseudo-columns generally provide the fastest access to the data because they typically are pointers to the exact location of the record. Because pseudo-columns are not part of the explicit table definition, they are not returned from `getColumns`. To determine if pseudo-columns exist, call `getBestRowIdentifier`.

Consider the previous example again:

```
...
ResultSet WSrowid = getBestRowIdentifier()
(.... "emp", ...);
...
WSs.executeUpdate ("UPDATE EMP SET ADDRESS = ?
WHERE ROWID = ?");
// fastest access to the data!
```

If your data source does not contain special pseudo-columns, then the result set of `getBestRowIdentifier` consists of the columns of the most optimal unique index on the specified table (if a unique index exists). Therefore, your application does not need to call `getIndexInfo` to find the smallest unique index.

# F SQL Escape Sequences for JDBC

Language features, such as outer joins and scalar function calls, are commonly implemented by database systems. The syntax for these features is often database-specific, even when a standard syntax has been defined. JDBC defines escape sequences that contain the standard syntax for the following language features:

- Date, time, and timestamp literals
- Scalar functions such as numeric, string, and data type conversion functions
- Outer joins
- Escape characters for wildcards used in LIKE clauses
- Procedure calls

The escape sequence used by JDBC is:

{extension}

The escape sequence is recognized and parsed by DataDirect Connect for JDBC drivers, which replaces the escape sequences with data store-specific grammar.

---

## Date, Time, and Timestamp Escape Sequences

The escape sequence for date, time, and timestamp literals is:

```
{literal-type 'value'}
```

where *literal-type* is one of the following:

literal-type	Description	Value Format
d	Date	yyyy-mm-dd
t	Time	hh:mm:ss [ ]
ts	Timestamp	yyyy-mm-dd hh:mm:ss[.f...]

**Example:**

```
UPDATE Orders SET OpenDate={d '1995-01-15'}
WHERE OrderID=1023
```

---

## 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 DataDirect Connect for JDBC drivers, as listed in [Table F-1](#).

**Example:**

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

**Table F-1. Scalar Functions Supported**

Data Store	String Functions	Numeric Functions	Timedate Functions	System Functions
DB2	ASCII	ABS or ABSVAL	CURDATE CURTIME	COALESCE Deref
	BLOB	ACOS	DATE	DLCOMMENT
	CHAR	ASIN	DAY	DLLINKTYPE
	CHR	ATAN	DAYNAME	DLURLCOMPLETE
	CLOB	ATANH	DAYOFWEEK	DLURLPATH
	CONCAT	ATAN2	DAYOFYEAR	DLURLPATHONLY
	DBCLOB	BIGINT	DAYS	DLURLSCHEME
	DIFFERENCE	CEILING	HOUR	DLURLSERVER
	GRAPHIC	or CEIL	JULIAN_DAY	DLVALUE
	HEX	COS	MICROSECOND	EVENT_MON_STATE
	INSERT	COSH	MIDNIGHT_SECONDS	GENERATE_UNIQUE
	LCASE or LOWER	COT	MINUTE	NODENUMBER
	LCASE	DECIMAL	MONTH	NULLIF
	(SYSFUN schema)	DEGREES	MONTHNAME	PARTITION
	LEFT	DIGITS	NOW	RAISE_ERROR
	LENGTH	DOUBLE	QUARTER	TABLE_NAME
	LOCATE	EXP	SECOND	TABLE_SCHEMA
	LONG_VARCHAR	FLOAT	TIME	TRANSLATE
	LONG_VARGRAPHIC	FLOOR	TIMESTAMP	TYPE_ID
	LTRIM	INTEGER	TIMESTAMP_ISO	TYPE_NAME
	LTRIM	LN	TIMESTAMPDIFF	TYPE_SCHEMA
	(SYSFUN schema)	LOG	WEEK	VALUE
	POSSTR	LOG10	YEAR	
	REPEAT	MOD		
	REPLACE	POWER		
	RIGHT	RADIANS		
	RTRIM	RAND		
	RTRIM	REAL		
	(SYSFUN schema)	ROUND		
	SOUNDEX	SIGN		
	SPACE	SIN		
	SUBSTR	SINH		
	TRUNCATE or TRUNC	SMALLINT		
	UCASE or UPPER	SQRT		
	VARCHAR	TAN		
	VARGRAPHIC	TANH		
		TRUNCATE		

**Table F-1. Scalar Functions Supported (cont.)**

Data Store	String Functions	Numeric Functions	Timedate Functions	System Functions
Informix	CONCAT LEFT LENGTH LTRIM REPLACE RTRIM SUBSTRING	ABS ACOS ASIN ATAN ATAN2 COS COT EXP FLOOR LOG LOG10 MOD PI POWER ROUND SIN SQRT TAN TRUNCATE	CURDATE CURTIME DAYOFMONTH DAYOFWEEK MONTH NOW TIMESTAMPADD TIMESTAMPDIFF YEAR	DATABASE USER

**Table F-1. Scalar Functions Supported (cont.)**

<b>Data Store</b>	<b>String Functions</b>	<b>Numeric Functions</b>	<b>Timedate Functions</b>	<b>System Functions</b>
MySQL	ASCII CHAR CONCAT INSERT LCASE LEFT LENGTH LOCATE LOCATE_2 LTRIM REPEAT REPLACE RIGHT RTRIM SOUNDEX SPACE SUBSTRING UCASE	ABS ACOS ASIN ATAN ATAN2 CEILING COS COT DEGREES EXP FLOOR LOG LOG10 MOD PI POWER RADIAN RAND ROUND SIGN SIN SQRT TAN TRUNCATE	CURDATE CURRENT_DATE CURRENT_TIME CURRENT_TIMESTAMP CURTIME DAYNAME DAYOFMONTH DAYOFWEEK DAYOFYEAR EXTRACT HOUR MINUTE MONTH MONTHNAME NOW QUARTER SECOND TIMESTAMPADD TIMESTAMPDIFF WEEK YEAR	DATABASE IFNULL USER

**Table F-1. Scalar Functions Supported (cont.)**

Data Store	String Functions	Numeric Functions	Timedate Functions	System Functions
Oracle	ASCII BIT_LENGTH CHAR CONCAT INSERT LCASE LEFT LENGTH LOCATE LOCATE2 LTRIM OCTET_LENGTH REPEAT REPLACE RIGHT RTRIM SOUNDEX SPACE SUBSTRING UCASE	ABS ACOS ASIN ATAN ATAN2 CEILING COS COT EXP FLOOR LOG LOG10 MOD PI POWER ROUND SIGN SIN SQRT TAN TRUNCATE	CURDATE DAYNAME DAYOFMONTH DAYOFWEEK DAYOFYEAR HOUR MINUTE MONTH MONTHNAME NOW QUARTER SECOND WEEK YEAR	IFNULL USER

**Table F-1. Scalar Functions Supported (cont.)**

<b>Data Store</b>	<b>String Functions</b>	<b>Numeric Functions</b>	<b>Timedate Functions</b>	<b>System Functions</b>
SQL Server	ASCII	ABS	DAYNAME	DATABASE
	CHAR	ACOS	DAYOFMONTH	IFNULL
	CONCAT	ASIN	DAYOFWEEK	USER
	DIFFERENCE	ATAN	DAYOFYEAR	
	INSERT	ATAN2	EXTRACT	
	LCASE	CEILING	HOUR	
	LEFT	COS	MINUTE	
	LENGTH	COT	MONTH	
	LOCATE	DEGREES	MONTHNAME	
	LTRIM	EXP	NOW	
	REPEAT	FLOOR	QUARTER	
	REPLACE	LOG	SECOND	
	RIGHT	LOG10	TIMESTAMPADD	
	RTRIM	MOD	TIMESTAMPDIFF	
	SOUNDEX	PI	WEEK	
	SPACE	POWER	YEAR	
	SUBSTRING	RADIANS		
	UCASE	RAND		
		ROUND		
		SIGN		
		SIN		
		SQRT		
		TAN		
		TRUNCATE		

**Table F-1. Scalar Functions Supported (cont.)**

<b>Data Store</b>	<b>String Functions</b>	<b>Numeric Functions</b>	<b>Timedate Functions</b>	<b>System Functions</b>
Sybase	ASCII CHAR CONCAT DIFFERENCE INSERT LCASE LEFT LENGTH LOCATE LTRIM REPEAT RIGHT RTRIM SOUNDEX SPACE SUBSTRING UCASE	ABS ACOS ASIN ATAN ATAN2 CEILING COS COT DEGREES EXP FLOOR LOG LOG10 MOD PI POWER RADIAN RAND ROUND SIGN SIN SQRT TAN	DAYNAME DAYOFMONTH DAYOFWEEK DAYOFYEAR HOUR MINUTE MONTH MONTHNAME NOW QUARTER SECOND TIMESTAMPADD TIMESTAMPDIFF WEEK YEAR	DATABASE IFNULL USER

---

# Outer Join Escape Sequences

JDBC supports the SQL92 left, right, and full outer join syntax.  
The escape sequence for outer joins is:

{oj *outer-join*}

where *outer-join* is:

*table-reference* {LEFT | RIGHT | FULL} OUTER JOIN  
{*table-reference* | *outer-join*} ON *search-condition*

where:

*table-reference* is a database table name.

*search-condition* is the join condition you want to use for the tables.

## Example:

```
SELECT Customers.CustID, Customers.Name, Orders.OrderID,  
Orders.Status  
FROM {oj Customers LEFT OUTER JOIN  
Orders ON Customers.CustID=Orders.CustID}  
WHERE Orders.Status='OPEN'
```

[Table F-2](#) lists the outer join escape sequences supported by DataDirect Connect for JDBC drivers for each data store.

**Table F-2. Outer Join Escape Sequences Supported**

Data Store	Outer Join Escape Sequences
DB2	Left outer joins Right outer joins Nested outer joins
Informix	Left outer joins Right outer joins Nested outer joins
MySQL	Left outer joins Right outer joins Nested outer joins
Oracle	Left outer joins Right outer joins Nested outer joins
SQL Server	Left outer joins Right outer joins Full outer joins Nested outer joins
Sybase	Left outer joins Right outer joins Nested outer joins

---

## LIKE Escape Character Sequence for Wildcards

You can specify the character to be used to escape wildcard characters (%) and (\_, for example) in LIKE clauses. The escape sequence for escape characters is:

```
{escape 'escape-character'}
```

where *escape-character* is the character used to escape the wildcard character.

For example, the following SQL statement specifies that an asterisk (\*) be used as the escape character in the LIKE clause for the wildcard character %:

```
SELECT col1 FROM table1 WHERE col1 LIKE '*%%' {escape '*'}
```

---

## Procedure Call Escape Sequences

A procedure is an executable object stored in the data store. Generally, it is one or more SQL statements that have been precompiled. The escape sequence for calling a procedure is:

```
{[?=]call procedure-name[(parameter[,parameter]...)]}
```

where:

*procedure-name* specifies the name of a stored procedure.

*parameter* specifies a stored procedure parameter.

NOTE: For DB2 for Linux/UNIX/Windows, a catalog name cannot be used when calling a stored procedure. Also, for DB2 v8.1 and v8.2 for Linux/UNIX/Windows, literal parameter values are supported for stored procedures. Other supported DB2 versions do not support literal parameter values for stored procedures.



# G Using DataDirect Test™

This appendix contains a tutorial that takes you through a step-by-step example of how to use DataDirect Test, a tool that allows you to test and debug your JDBC applications during development.

Use DataDirect Test to test your JDBC applications and learn the JDBC API. DataDirect Test contains menu selections that correspond to specific JDBC functions—for example, connecting to a database or passing a SQL statement. DataDirect Test allows you to perform the following tasks:

- Execute a single JDBC method or execute multiple JDBC methods simultaneously, so that you can easily perform some common tasks, such as returning result sets
- Display the results of all JDBC function calls in one window, while displaying fully commented, JDBC code in an alternate window

DataDirect Test only works with DataDirect Connect *for JDBC* drivers.

---

## DataDirect Test™ Tutorial

This DataDirect Test tutorial explains how to use the most important features of DataDirect Test (and the JDBC API) and assumes that you can connect to a database with the standard available demo table or fine-tune the sample SQL statements shown in this example as appropriate for your environment.

NOTE: The step-by-step examples used in this tutorial do not show typical clean-up routines (for example, closing result sets and connections). These steps have been omitted to simplify the examples. Do not forget to add these steps when you use equivalent code in your applications.

## Configuring DataDirect Test™

The default DataDirect Test configuration file is:

*install\_dir/testforjdbc/Config.txt*

where *install\_dir* is your DataDirect Connect for JDBC installation directory.

The DataDirect Test configuration file can be edited as appropriate for your environment using any text editor. All parameters are configurable, but the most commonly configured parameters are:

Drivers	A list of colon-separated JDBC driver classes.
DefaultDriver	The default JDBC driver that appears in the Get Driver URL window.
Databases	A list of comma-separated JDBC URLs. The first item in the list appears as the default in the database selection window. You can use one of these URLs as a template when you make a JDBC connection. The default Config.txt file contains example URLs for most databases.
InitialContextFactory	Set to com.sun.jndi.fscontext.RefFSContextFactory if you are using file system data sources, or com.sun.jndi.ldap.LdapCtxFactory if you are using LDAP.

ContextProviderURL	The location of the .bindings file if you are using file system data sources, or your LDAP Provider URL if you are using LDAP.
Datasources	A list of comma-separated JDBC data sources. The first item in the list appears as the default in the data source selection window.

## Starting DataDirect Test™

How you start DataDirect Test depends on your platform:

- **As a Java application on Windows**—Run the testforjdbc.bat file located in the testforjdbc subdirectory in your DataDirect Connect *for JDBC* installation directory.

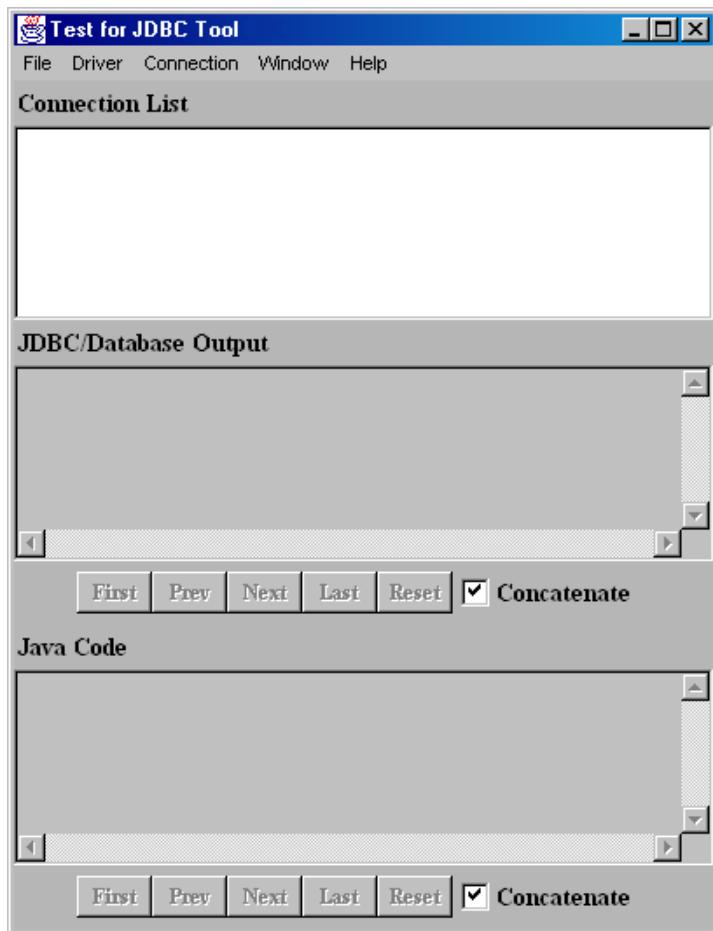
On Windows 9x and Me, double-clicking testforjdbc.bat opens a DOS window and displays the error "Out of environment space." To prevent this, use the following procedure:

- a After installing DataDirect Connect *for JDBC*, locate the testforjdbc.bat file in the testforjdbc subdirectory in the installation directory.
- b Right-click testforjdbc.bat and select **Properties**. After the properties display, select the **Memory** tab.
- c On the Memory tab, locate the Initial environment setting. From this drop-down list, select **1024**. Then, select the **Protected** check box. Click **OK**. A testforjdbc shortcut is created in the same directory with testforjdbc.bat.
- d Double-click either testforjdbc.bat or the testforjdbc shortcut. DataDirect Test will open normally without generating the error.

NOTE: Do not delete the testforjdbc shortcut; the 1024 environment setting will be lost if the shortcut is deleted.

- **As a Java application on UNIX**—Run the testforjdbc.sh shell script located in the testforjdbc subdirectory in the installation directory.

After you start DataDirect Test, the Test for JDBC Tool window appears.



The main Test for JDBC Tool window shows the following information:

- In the Connection List box, a list of available connections.
- In the JDBC/Database Output scroll box, a report indicating whether the last action succeeded or failed.
- In the Java Code scroll box, the actual Java code used to implement the last action.

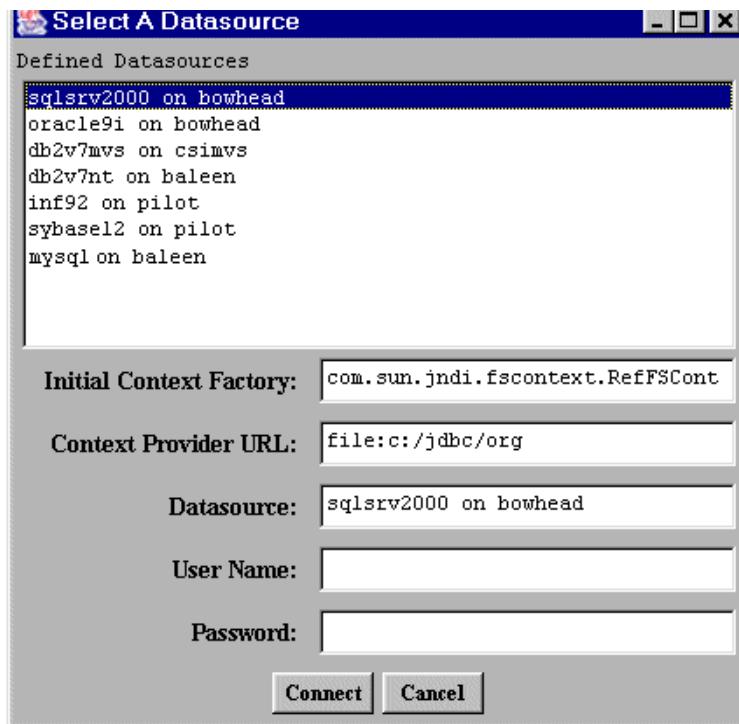
TIP: DataDirect Test windows contain two Concatenate check boxes. Select a Concatenate check box to see a cumulative record of previous actions; otherwise, only the last action is shown. Selecting Concatenate can degrade performance, particularly when displaying large result sets.

## Connecting Using DataDirect Test™

There are two methods to connect using DataDirect Test: using a data source or using a driver/database selection.

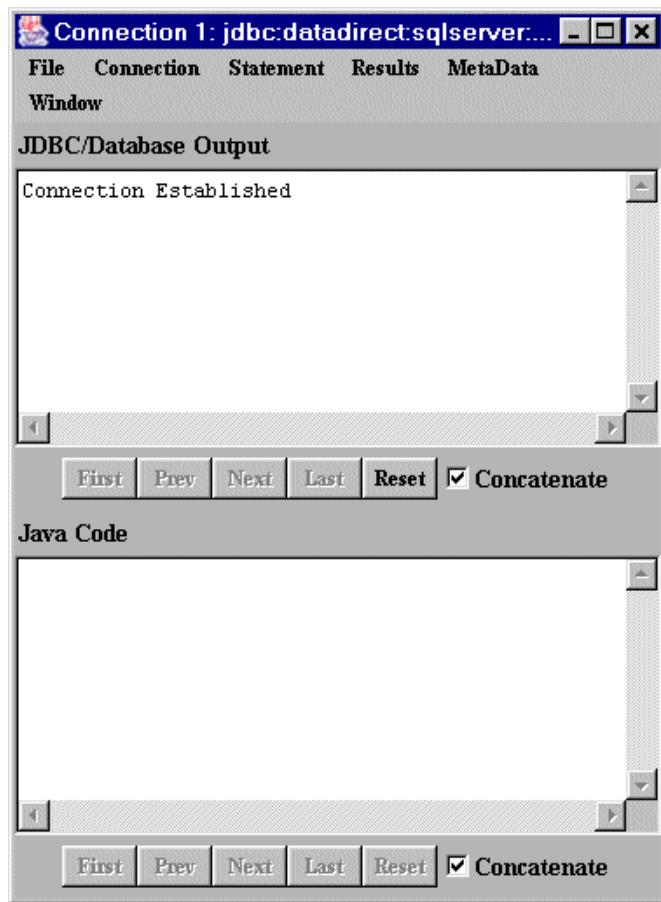
### ***Connecting Using a Data Source***

- 1 From the main Test for JDBC Tool window menu, select **Connection / Connect to DB via Data Source**. DataDirect Test displays the Select A Datasource window.



- 2 Select a data source from the Defined Datasources pane. In the User Name and Password fields, type the values for the User and Password connection properties; then, click **Connect**. For information about JDBC connection properties, see the appropriate driver chapter.

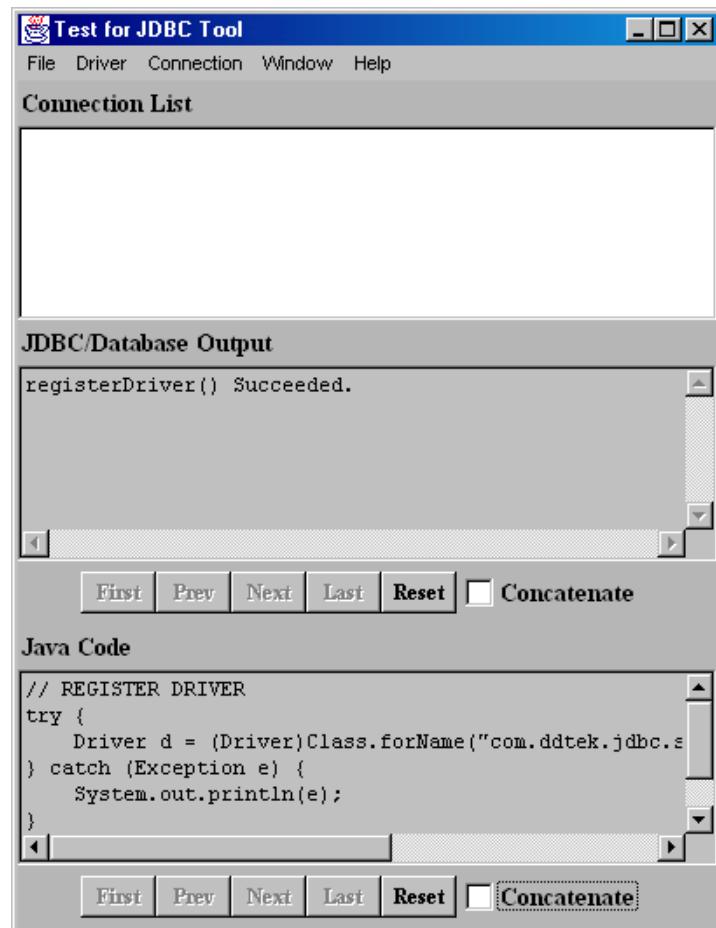
- 3 If the connection was successful, the Connection window appears and shows the Connection Established message in the JDBC/Database Output scroll box.



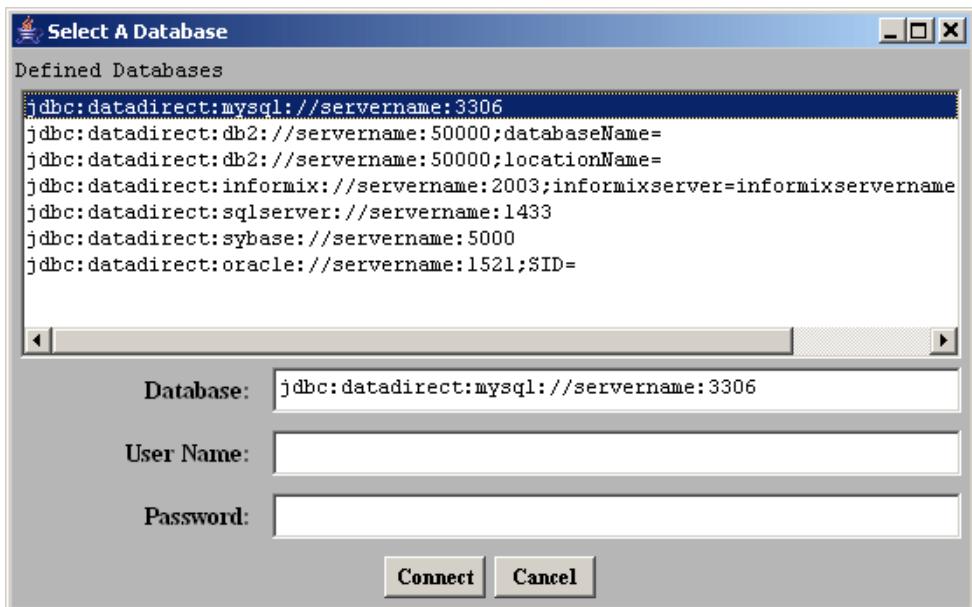
## Connecting Using Driver/Database Selection

- 1 From the main Test for JDBC Tool window menu, select **Driver / Register Driver**. DataDirect Test prompts for a JDBC driver name.
- 2 In the Please Supply a Driver URL field, make sure that a driver is specified (for example com.ddtek.jdbc.sqlserver.SQLServerDriver); then, click **OK**.

If the DataDirect Connect for JDBC driver was registered successfully, the main Test for JDBC Tool window appears with a confirmation in the JDBC/Database Output scroll box.



- 3 Select Connection / Connect to DB from the main Test for JDBC Tool menu. The Select A Database window prompts with a list of default connection URLs.

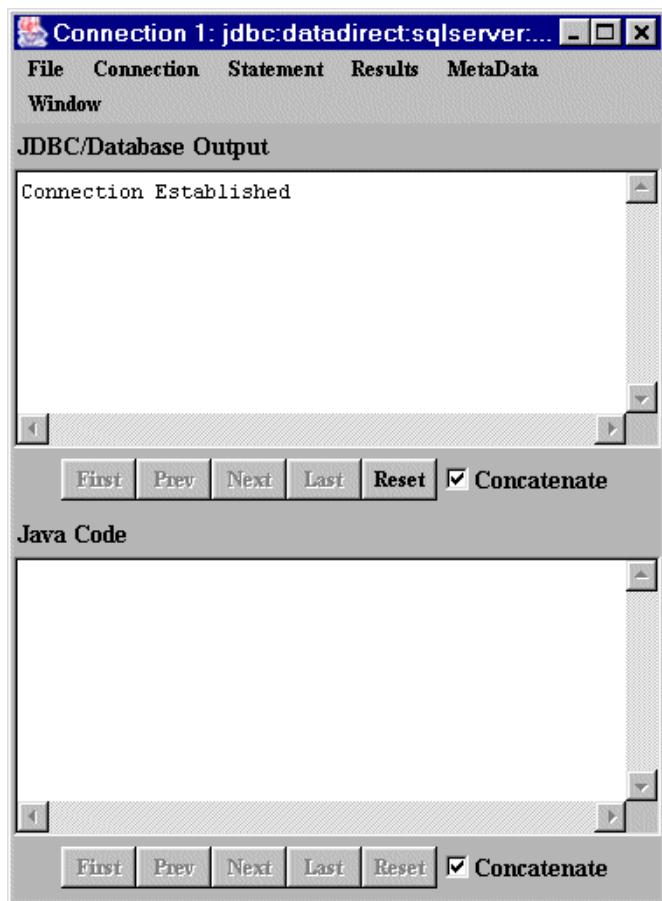


- 4 Select one of the default DataDirect Connect for JDBC driver connection URLs. In the Database field, modify the default values of the connection URL appropriately for your environment.

NOTE: There are two entries for DB2, one with locationName and another with databaseName. If you are connecting to DB2 for z/OS or DB2 for iSeries, select the entry containing locationName. If you are connecting to DB2 for Linux/UNIX/Windows, select the entry containing databaseName.

- 5 In the User Name and Password fields, type the values for the User and Password connection properties; then, click **Connect**. For information about JDBC connection properties, see the appropriate driver chapter.

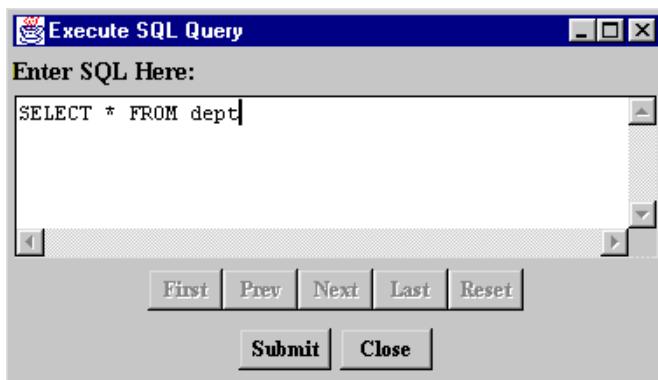
- 6 If the connection was successful, the Connection window appears and shows the Connection Established message in the JDBC/Database Output scroll box.



## Executing a Simple Select Statement

This example explains how to execute a simple Select statement and return the results.

- 1 From the Connection window menu, select **Connection / Create Statement**. The connection window indicates that the creation of the statement was successful.
- 2 Select **Statement / Execute Stmt Query**. DataDirect Test displays a dialog box that prompts for a SQL statement.
- 3 Specify the Select statement that you want to execute.



Click **Submit**; then, click **Close**.

- 4 Select **Results / Show All Results**. The data from your result set is displayed.

The screenshot shows the DataDirect Test application interface. At the top, a menu bar includes File, Connection, Statement, Results, MetaData, and Window. Below the menu is a title bar for "Connection 1: jdbc:datadirect:sqlserver...". The main area is divided into two panes: "JDBC/Database Output" and "Java Code".

**JDBC/Database Output:** This pane displays a table of department information:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

**Java Code:** This pane contains Java code that processes the results:buf.append("\n");
rowCount++;
}
results.close();
} catch (Exception e) {
System.out.println(e);
return;
}

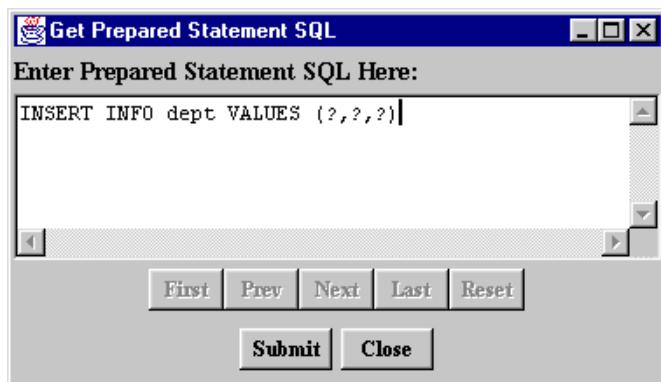
Below the code are navigation buttons: First, Prev, Next, Last, Reset, and a checkbox labeled "Concatenate". The "Concatenate" checkbox is checked in the Java Code pane but unchecked in the JDBC/Database Output pane.

- 5 Scroll through the code in the Java Code scroll box to see which JDBC calls have been implemented by DataDirect Test.

## Executing a Prepared Statement

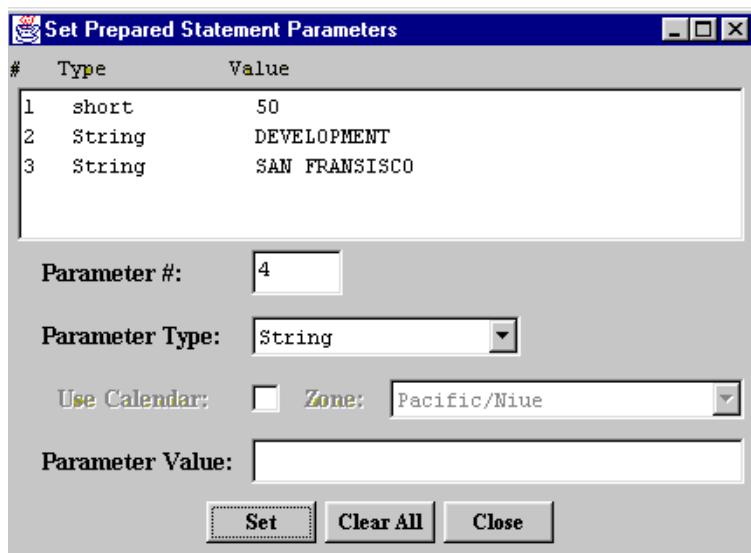
This example explains how to execute a parameterized statement multiple times.

- 1 From the Connection window menu, select **Connection / Create Prepared Statement**. DataDirect Test prompts you for a SQL statement.
- 2 Specify the Insert statement that you want to execute.



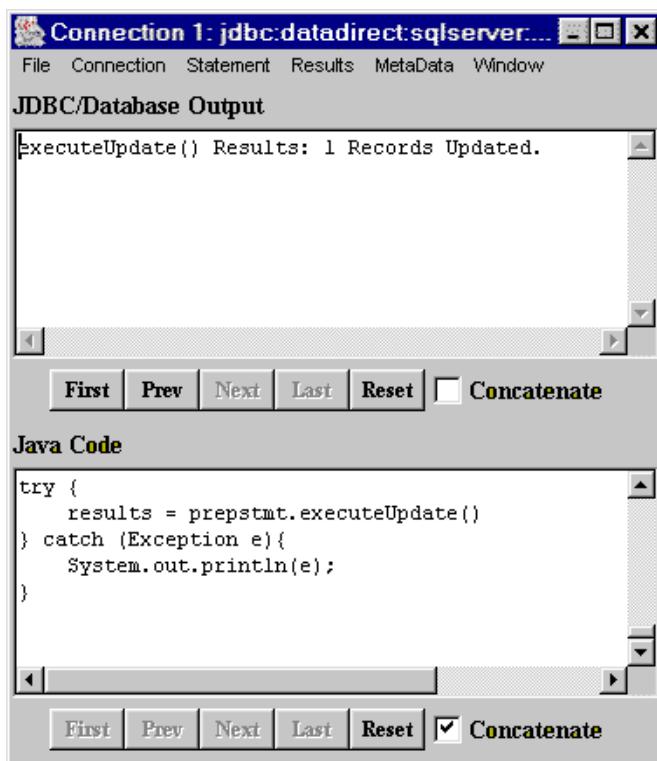
Click **Submit**; then, click **Close**.

- 3 Select Statement / Set Prepared Parameters.** To set the value and type for each parameter:
- a Type the parameter number.
  - b Select the parameter type.
  - c Type the parameter value.
  - d Click **Set** to pass this information to the JDBC driver.



- 4 When you are finished, click **Close**.**

- 5 Select Statement / Execute Stmt Update. The JDBC/Database Output scroll box indicates that one row has been inserted.



- 6 If you want to insert multiple records, repeat Step 3 and Step 5 for each record.

- 7 If you repeat the steps described in “[Executing a Simple Select Statement](#)” on page 739, you will see that the previously inserted records are also returned.

The screenshot shows the DataDirect Test application interface. At the top, there's a menu bar with File, Connection, Statement, Results, MetaData, and Window. Below the menu is a title bar for "Connection 1: jdbc:datadirect:sqlserver...". The main area is divided into two sections: "JDBC/Database Output" and "Java Code".

**JDBC/Database Output:** This section displays a table of department information:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	DEVELOPMENT	SAN FRANSISCO

Below the table are navigation buttons: First, Prev, Next, Last, Reset, and a checkbox labeled "Concatenate".

**Java Code:** This section contains the following Java code:

```
        }
        results.close();
    } catch (Exception e) {
        System.out.println(e);
        return;
    }
```

Below the code are navigation buttons: First, Prev, Next, Last, Reset, and a checked checkbox labeled "Concatenate".

## Retrieving Database Metadata

- 1 From the Connection window menu, select **Connection / Get DB Meta Data**.
- 2 Select **MetaData / Show Meta Data**. Information about the JDBC driver and the database to which you are connected is returned.

The screenshot shows the 'Connection 4: db2v7mvs on csimvs' window. The 'MetaData' tab is selected. The 'JDBC/Database Output' section displays various connection parameters:

Time Date Functions:	ALTDATE, ALTTIME, ...
URL:	jdbc:datadirect:db
User Name:	jdbc01
Catalog At Start:	false
Read Only:	false
// JDBC 3.0	
Locators Update Copy:	true
Null + Non Null Is Null:	true
Nulls Are Sorted At End:	false

Below this are 'First', 'Prev', 'Next', 'Last', 'Reset', and a checked 'Concatenate' checkbox.

The 'Java Code' section contains the following code:

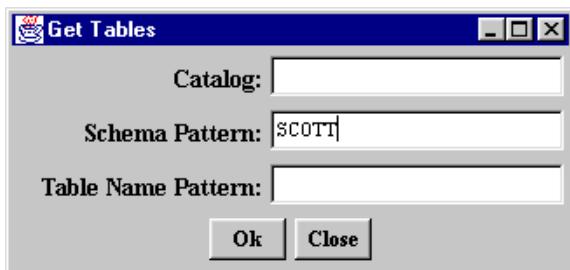
```
bval = dmd.usesLocalFilePerTable();
bval = dmd.usesLocalFiles();
// JDBC 2.0
bval = dmd.supportsBatchUpdates();
} catch (Exception e){
    System.out.println("Get Database Meta Data Fa
}
```

Below this are 'First', 'Prev', 'Next', 'Last', 'Reset', and a checked 'Concatenate' checkbox.

- 3 Scroll through the Java code in the Java Code scroll box to find out which JDBC calls have been implemented by DataDirect Test.

Metadata also allows you to query the database catalog (enumerate the tables in the database, for example). In this example, we will query all tables owned by the user SCOTT.

- 4 Select **MetaData / Tables**.
- 5 In the Schema Pattern field, type SCOTT.



- 6 Click **Ok**. The Connection window indicates that `getTables()` succeeded.

- 7 Select Results / Show All Results. All tables owned by SCOTT are returned.

The screenshot shows the DataDirect Test interface with two main panes. The top pane, titled 'JDBC/Database Output', displays a list of database objects owned by SCOTT, all of which are TABLES. The bottom pane, titled 'Java Code', contains a snippet of Java code related to database processing.

**JDBC/Database Output:**

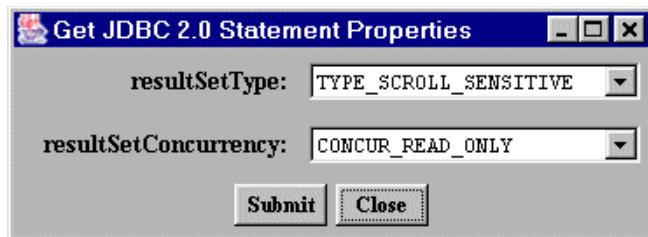
Owner	Name	Type
SCOTT	BONUS	TABLE
SCOTT	DEPT	TABLE
SCOTT	EMP	TABLE
SCOTT	LOB_TEST	TABLE
SCOTT	MFSQLTEST	TABLE
SCOTT	SALGRADE	TABLE
SCOTT	T01	TABLE
SCOTT	T02	TABLE

**Java Code:**

```
        rowcount++;
    }
    results.close();
} catch (Exception e) {
    System.out.println(e);
    return;
}
```

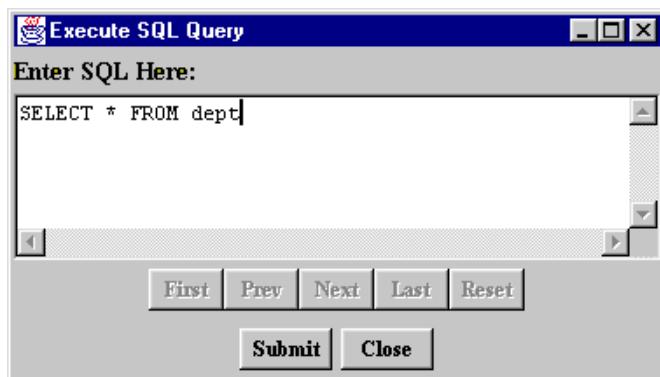
## Scrolling Through a Result Set

- 1 From the Connection window menu, select **Connection / Create JDBC 2.0 Statement**. DataDirect Test prompts for a result set type and concurrency.
- 2 In the resultSetType field, select **TYPE\_SCROLL\_SENSITIVE**. In the resultSetConcurrency field, select **CONCUR\_READ\_ONLY**.



Click **Submit**; then, click **Close**.

- 3 Select **Statement / Execute Stmt Query**.
- 4 Specify the Select statement that you want to execute.

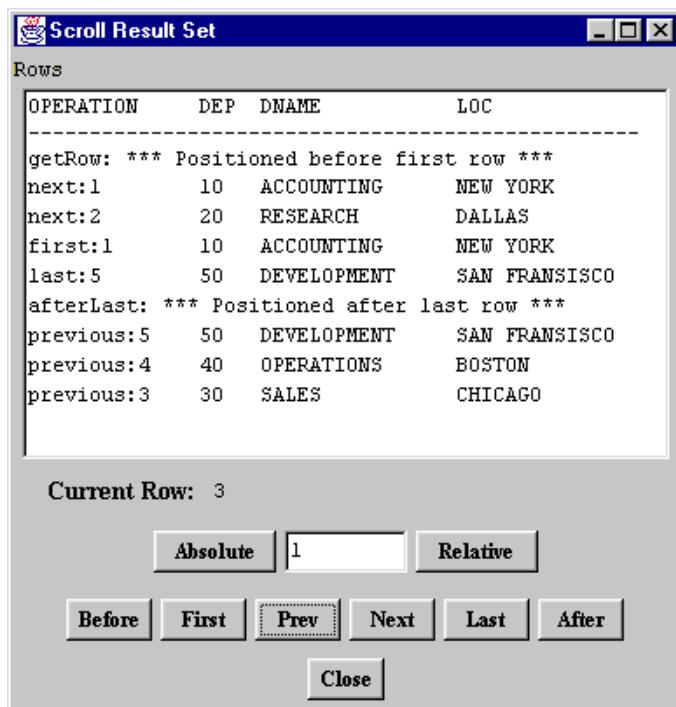


Click **Submit**; then, click **Close**.

- 5 Select Results / Scroll Results. The Scroll Result Set window indicates that the cursor is positioned before the first row.



- 6 Click the **Absolute**, **Relative**, **Before**, **First**, **Prev**, **Next**, **Last**, and **After** buttons as appropriate to navigate through the result set. After each action, the Scroll Result Set window displays the data at the current position of the cursor.

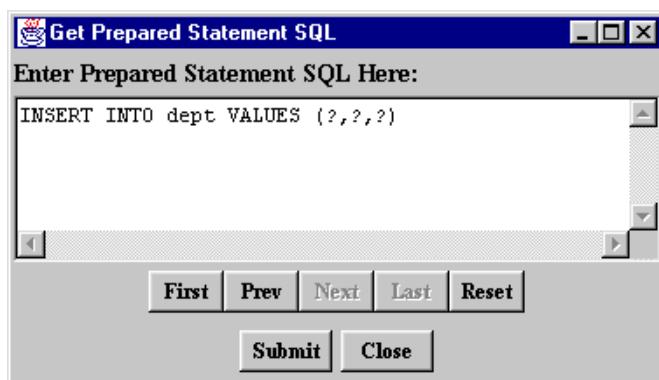


- 7 Click **Close**.

## Batch Execution on a Prepared Statement

Batch execution on a prepared statement allows you to update or insert multiple records simultaneously. In some cases, this can significantly improve system performance because fewer roundtrips to the database are required.

- 1 From the Connection window menu, select **Connection / Create Prepared Statement**.
- 2 Specify the Insert statement that you want to execute.



Click **Submit**; then, click **Close**.

- 3 Select **Statement / Add Stmt Batch**.

- 4 For each parameter:
- Type the parameter number.
  - Select the parameter type.
  - Type the parameter value.
  - Click **Set**.



- 5 Click **Add** to add the specified set of parameters to the batch. To add multiple parameter sets to the batch, repeat Step 3 through Step 5 as many times as necessary. When you are finished adding parameter sets to the batch, click **Close**.

- 6 Select Statement / Execute Stmt Batch. DataDirect Test displays the rowcount for each of the elements in the batch.

The screenshot shows the DataDirect Test application window titled "Connection 1: jdbc:datadirect:sqlserver...". The window has tabs: File, Connection, Statement, Results, MetaData, and Window. The "Statement" tab is active. The "JDBC/Database Output" pane displays the results of an UPDATE statement:

```
UPDATE COUNTS
-----
1
```

The "Java Code" pane shows the corresponding Java code for executing the batch:

```
// EXECUTE QUERY
int[] updateCounts;
try {
    updateCounts = stmt.executeBatch()
} catch (Exception e) {
    System.out.println(e);
}
```

At the bottom of both panes are navigation buttons: First, Prev, Next, Last, Reset, and Concatenate. In the "Java Code" pane, the "Concatenate" checkbox is checked.

- 7 If you re-execute the Select statement from “[Executing a Simple Select Statement](#)” on page 739, you see that the previously inserted records are returned.

The screenshot shows the DataDirect Test application interface. At the top, there's a menu bar with File, Connection, Statement, Results, MetaData, and Window. Below the menu is a title bar for "Connection 1: jdbc:datedirect:sqlserver...".

The main area is divided into two panes:

- JDBC/Database Output:** This pane displays a table of department information. The columns are DEPTNO, DNAME, and LOC. The data is as follows:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	DEVELOPMENT	SAN FRANSISCO
60	MARKETING	NEW YORK

- Java Code:** This pane contains the following Java code snippet:

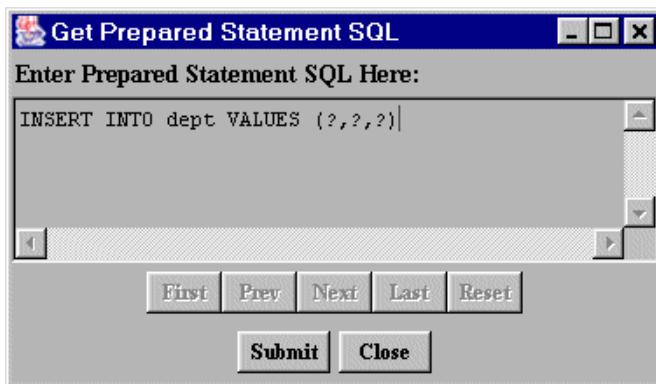
```
        }
        buf.append("\n");
        rowcount++;
    }
    results.close();
} catch (Exception e) {
    System.out.println(e);
    return;
}
```

At the bottom of each pane are navigation buttons: First, Prev, Next, Last, Reset, and Concatenate. In the Java Code pane, the "Concatenate" checkbox is checked.

## Returning ParameterMetaData

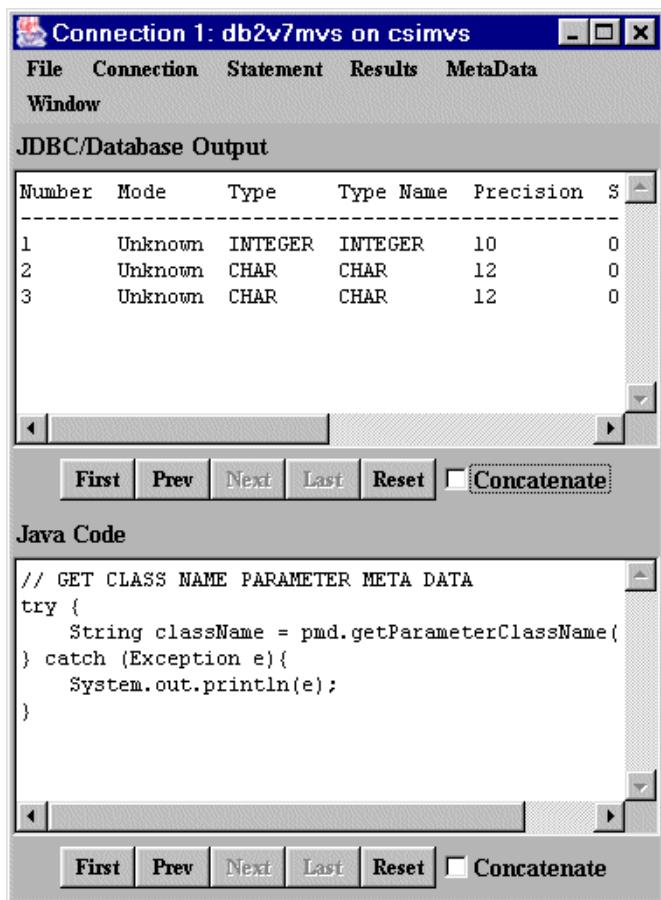
NOTE: Returning ParameterMetaData is a JDBC feature and requires a J2SE 1.4 or higher JVM.

- 1 From the Connection window menu, select **Connection / Create Prepared Statement**.
- 2 Specify the prepared statement that you want to execute.



Click **Submit**; then, click **Close**.

- 3 Select Statement / Get ParameterMetaData. The Connection window displays ParameterMetaData.



## Establishing Savepoints

NOTE: Savepoints is a JDBC feature and requires a J2SE 1.4 or higher JVM.

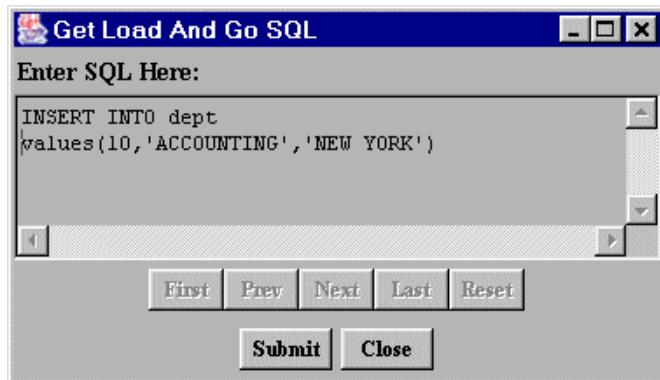
- 1 From the Connection window menu, select **Connection / Connection Properties**.
- 2 Select **TRANSACTION\_COMMITTED** from the Transaction Isolation drop-down list. Do not select the Auto Commit check box.



Click **Set**; then, click **Close**.

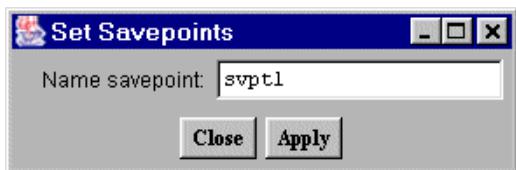
- 3 From the Connection window menu, select **Connection / Load and Go**. The Get Load And Go SQL window appears.

4 Specify the statement that you want to execute.

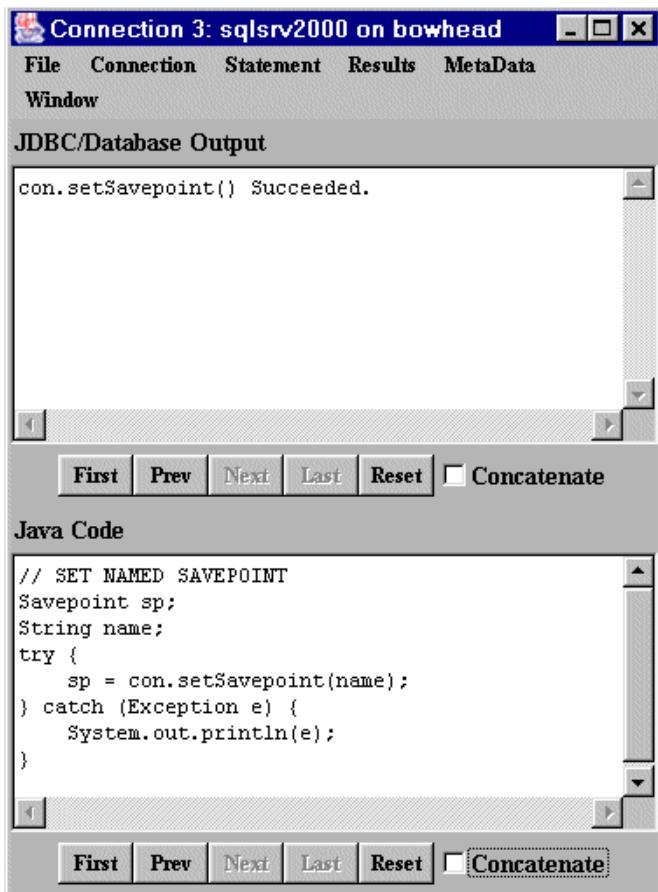


Click **Submit**.

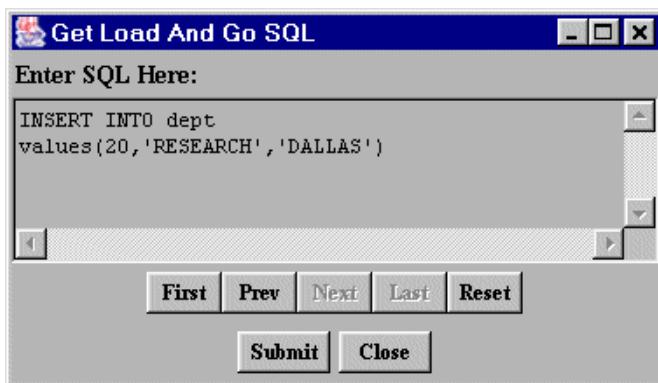
- 5 Select Connection / Set Savepoint. In the Set Savepoints window, specify a savepoint name.



Click **Apply**; then, click **Close**. The Connection window indicates whether or not the savepoint succeeded.



- 6 Return to the Get Load And Go SQL window and specify another statement.

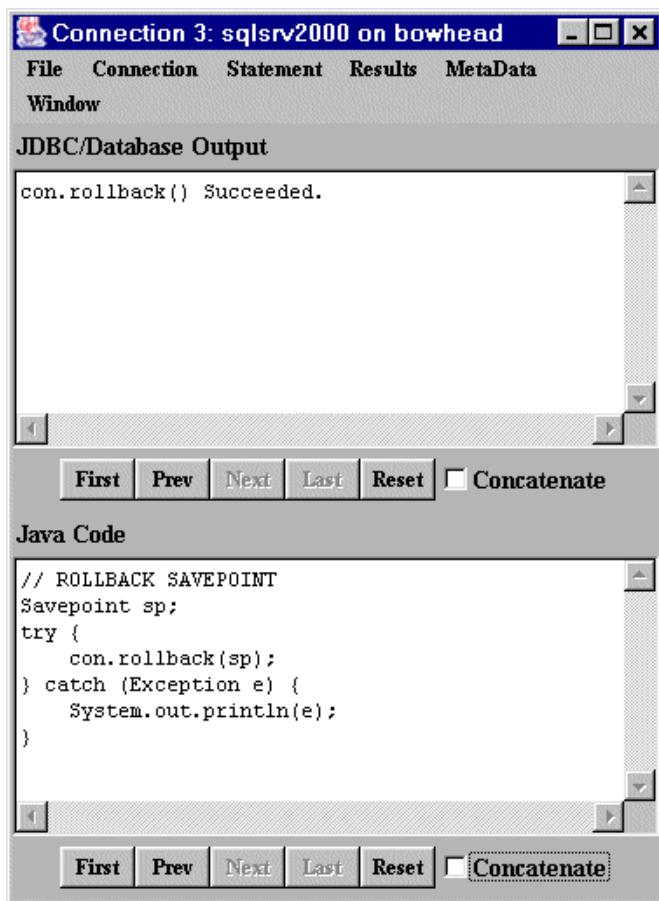


Click **Submit**.

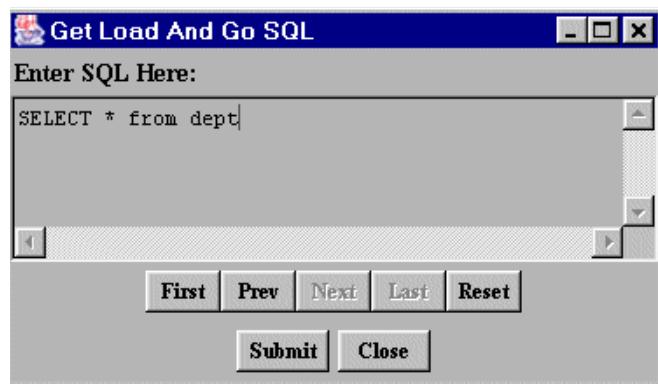
- 7 Select **Connection / Rollback Savepoint**. In the Rollback Savepoints window, specify the savepoint name.



Click **Apply**; then, click **Close**. The Connection window indicates whether or not the savepoint rollback succeeded.



- 8 Return to the Get Load And Go SQL window and specify another statement.



Click **Submit**; then, click **Close**. The Connection window displays data that was inserted before the first Savepoint. The second insert was rolled back.

The screenshot shows a window titled "Connection 3: sqlsrv2000 on bowhead". The menu bar includes File, Connection, Statement, Results, MetaData, Window, and JDBC/Database Output. The main area displays a table with three columns: DEPTNO, DNAME, and LOC. The data row is 10, ACCOUNTING, NEW YORK. Below the table are navigation buttons: First, Prev, Next, Last, Reset, and Concatenate. A scroll bar is visible on the right side of the table area. Below this is a "Java Code" section containing the following code:

```
// GET ALL RESULTS
StringBuffer buf = new StringBuffer();
try {
    ResultSetMetaData rsmd = results.getMetaData();
    int numCols = rsmd.getColumnCount();
    int i, rowcount = 0;

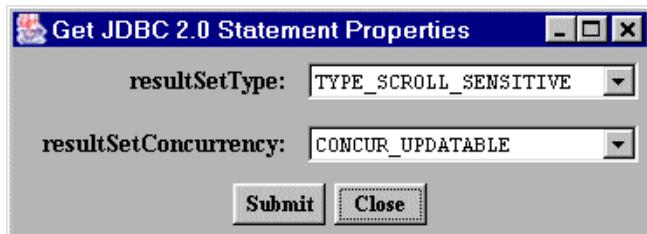
    // get column header info
    for (i=1; i <= numCols; i++) {
```

## Updatable Result Sets

The following examples illustrate updatable result sets by deleting, inserting, and updating a row.

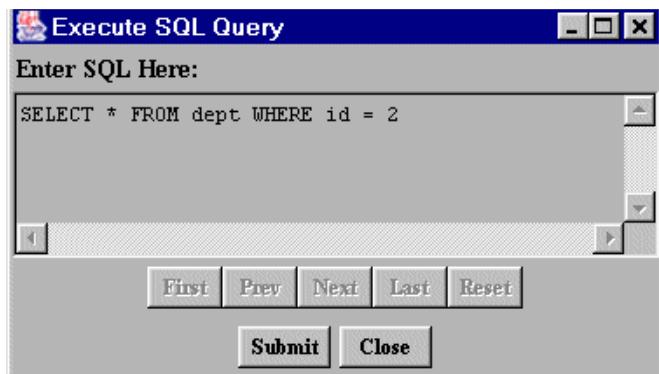
### ***Deleting a Row***

- 1 From the Connection window menu, select **Connection / Create JDBC 2.0 Statement**.
- 2 In the resultSetType field, select **TYPE\_SCROLL\_SENSITIVE**. In the resultSetConcurrency field, select **CONCUR\_UPDATABLE**.



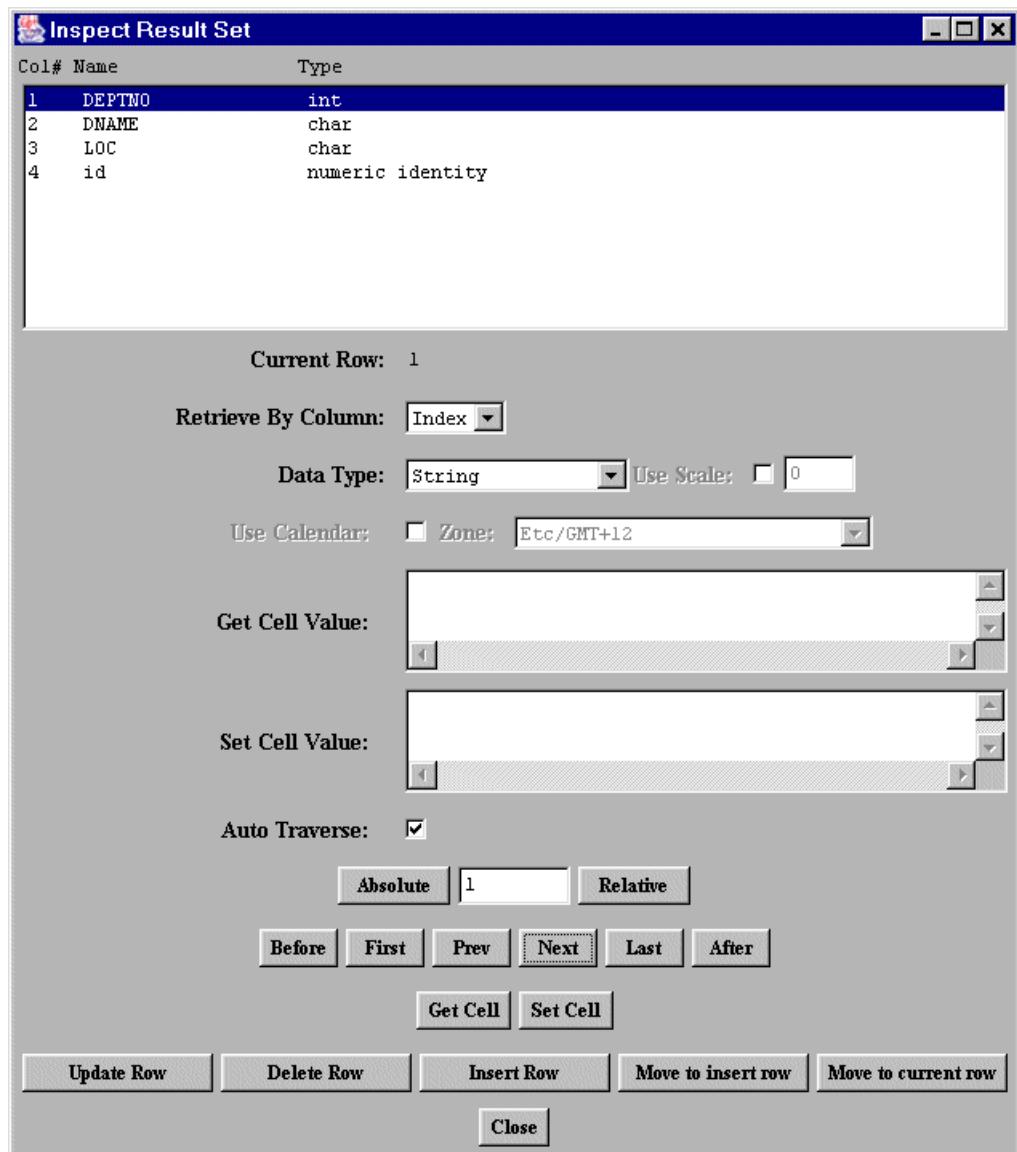
Click **Submit**; then, click **Close**.

- 3 Select **Statement / Execute Stmt Query**.
- 4 Specify the Select statement that you want to execute.



Click **Submit**; then, click **Close**.

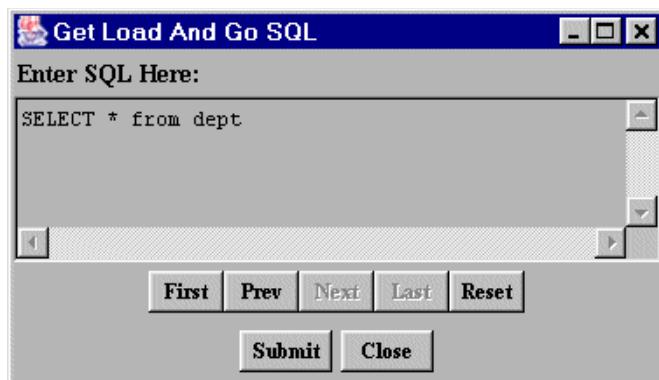
- 5 Select Results / Inspect Results. The Inspect Result Set window appears.



6 Click Next. Current Row changes to 1.

7 Click Delete Row.

- 8 To verify the result, return to the Connection menu and select **Connection / Load and Go**. The Get Load And Go SQL window appears.
- 9 Specify the statement that you want to execute.



Click **Submit**; then, click **Close**.

10 The Connection window shows one row returned.

The screenshot shows a window titled "Connection 3: sqlsrv2000 on bowhead". The menu bar includes File, Connection, Statement, Results, MetaData, Window, and JDBC/Database Output. The main area displays a table with the following data:

DEPTNO	DNAME	LOC	id
10	ACCOUNTING	NEW YORK	1

Below the table are navigation buttons: First, Prev, Next, Last, Reset, and Concatenate. The "Java Code" section contains the following Java code:

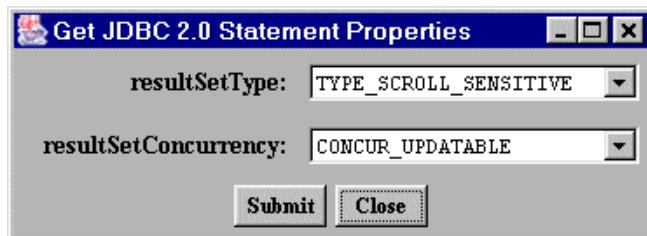
```
// GET ALL RESULTS
StringBuffer buf = new StringBuffer();
try {
    ResultSetMetaData rsmd = results.getMetaData();
    int numCols = rsmd.getColumnCount();
    int i, rowcount = 0;

    // get column header info
    for (i=1; i <= numCols; i++) {
        // code omitted for brevity
    }
}
```

Below the code are the same navigation buttons: First, Prev, Next, Last, Reset, and Concatenate.

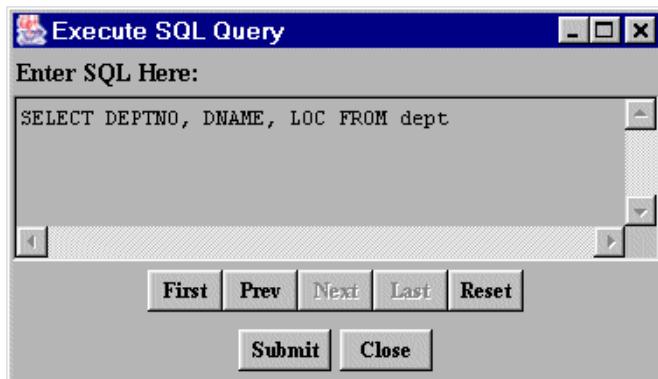
## *Inserting a Row*

- 1 From the Connection window menu, select **Connection / Create JDBC 2.0 Statement**.
- 2 In the resultSetType field, select **TYPE\_SCROLL\_SENSITIVE**. In the resultSetConcurrency field, select **CONCUR\_UPDATABLE**.



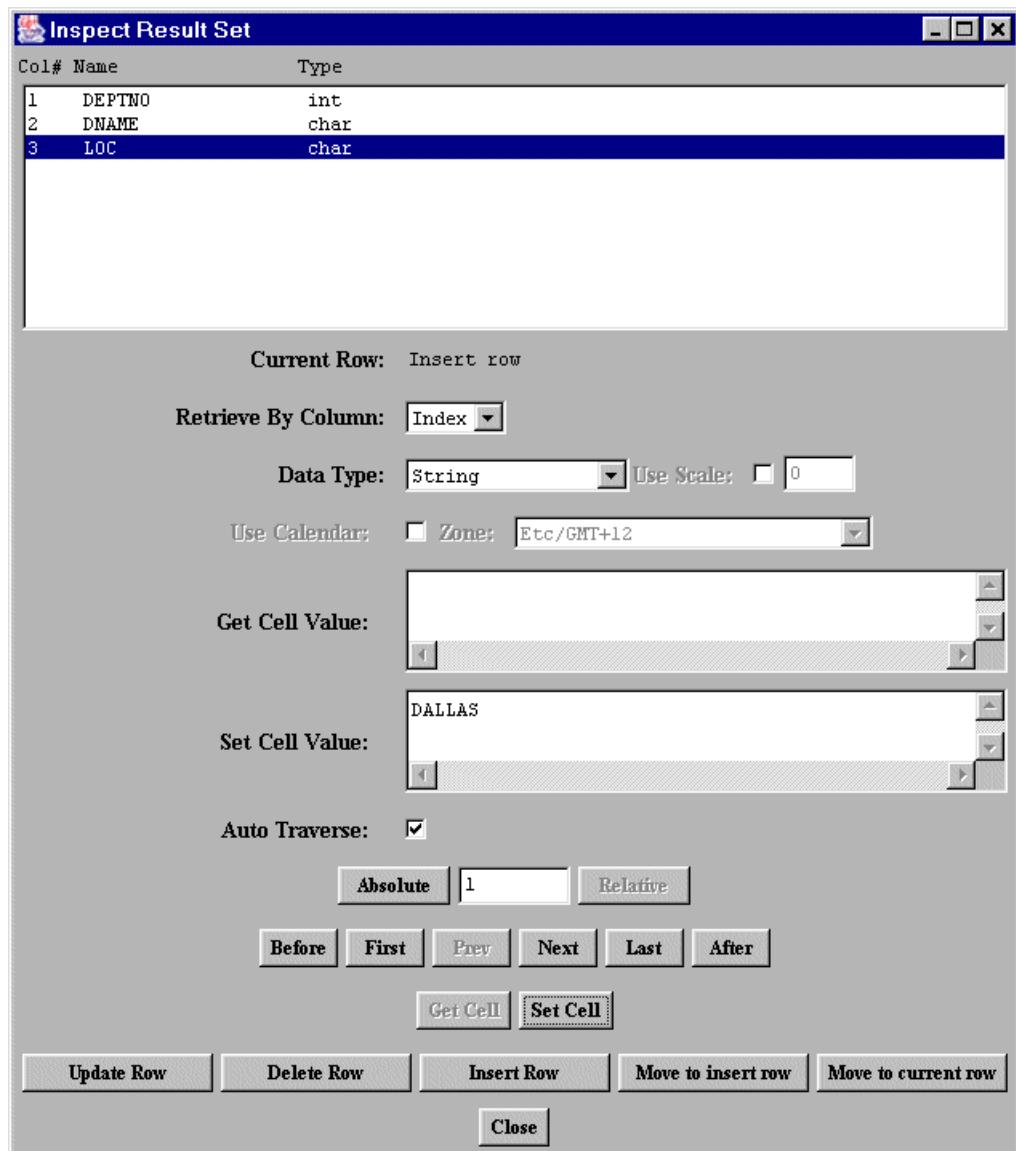
Click **Submit**; then, click **Close**.

- 3 Select **Statement / Execute Stmt Query**.
- 4 Specify the Select statement that you want to execute.



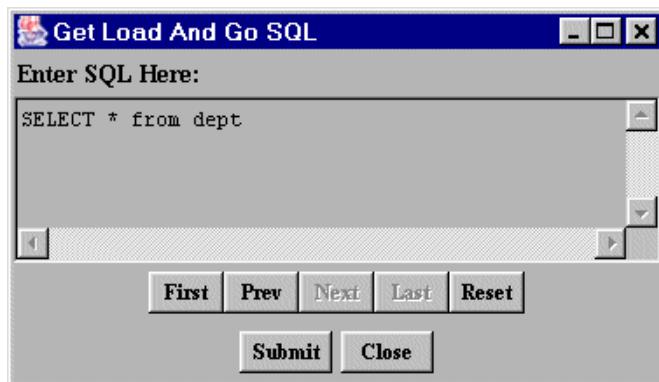
Click **Submit**; then, click **Close**.

- 5 Select Results / Inspect Results. The Inspect Result Set window is displayed.



- 6 Click Move to insert row; Current Row is now Insert row.

- 7 Change Data Type to int. In Set Cell Value, enter 20. Click **Set Cell**.
- 8 Select the second row in the top pane. Change the Data Type to String. In Set Cell Value, enter RESEARCH. Click **Set Cell**.
- 9 Select the third row in the top pane. In Set Cell Value, enter DALLAS. Click **Set Cell**.
- 10 Click **Insert Row**.
- 11 To verify the result, return to the Connection menu and select **Connection / Load and Go**. The Get Load And Go SQL window appears.
- 12 Specify the statement that you want to execute.



Click **Submit**; then, click **Close**.

13 The Connection window shows two rows returned.

The screenshot shows the DataDirect Test Connection window titled "Connection 5: sqlsrv2000 on bowhead". The window has tabs for File, Connection, Statement, Results, MetaData, and Window. The "Results" tab is active, displaying the "JDBC/Database Output" pane. The output pane shows the following text:  
createStatement() Succeeded.  
execute() Results: Returned ResultSet  
Show All Results Succeeded.  
DEPTNO DNAME LOC id  
-----  
10 ACCOUNTING NEW YORK 1  
20 RESEARCH DALLAS 3

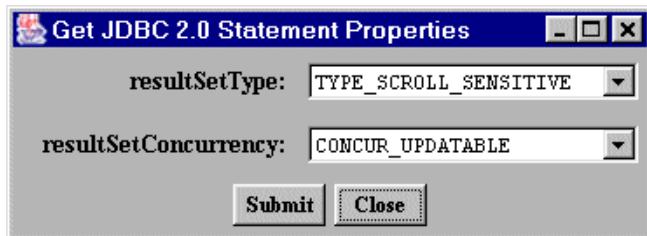
Below the output pane is a toolbar with buttons for First, Prev, Next, Last, Reset, and a checked checkbox for Concatenate. The "Java Code" pane below contains the following Java code:  
rowcount++;  
}  
results.close();  
} catch (Exception e) {  
System.out.println(e);  
return;  
}

Below the code pane is another toolbar with the same set of buttons as the first one.

Note that the ID will be 3 for the row just inserted, because it is an auto increment column.

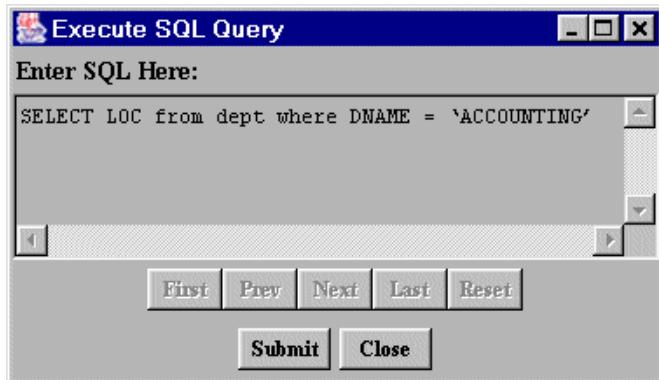
## ***Updating a Row***

- 1 From the Connection window menu, select **Connection / Create JDBC 2.0 Statement**.
- 2 In the resultSetType field, select **TYPE\_SCROLL\_SENSITIVE**. In the resultSetConcurrency field, select **CONCUR\_UPDATABLE**.



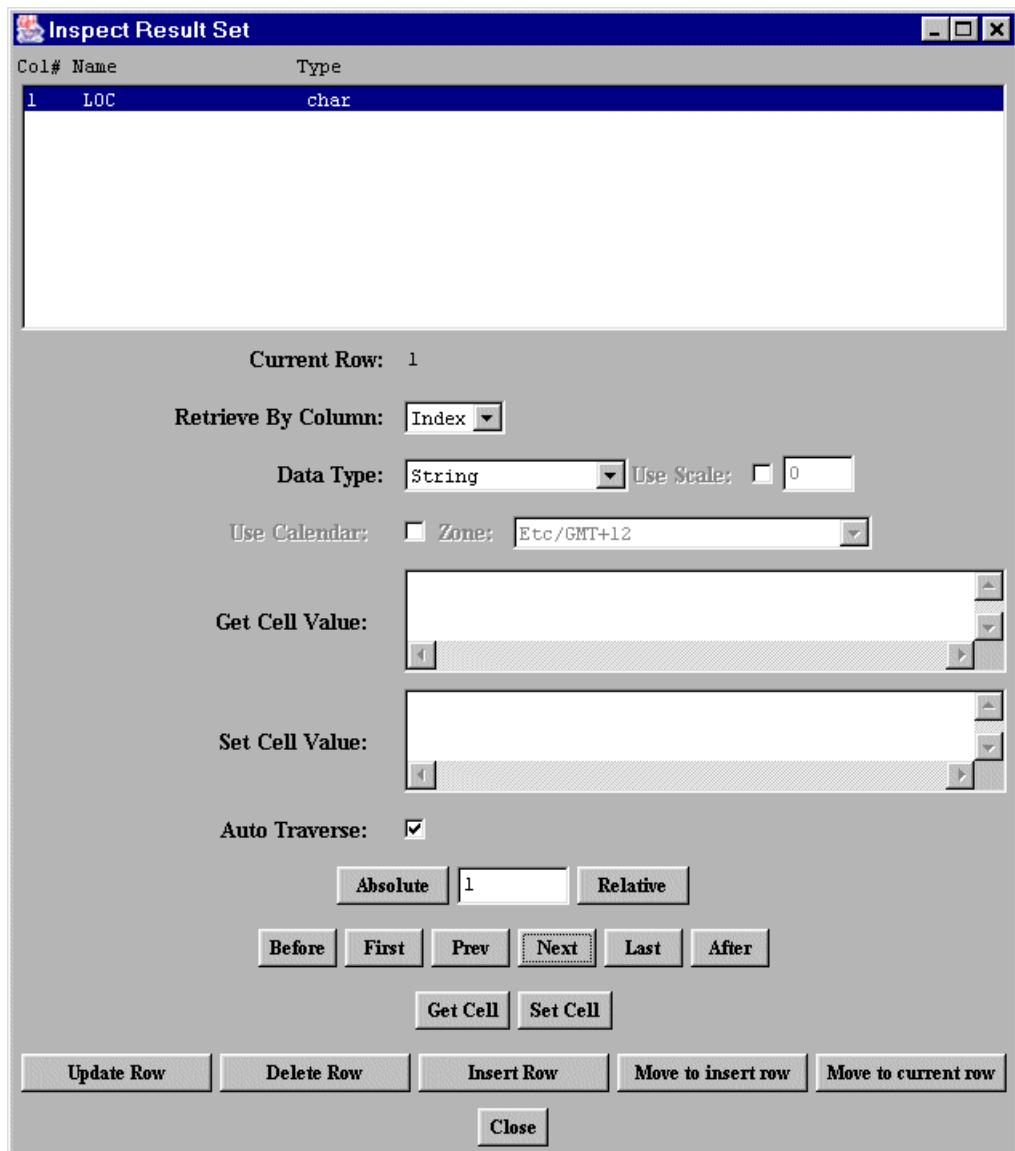
Click **Submit**; then, click **Close**.

- 3 Select **Statement / Execute Stmt Query**.
- 4 Specify the Select statement that you want to execute.



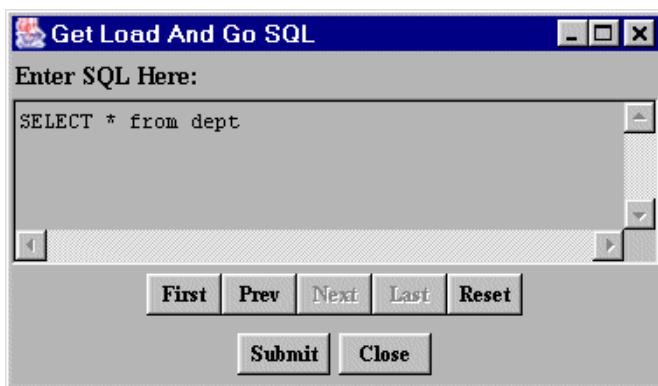
Click **Submit**; then, click **Close**.

- 5 Select Results / Inspect Results. The Inspect Result Set window is displayed.



- 6 Click Next. Current Row changes to 1.  
7 In Set Cell Value, enter RALEIGH. Click Set Cell.

- 8 Click **Update Row**.
- 9 To verify the result, return to the Connection menu and select **Connection / Load and Go**. The Get Load And Go SQL window appears.
- 10 Specify the statement that you want to execute.



Click **Submit**; then, click **Close**.

- 11 The Connection window shows LOC for accounting changed from NEW YORK to RALEIGH.

The screenshot shows a window titled "Connection 5: sqlsrv2000 on bowhead". The window has tabs: File, Connection, Statement, Results, MetaData, Window, and JDBC/Database Output. The JDBC/Database Output tab is active, displaying a table with four columns: DEPTNO, DNAME, LOC, and id. The data shows two rows: one for department 10 with location RALEIGH and id 1, and another for department 20 with location DALLAS and id 3. Below the table are navigation buttons: First, Prev, Next, Last, Reset, and Concatenate. The Java Code tab is also visible, containing Java code for retrieving results from a ResultSet. The code uses a StringBuffer to build a string and a ResultSetMetaData object to get column information. It loops through the columns to print their names.

DEPTNO	DNAME	LOC	id
10	ACCOUNTING	RALEIGH	1
20	RESEARCH	DALLAS	3

```
// GET ALL RESULTS
StringBuffer buf = new StringBuffer();
try {
    ResultSetMetaData rsmd = results.getMetaData();
    int numCols = rsmd.getColumnCount();
    int i, rowcount = 0;

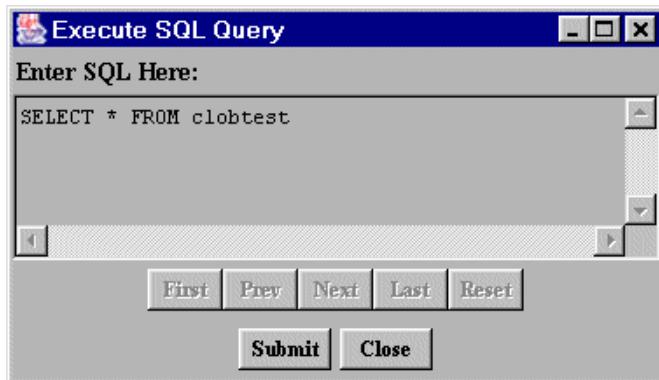
    // get column header info
    for (i=1; i <= numCols; i++) {
        System.out.print(rsmd.getColumnName(i) + "\t");
    }
}
```

## Large Object (LOB) Support

NOTE: LOB support (Blobs and Clob)s is a JDBC feature and requires a J2SE 1.4 or higher JVM.

The following example uses Clob data; however, this procedure also applies to Blob data. This example illustrates only one of several ways in which LOB data can be processed.

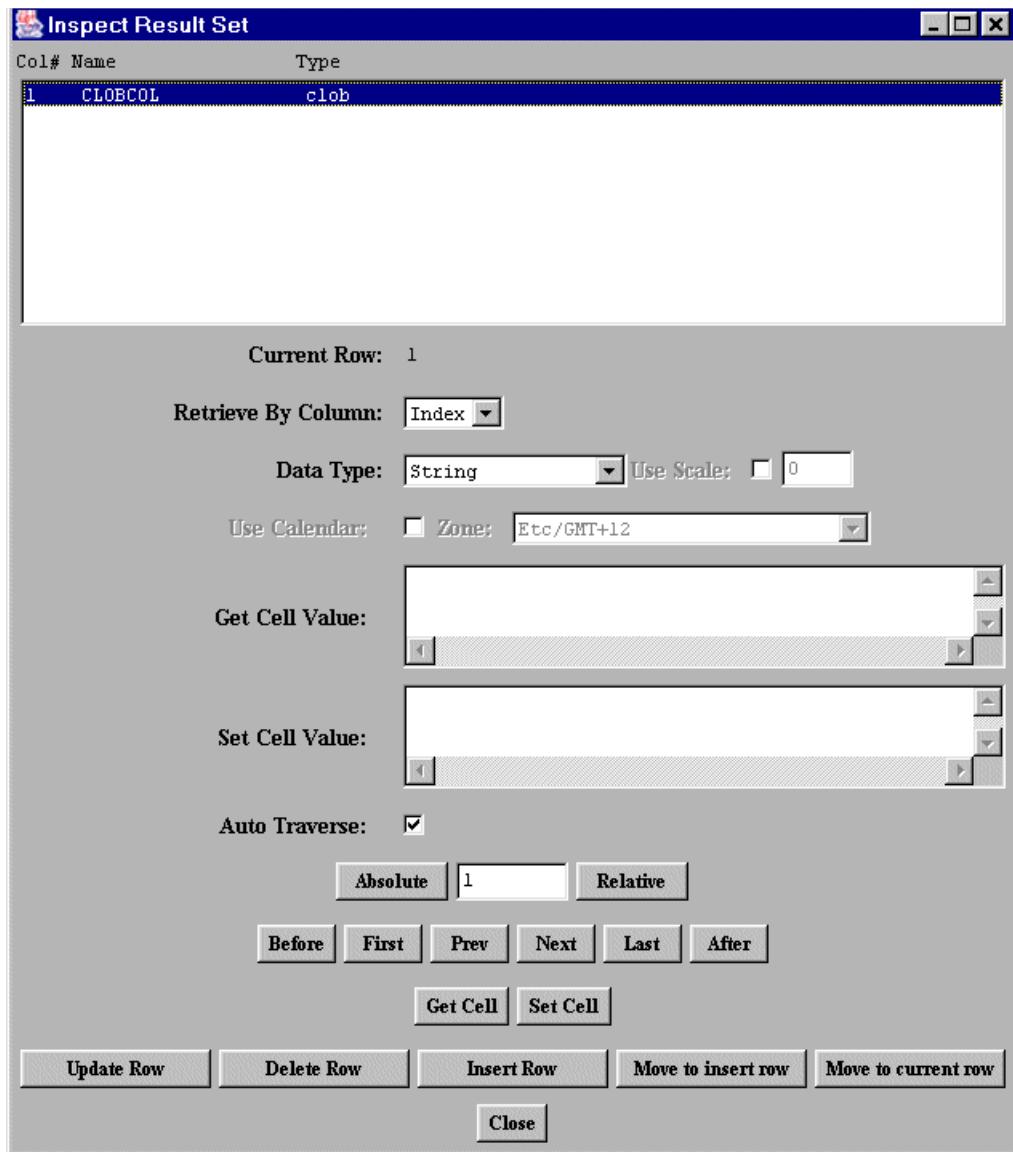
- 1 From the Connection window menu, select **Connection / Create Statement**.
- 2 Select **Statement / Execute Stmt Query**.
- 3 Specify the Select statement that you want to execute.



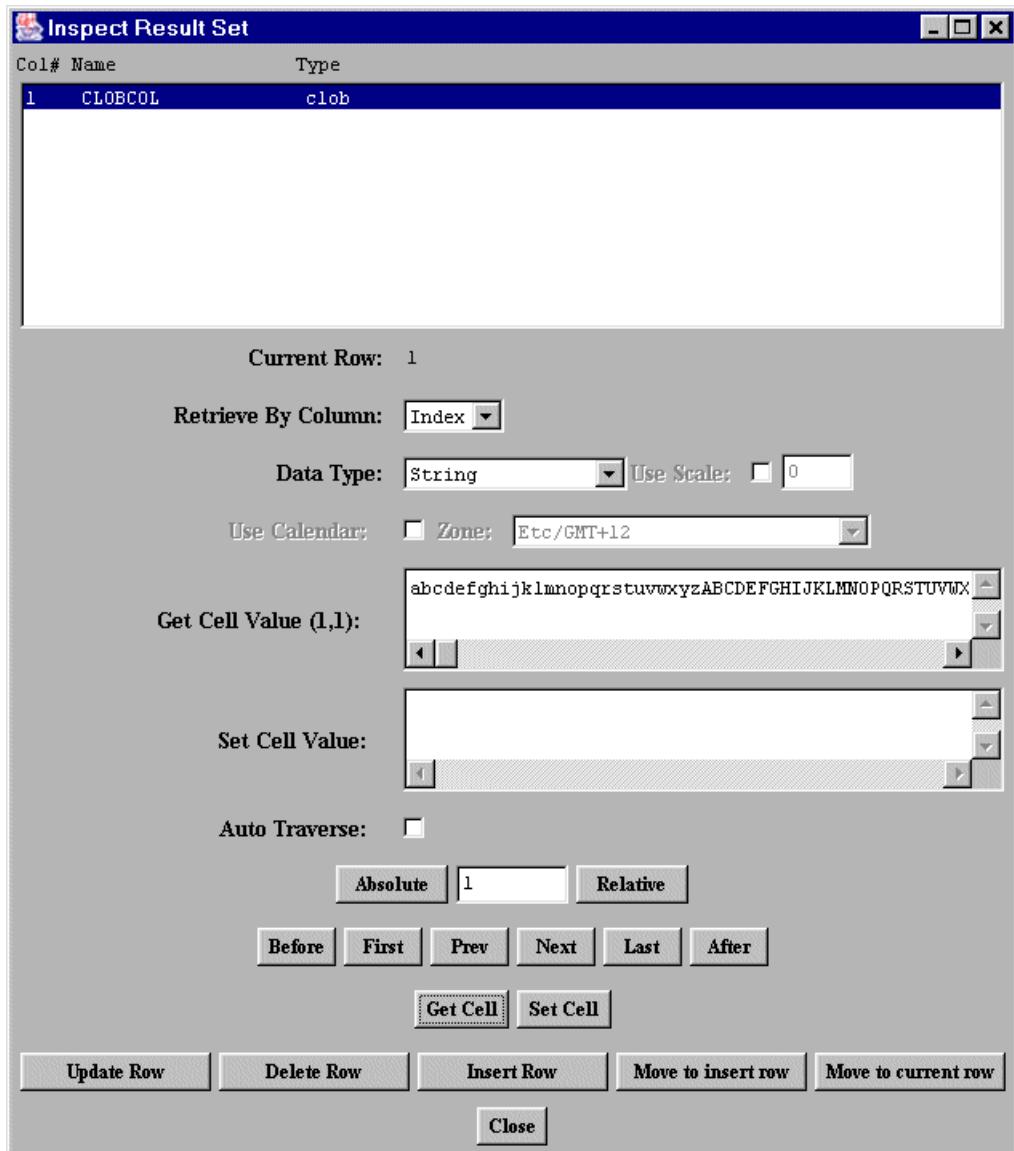
Click **Submit**; then, click **Close**.

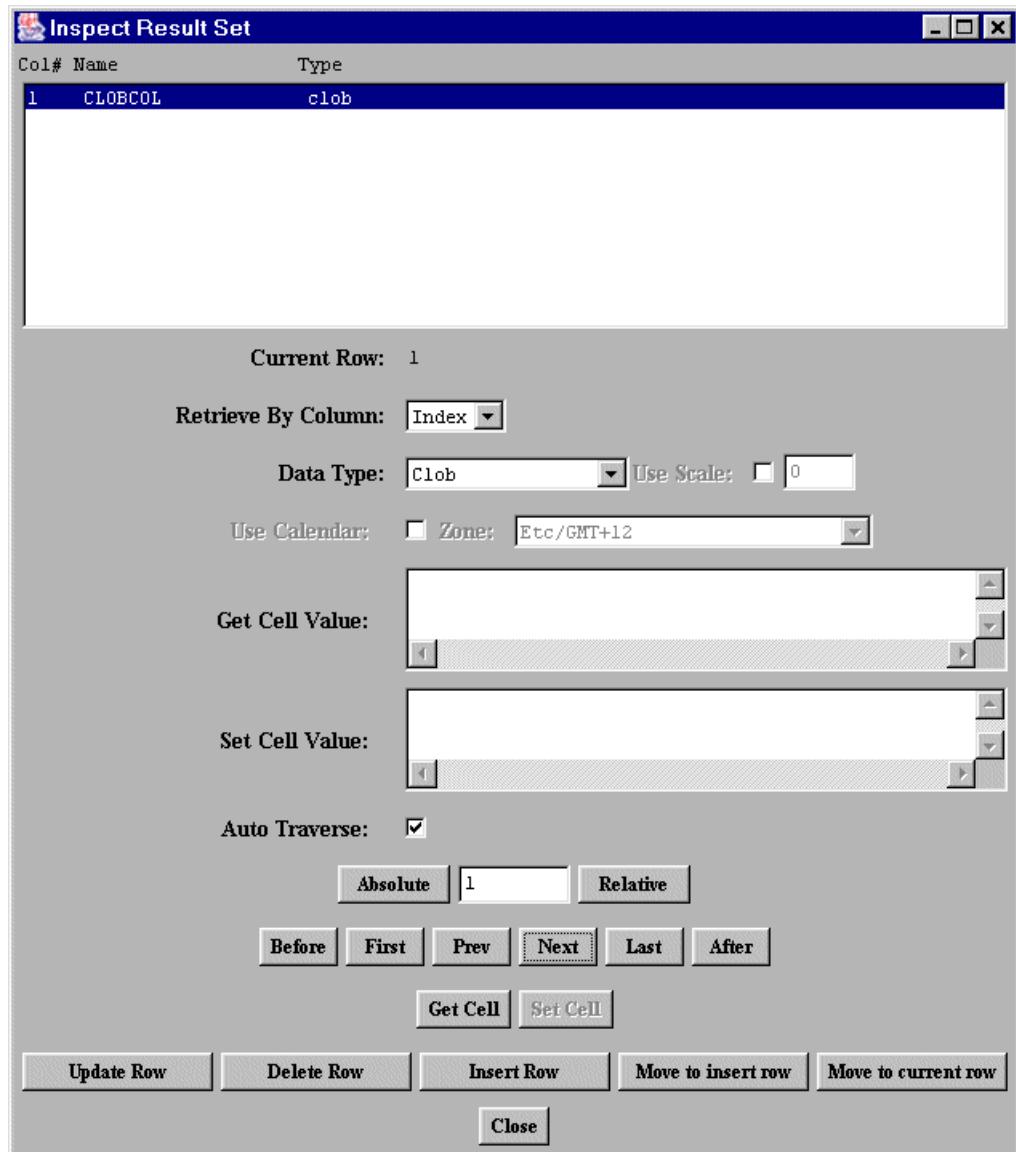
- 4 Select **Results / Inspect Results**. The Inspect Result Set window appears.

5 Click **Next**. Current Row changes to 1.

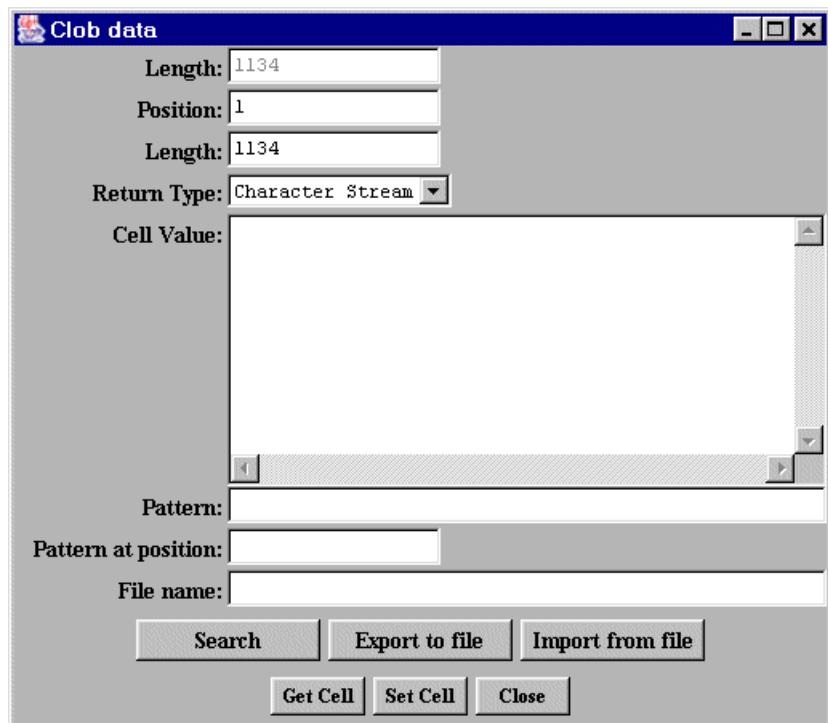


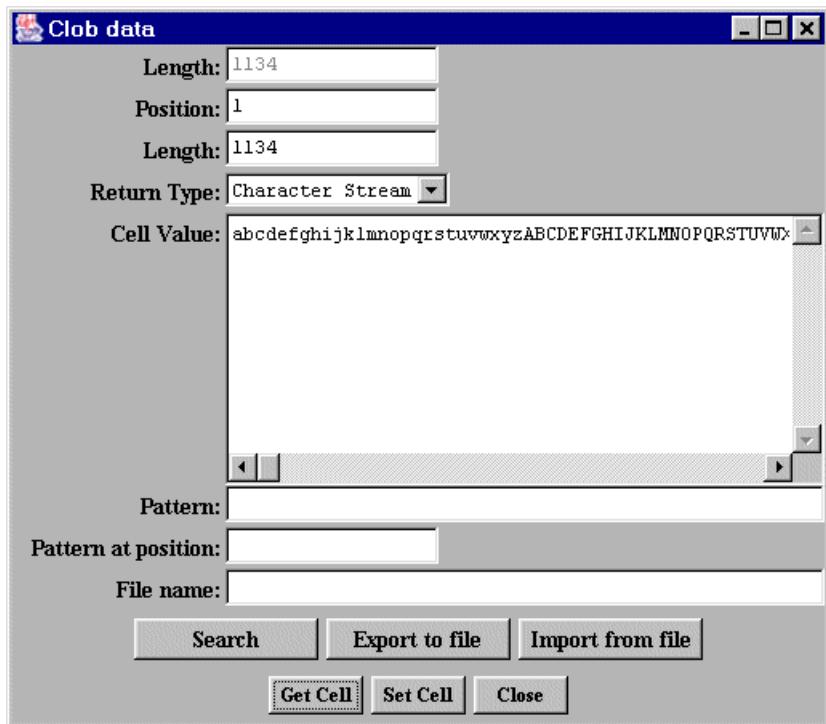
6 Deselect **Auto Traverse**. This disables automatic traversal to the next row.

**7 Click Get Cell.****8 Values are returned in the Get Cell Value field.**

**9 Change the data type to Clob.**

10 Click **Get Cell**. The Blob data window appears.



**11 Click Get Cell.****12 Values are returned in the Cell Value field.**



# H Tracking JDBC Calls with DataDirect Spy™

DataDirect Spy is functionality that is built into all DataDirect Connect *for JDBC* drivers. It is used to log detailed information about calls your driver makes and provide information you can use for troubleshooting. DataDirect Spy provides the following advantages:

- Logging is JDBC 4.0-compliant.
- Logging is consistent, regardless of the DataDirect Connect *for JDBC* driver used.
- All parameters and function results for JDBC calls can be logged.
- Logging works with all DataDirect Connect *for JDBC* drivers.
- Logging can be enabled without changing the application.

When you enable DataDirect Spy for a connection, you can customize logging by setting one or multiple options for DataDirect Spy. For example, you may want to direct logging to a local file on your machine.

Once logging is enabled for a connection, you can turn it on and off at runtime using the `setEnableLogging()` method in the `com.ddtek.jdbc.extensions.ExtLogControl` interface. See [“Troubleshooting Your Application” on page 845](#) for information about using a DataDirect Spy log for troubleshooting.

---

# Enabling DataDirect Spy™

You can enable DataDirect Spy for a connection using either of the following methods:

- Specifying the SpyAttributes connection property for connections using the JDBC Driver Manager. See “[Using the JDBC Driver Manager](#)” on page 784 for instructions.
- Specifying DataDirect Spy attributes using a JDBC data source. See “[Using JDBC Data Sources](#)” on page 785 for instructions.

You can set one or multiple options to customize DataDirect Spy logging. See “[DataDirect Spy™ Attributes](#)” on page 787 for a complete list of supported attributes.

## Using the JDBC Driver Manager

The SpyAttributes connection property allows you to specify a semi-colon separated list of DataDirect Spy attributes (see “[DataDirect Spy™ Attributes](#)” on page 787). The format for the value of the SpyAttributes property is:

*(spy\_attribute[;spy\_attribute]...)*

where *spy\_attribute* is any valid DataDirect Spy attribute. See “[DataDirect Spy™ Attributes](#)” on page 787 for a list of supported attributes.

### Example on Windows:

The following example uses the JDBC Driver Manager to connect to Microsoft SQL Server while enabling DataDirect Spy:

```
Class.forName("com.ddtek.jdbc.sqlserver.SQLServerDriver");  
Connection conn = DriverManager.getConnection  
( "jdbc:datadirect:sqlserver://Server1:1433;User=TEST;  
    Password=secret;SpyAttributes=(log=(file)C:\\temp\\\\spy.log;  
    linelimit=80;logTName=yes;timestamp=yes)");
```

**NOTE:** 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.
```

Using this example, DataDirect Spy would load the DataDirect Connect *for JDBC* SQL Server driver and log all JDBC activity to the spy.log file located in the C:\temp directory

(log=(file)C:\\temp\\spy.log). The spy.log file logs a maximum of 80 characters on each line (linelimit=80) and includes the name of the current thread (logTName=yes) and a timestamp on each line in the log (timestamp=yes).

#### **Example on UNIX:**

The following code example uses the JDBC Driver Manager to connect to DB2 while enabling DataDirect Spy:

```
Class.forName("com.ddtek.jdbc.sqlserver.DB2Driver");
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:db2://Server1:50000;User=TEST;
   Password=secret;SpyAttributes=(log=(file)/tmp/spy.log;
   logTName=yes;timestamp=yes)");
```

Using this example, DataDirect Spy would load the DataDirect Connect *for JDBC* DB2 driver and log all JDBC activity to the spy.log file located in the /tmp directory

(log=(file)/tmp/spy.log). The spy.log file includes the name of the current thread (logTName=yes) and a timestamp on each line in the log (timestamp=yes).

## **Using JDBC Data Sources**

The DataDirect Connect *for JDBC* drivers implement the following JDBC features:

- JNDI for Naming Databases
- Connection Pooling
- JTA

You can use DataDirect Spy to track JDBC calls made by a running application with any of these features. The `com.ddtek.jdbcx.datasource.DriverDataSource` class, where *Driver* is the driver name, supports setting a semi-colon-separated list of DataDirect Spy attributes (see “[DataDirect Spy™ Attributes](#)” on page 787).

See “[Connecting Through Data Sources](#)” on page 48 for more information about configuring data sources.

### Example on Windows:

The following example creates a JDBC data source for the DataDirect Connect *for* JDBC DB2 driver, which enables DataDirect Spy.

```
DB2DataSource sds=new DB2DataSource();
sds.setServerName("Server1");
sds.setPortNumber(1433);
sds.setSpyAttributes("log=(file)C:\\temp\\\\spy.log;logIS=yes;logTName=yes");
Connection conn=sds.getConnection("TEST","secret");
...

```

NOTE: 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;logIS=yes;logTName=yes".
```

Using this example, DataDirect Spy would load the DataDirect Connect *for* JDBC DB2 driver and log all JDBC activity to the spy.log file located in the C:\temp directory (log=(file)C:\\temp\\\\spy.log). In addition to regular JDBC activity, the spy.log file also logs activity on InputStream and Reader objects (logIS=yes). It also includes the name of the current thread (logTName=yes).

### **Example on UNIX:**

The following example creates a JDBC data source for the DataDirect Connect for JDBC Oracle driver, which enables DataDirect Spy.

```
OracleDataSource mds = new OracleDataSource();
mds.setServerName("Server1");
mds.setPortNumber(1521);
mds.setSID("ORCL");...
sds.setSpyAttributes("log=(file)/tmp/spy.log;logTName=yes");
Connection conn=sds.getConnection("TEST","secret");
...
```

Using this example, DataDirect Spy would load the DataDirect Connect for JDBC Oracle driver and log all JDBC activity to the spy.log file located in the /tmp directory (log=(file)/tmp/spy.log). The spy.log file includes the name of the current thread (logTName=yes).

## **DataDirect Spy™ Attributes**

DataDirect Spy supports the following attributes:

<code>log=System.out</code>	Directs logging to the Java output standard, System.out.
<code>log=(file)<i>filename</i></code>	Redirects logging to the file specified by <i>filename</i> .
<code>load=<i>classname</i></code>	Loads the driver specified by <i>classname</i> . For example, com.ddtek.jdbc.db2.DB2Driver loads the DataDirect Connect for JDBC DB2 driver.
<code>linelimit=<i>numberofchars</i></code>	Sets the maximum number of characters that DataDirect Spy will log on any one line. The default is 0 (there is no maximum limit).

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

# I Connection Pool Manager

Connection pooling means that connections are reused rather than created each time a connection is requested. Your application can use connection pooling through the DataDirect Connection Pool Manager.

Connection pooling is performed in the background and does not affect how an application is coded; however, the application must use a `DataSource` object (an object implementing the `DataSource` interface) to obtain a connection instead of using the `DriverManager` class. A class implementing the `DataSource` interface may or may not provide connection pooling. A `DataSource` object registers with a JNDI naming service. Once a `DataSource` object is registered, the application retrieves it from the JNDI naming service in the standard way.

---

## Configuring a Connection Pool

The number of connection pools owned by an application depends on the number of data sources used in the application. You can define attributes of a connection pool for optimal performance and scalability using the methods provided by the DataDirect Connection Pool Manager classes (see “[DataDirect Connection Pool Manager Classes](#)” on page 803).

Some commonly set attributes include:

- Minimum pool size, which is the minimum number of connections be maintained in the pool
- Maximum pool size, which is the maximum number of connections in the pool

- Initial pool size, which is the number of connections created when the connection pool is initialized
- Maximum idle time, which is the amount of time a pooled connection remains idle before it is removed from the connection pool

The Pool Manager implements minimum pool size, maximum pool size, and initial pool size differently depending on whether reauthentication is enabled. See “[Using Reauthentication with the Pool Manager](#)” on page 790 for details.

---

## Using Reauthentication with the Pool Manager

Reauthentication, or the ability to switch a user on a connection, is a useful strategy for minimizing the number of connections that are required in a connection pool. See “[Using Reauthentication](#)” on page 55 for an introduction to reauthentication.

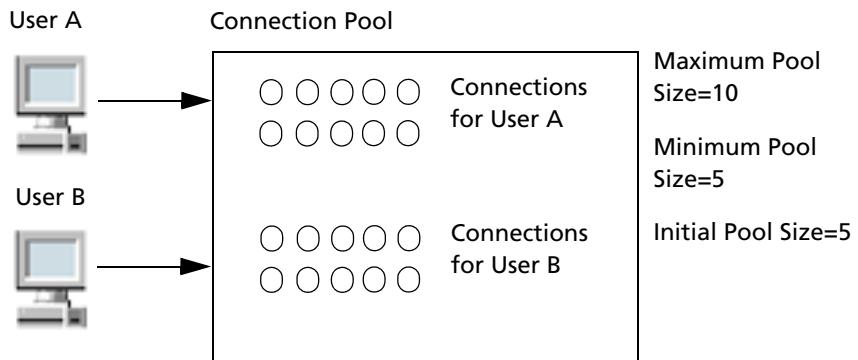
If you are using the DataDirect Connection Pool Manager for connection pooling, you can enable reauthentication in the Pool Manager. By default, reauthentication is disabled. To enable reauthentication, call `setReauthentication(enable)` on the `PooledConnectionDataSource`. To disable reauthentication, call `setReauthentication(disable)`.

The Pool Manager implements the maximum pool size, minimum pool size, and initial pool size attributes differently depending on whether reauthentication is enabled. For example, compare the following examples. In both examples, the maximum pool size is set to a value of 10, the minimum pool size is set to 5, and the initial pool size is set to 5.

Example A shows a connection pool that is configured to work without reauthentication. Two users share connections from the connection pool, but the connections are functionally separated into a group of connections for User A and another group of connections for User B. When User A requests a connection, the Pool Manager assigns an available connection associated with User A. Similarly, if User B requests a connection, the Pool Manager assigns an available connection associated with User B. If a connection is unavailable for a particular user, the Pool Manager creates a new connection for that user, up to a maximum of ten connections for each user. In this case, the maximum number of connections in the pool is 20, or 10 connections for each user.

The Pool Manager implements the minimum pool size and initial pool size in a similar way. The Pool Manager initially populates five connections for User A and five connections for User B, and ensures that, at a minimum, five connections are maintained in the pool for each user.

#### Example A: Connection Pool Without Reauthentication

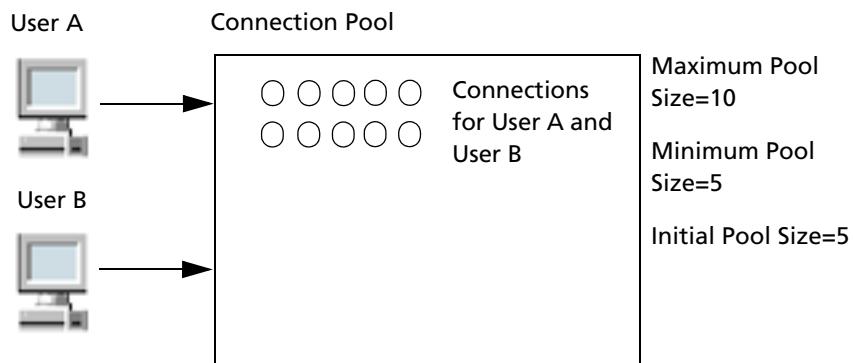


In contrast, Example B shows a connection pool that is configured to work with reauthentication. The Pool Manager treats all connections as one group of connections. When User A requests a connection, the Pool Manager assigns an available connection associated with User A. Similarly, when User B requests a connection, the Pool Manager assigns an available

connection associated with User B. If a connection is unavailable for a particular user, the Pool Manager assigns any available connection to that user, switching the user associated with the connection to the new user. In this case, the maximum number of connections in the pool is 10, regardless of how many users are using the connection pool.

The Pool Manager initially populates the pool with five connections and ensures that, at a minimum, five connections are maintained in the pool for all users.

### Example B: Connection Pool With Reauthentication



---

## Checking the Pool Manager Version

To check the version of your DataDirect Connection Pool Manager, navigate to the directory containing the DataDirect Connection Pool Manager (*install\_dir*/pool manager where *install\_dir* is your DataDirect Connect for JDBC installation directory). At a command prompt, enter the command:

**On Windows:**

```
java -classpath poolmgr_dir\pool.jar com.ddtek.pool.PoolManagerInfo
```

**On UNIX:**

```
java -classpath poolmgr_dir/pool.jar com.ddtek.pool.PoolManagerInfo
```

where *poolmgr\_dir* is the directory containing the DataDirect Connection Pool Manager.

Alternatively, you can obtain the name and version of the DataDirect Connection Pool Manager programmatically by invoking the following static methods:

`com.ddtek.pool.PoolManagerInfo.getPoolManagerName` and  
`com.ddtek.pool.PoolManagerInfo.getPoolManagerVersion`.

---

## Enabling Pool Manager Tracing

You can enable Pool Manager tracing by calling `setTracing(true)` on the `PooledConnectionDataSource` connection. To disable logging, call `setTracing(false)` on the connection.

By default, the DataDirect Connection Pool Manager logs its pool activities to the standard output `System.out`. You can change where the Pool Manager trace information is written by

calling the `setLogWriter()` method on the `PooledConnectionDataSource` connection.

See “[Troubleshooting Connection Pooling](#)” on page 851 for information about using a Pool Manager trace file for troubleshooting.

---

## Creating a Connection Pool

- 1 Create and register with JNDI a DataDirect Connect *for JDBC* driver `DataSource` object. Once created, this `DataSource` object can be used by a connection pool (`PooledConnectionDataSource` object created in [Step 2](#)) to create connections for one or multiple connection pools.
- 2 To create a connection pool, you must create and register with JNDI a `PooledConnectionDataSource` object. A `PooledConnectionDataSource` creates and manages one or multiple connection pools. The `PooledConnectionDataSource` uses the driver `DataSource` object created in [Step 1](#) to create the connections for the connection pool.

## Creating a Driver `DataSource` Object

The following Java code example creates a DataDirect Connect *for JDBC* driver `DataSource` object and registers it with a JNDI naming service.

The `DataSource` class is provided by DataDirect Connect *for JDBC* and is database-dependent. In this example we use Oracle, so the `DataSource` class is `OracleDataSource`. Refer to the specific driver chapters for the name of the `DataSource` class for your driver.

**NOTE:** The `DataSource` class implements the `ConnectionPoolDataSource` interface for pooling in addition to the `DataSource` interface for non-pooling.

```
/******
**
// This code creates a DataDirect Connect for JDBC data source and
// registers it to a JNDI naming service. This DataDirect Connect for
// JDBC data source uses the DataSource implementation provided by
// DataDirect Connect for JDBC Drivers.
//
// This data source registers its name as <jdbc/ConnectSparkyOracle>.
*/
*****
```

// From DataDirect Connect for JDBC:

```
import com.ddtek.jdbcx.oracle.OracleDataSource;

import javax.sql.*;
import java.sql.*;
import javax.naming.*;
import javax.naming.directory.*;
import java.util.Hashtable;

public class OracleDataSourceRegisterJNDI
{
    public static void main(String argv[])
    {
        try {
            // Set up data source reference data for naming context:
            //
            // -----
            // Create a class instance that implements the interface
            // ConnectionPoolDataSource
            OracleDataSource ds = new OracleDataSource();

            ds.setDescription("Oracle on Sparky - Oracle Data Source");
            ds.setServerName("sparky");
            ds.setPortNumber(1521);
            ds.setUser("scott");
            ds.setPassword("test");
```

```
// Set up environment for creating initial context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.fscontext.RefFSContextFactory");
env.put(Context.PROVIDER_URL, "file:c:\\JDBCDataSource");
Context ctx = new InitialContext(env);

// Register the data source to JNDI naming service
ctx.bind("jdbc/ConnectSparkyOracle", ds);

} catch (Exception e) {
    System.out.println(e);
    return;
}
} // Main
} // class OracleDataSourceRegisterJNDI
```

## Creating the Connection Pool

To create a connection pool, you must create and register with JNDI a PooledConnectionDataSource object. The following Java code creates a PooledConnectionDataSource object and registers it with a JNDI naming service.

To specify the driver DataSource object to be used by the connection pool to create pooled connections, set the parameter of the DataSourceName method to the JNDI name of a registered driver DataSource object. For example, the following code sets the parameter of the DataSourceName method to the JNDI name of the driver DataSource object created in ["Creating a Driver DataSource Object" on page 794](#).

The PooledConnectionDataSource class is provided by the DataDirect com.ddtek.pool package. See ["PooledConnectionDataSource" on page 804](#) for a description of the methods supported by the PooledConnectionDataSource class.

```
//*****
// This code creates a data source and registers it to a JNDI naming service.
// This data source uses the PooledConnectionDataSource
// implementation provided by the DataDirect com.ddtek.pool package.
//
// This data source refers to a registered
// DataDirect Connect for JDBC driver DataSource object.
//
// This data source registers its name as <jdbc/SparkyOracle>.
//
//*****
// From the DataDirect connection pooling package:
import com.ddtek.pool.PooledConnectionDataSource;

import javax.sql.*;
import java.sql.*;
import javax.naming.*;
import javax.naming.directory.*;
import java.util.Hashtable;

public class PoolMgrDataSourceRegisterJNDI
{
    public static void main(String argv[])
    {
        try {
            // Set up data source reference data for naming context:
            // -----
            // Create a pooling manager's class instance that implements
            // the interface DataSource
            PooledConnectionDataSource ds = new PooledConnectionDataSource();

            ds.setDescription("Sparky Oracle - Oracle Data Source");

            // Specify a registered driver DataSource object to be used
            // by this data source to create pooled connections
            ds.setDataSourceName("jdbc/ConnectSparkyOracle");
        }
    }
}
```

```
// The pool manager will be initiated with 5 physical connections
ds.setInitialPoolSize(5);

// The pool maintenance thread will make sure that there are 5
// physical connections available
ds.setMinPoolSize(5);

// The pool maintenance thread will check that there are no more
// than 10 physical connections available
ds.setMaxPoolSize(10);

// The pool maintenance thread will wake up and check the pool
// every 20 seconds
ds.setPropertyCycle(20);

// The pool maintenance thread will remove physical connections
// that are inactive for more than 300 seconds
ds.setMaxIdleTime(300);

// Set tracing off because we choose not to see an output listing
// of activities on a connection
ds.setTracing(false);

// Set up environment for creating initial context
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.fscontext.RefFSContextFactory");
env.put(Context.PROVIDER_URL, "file:c:\\JDBCDataSource");
Context ctx = new InitialContext(env);

// Register this data source to the JNDI naming service
ctx.bind("jdbc/SparkyOracle", ds);

catch (Exception e) {
    System.out.println(e);
    return;
}
}

}
```

---

# Connecting Using a Connection Pool

Because an application uses connection pooling by referencing the JNDI name of a registered PooledConnectionDataSource object, code changes are not required for an application to use connection pooling.

The following example shows Java code that looks up and uses the JNDI-registered PooledConnectionDataSource object created in “[Creating the Connection Pool](#)” on page 796.

```
//*****
//  
// Test program to look up and use a JNDI-registered data source.  
//  
// To run the program, specify the JNDI lookup name for the  
// command-line argument, for example:  
//  
//      java TestDataSourceApp <jdbc/SparkyOracle>  
//  
//*****  
import javax.sql.*;  
import java.sql.*;  
import javax.naming.*;  
import java.util.Hashtable;  
  
public class TestDataSourceApp  
{  
    public static void main(String argv[])  
    {  
        String strJNDILookupName = "";  
  
        // Get the JNDI lookup name for a data source  
        int nArgv = argv.length;  
        if (nArgv != 1) {  
            // User does not specify a JNDI lookup name for a data source,  
            System.out.println(  
                "Please specify a JNDI name for your data source");  
            System.exit(0);  
    }  
}
```

```
else {
    strJNDILookupName = argv[0];
}

DataSource ds = null;
Connection con = null;
Context ctx = null;
Hashtable env = null;

long nStartTime, nStopTime, nElapsedTime;

// Set up environment for creating InitialContext object
env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.fscontext.RefFSContextFactory");
env.put(Context.PROVIDER_URL, "file:c:\\JDBCDataSource");

try {
    // Retrieve the DataSource object that is bound to the logical
    // lookup JNDI name
    ctx = new InitialContext(env);
    ds = (DataSource) ctx.lookup(strJNDILookupName);
    catch (NamingException eName) {
        System.out.println("Error looking up " +
                           strJNDILookupName + ": " +eName);
        System.exit(0);
}

int numOfTest = 4;
int [] nCount = {100, 100, 1000, 3000};

for (int i = 0; i < numOfTest; i++) {
    // Log the start time
    nStartTime = System.currentTimeMillis();
    for (int j = 1; j <= nCount[i]; j++) {
        // Get Database Connection
        try {
            con = ds.getConnection("scott", "tiger");
            // Do something with the connection
            // ...
        }
```

```
// Close Database Connection
if (con != null) con.close();
} catch (SQLException eCon) {
    System.out.println("Error getting a connection: " + eCon);
    System.exit(0);
} // try getConnection
} // for j loop

// Log the end time
nStopTime = System.currentTimeMillis();

// Compute elapsed time
nElapsedTime = nStopTime - nStartTime;
System.out.println("Test number " + i + ": looping " +
    nCount[i] + " times");
System.out.println("Elapsed Time: " + nElapsedTime + "\n");
} // for i loop

// All done
System.exit(0);

// Main
} // TestDataSourceApp
```

NOTE: To use non-pooled connections, specify the JNDI name of a registered driver `DataSource` object as the command-line argument when you run the preceding application. For example, the following command specifies the driver `DataSource` object created in ["Creating a Driver `DataSource` Object" on page 794](#):

```
java TestDataSourceApp jdbc/ConnectSparkyOracle
```

## Closing the Connection Pool

To ensure that the connection pool is closed correctly when an application stops running, the application must notify the DataDirect Connection Pool Manager when it stops. If your application runs on JRE 1.3 or higher, notification occurs automatically when the application stops running. If your application runs on JRE 1.2, the application must explicitly notify the pool manager when it stops using the `PooledConnectionDataSource.close()` method as shown in the following code:

```
if (ds instanceof com.ddtek.pool.PooledConnectionDataSource) {  
    com.ddtek.pool.PooledConnectionDataSource pcds =  
        (com.ddtek.pool.PooledConnectionDataSource) ds;  
    pcds.close();  
}
```

The `PooledConnectionDataSource.close()` method also can be used to explicitly close the connection pool while the application is running. For example, if changes are made to the pool configuration using a pool management tool, the `PooledConnectionDataSource.close()` method can be used to force the connection pool to close and re-create the pool using the new configuration values.

---

# DataDirect Connection Pool Manager Classes

This section describes the methods used by the DataDirect Connection Pool Manager classes, `PooledConnectionDataSourceFactory`, `PooledConnectionDataSource`, and `ConnectionPoolMonitor`.

## PooledConnectionDataSourceFactory

This class is used to create a `PooledConnectionDataSource` object from a `Reference` object that is stored in a naming or directory service. The methods in this object are typically invoked by a JNDI service provider; they are not normally invoked by a user application. The following table describes the methods in the `PooledConnectionDataSourceFactory` class.

<b>PooledConnectionDataSourceFactory</b>	
<b>Methods</b>	<b>Description</b>
<code>static Object getObjectInstance(Object refObj, Name name, Context nameCtx, Hashtable env)</code>	Creates a <code>PooledConnectionDataSource</code> object from a <code>Reference</code> object that is stored in a naming or directory service. This is an implementation of the method of the same name defined in the <code>javax.naming.spi.ObjectFactory</code> interface. Refer to the Javadoc for this interface for a description.

## PooledConnectionDataSource

This class is used to create a PooledConnectionDataSource object for use with the DataDirect Connection Pool Manager.

<b>PooledConnectionDataSource</b>	
<b>Methods</b>	<b>Description</b>
<code>void close()</code>	Closes the connection pool. All physical connections in the pool are closed. Any subsequent connection request re-initializes the connection pool.
<code>Connection getConnection()</code>	Obtains a physical connection from the connection pool.
<code>Connection getConnection(String user, String password)</code>	Obtains a physical connection from the connection pool, where <i>user</i> is the user requesting the connection and <i>password</i> is the password for the connection.
<code>String getDataSourceName()</code>	Returns the JNDI name that is used to look up the DataDirect DataSource object referenced by this PooledConnectionDataSource.
<code>String getDescription()</code>	Returns the description of this PooledConnectionDataSource.
<code>String getReauthentication()</code>	Returns whether the Pool Manager is enabled for reauthentication. See “ <a href="#">Using Reauthentication with the Pool Manager</a> ” on page 790 for more information.
<code>int getInitialPoolSize()</code>	Returns the value of the initial pool size, which is the number of physical connections created when the connection pool is initialized.
<code>int getLoginTimeout()</code>	Returns the value of the login timeout, which is the time allowed for the database login to be validated.
<code>PrintWriter getLogWriter()</code>	Returns the writer to which the Pool Manager sends trace information about its activities.

<b>PooledConnectionDataSource (cont.)</b>	
<b>Methods</b>	<b>Description</b>
<code>int getMaxIdleTime()</code>	Returns the value of the maximum idle time, which is the time a physical connection can remain idle in the connection pool before it is removed from the connection pool.
<code>int getMaxPoolSize()</code>	Returns the value of the maximum pool size, which is the maximum number of physical connections allowed within a single pool at any time. When this number is reached, additional connections that would normally be placed in a connection pool are closed.
<code>int getMinPoolSize()</code>	Returns the value of the minimum pool size, which is the minimum number of physical connections be kept open.
<code>int getPropertyCycle()</code>	Returns the value of the property cycle, which specifies how often the pool maintenance thread wakes up and checks the connection pool.
<code>Reference getReference()</code>	Obtains a javax.naming.Reference object for this PooledConnectionDataSource. The Reference object contains all the state information needed to recreate an instance of this data source using the PooledConnectionDataSourceFactory object. This method is typically called by a JNDI service provider when this PooledConnectionDataSource is bound to a JNDI naming service.
<code>public static ConnectionPoolMonitor[ ] getMonitor()</code>	Returns an array of Connection Pool Monitors, one for each connection pool managed by the Pool Manager.

<b>PooledConnectionDataSource (cont.)</b>	
<b>Methods</b>	<b>Description</b>
<code>public static ConnectionPoolMonitor getMonitor(String name)</code>	<p>Returns the name of the Connection Pool Monitor for the connection pool specified by <i>name</i>. If a pool with the specified name cannot be found, this method returns null. The connection pool name has the form:</p> <p><i>jndi_name-user_id</i></p> <p>where <i>jndi_name</i> is the name used for the JNDI lookup of the driver DataSource object from which the pooled connection was obtained and <i>user_id</i> is the user ID used to establish the connections contained in the pool.</p> <p>The following example shows how to return the Connection Pool Monitor for the connection pool that is bound to the JNDI lookup name jdbc/SQLServerPool and connections established by user test04.</p> <pre>DataSource ds = (DataSource) ctx.lookup("jdbc/SQLServerPool"); Connection con = ds.getConnection("test04", "test04"); ConnectionPoolMonitor monitor = PooledConnectionDataSource.getMonitor( "jdbc/SQLServerPool-test04");</pre>
<code>boolean isTracing()</code>	Determines whether tracing is enabled. If enabled, tracing information is sent to the PrintWriter that is passed to the setLogWriter() method or the standard output System.out if the setLogWriter() method is not called.
<code>void setDataSourceName(String dataSourceName)</code>	Sets the JNDI name, which is used to look up the driver DataSource object referenced by this PooledConnectionDataSource. The driver DataSource object bound to this PooledConnectionDataSource, specified by <i>dataSourceName</i> , is not persisted. Any changes made to the PooledConnectionDataSource bound to the specified driver DataSource object affect this PooledConnectionDataSource.

<b>PooledConnectionDataSource (cont.)</b>	
<b>Methods</b>	<b>Description</b>
<code>void setDataSourceName(String dataSourceName, ConnectionPoolDataSource dataSource)</code>	Sets the JNDI name associated with this PooledConnectionDataSource, specified by <i>dataSourceName</i> , and the driver DataSource object, specified by <i>dataSource</i> , referenced by this PooledConnectionDataSource.  The driver DataSource object, specified by <i>dataSource</i> , is persisted with this PooledConnectionDataSource. Changes made to the specified driver DataSource object after this PooledConnectionDataSource is persisted do not affect this PooledConnectionDataSource.
<code>void setDataSourceName(String dataSourceName, Context ctx)</code>	Sets the JNDI name, specified by <i>dataSourceName</i> , and context, specified by <i>ctx</i> , to be used to look up the driver DataSource referenced by this PooledConnectionDataSource.  The JNDI name, specified by <i>dataSourceName</i> , and context, specified by <i>ctx</i> , are used to look up a driver DataSource object. The driver DataSource object is persisted with this PooledConnectionDataSource. Changes made to the driver DataSource after this PooledConnectionDataSource is persisted do not affect this PooledConnectionDataSource.
<code>void setDescription(String description)</code>	Sets the description of the PooledConnectionDataSource, where <i>description</i> is the description.
<code>void setInitialPoolSize(int initialPoolSize)</code>	Sets the value of the initial pool size, which is the number of physical connections created when the connection pool is initialized.
<code>void setLoginTimeout(int i)</code>	Sets the value of the login timeout, where <i>i</i> is the login timeout, which is the time allowed for the database login to be validated.
<code>void setLogWriter(PrintWriter printWriter)</code>	Sets the writer, where <i>printWriter</i> is the writer to which the stream will be printed.

<b>PooledConnectionDataSource</b> (cont.)	
<b>Methods</b>	<b>Description</b>
<code>void setMaxIdleTime(int <i>maxIdleTime</i>)</code>	Sets the value of the maximum idle time, which is the time a physical connection can remain idle in the connection pool before it is removed from the connection pool.
<code>void setMaxPoolSize(int <i>maxPoolSize</i>)</code>	Sets the value of the maximum pool size, which is the maximum number of physical connections allowed within a single pool at any time. When the maximum number is reached, additional connections that would normally be placed in a connection pool are closed.  The Pool Manager implements this attribute differently depending on whether reauthentication is enabled. See <a href="#">"Using Reauthentication with the Pool Manager" on page 790</a> for details.
<code>void setMinPoolSize(int <i>minPoolSize</i>)</code>	Sets the value of the minimum pool size, which is the minimum number of physical connections kept open in the connection pool.
<code>void setPropertyCycle(int <i>propertyCycle</i>)</code>	Sets the value of the property cycle, which specifies how often the pool maintenance thread wakes up and checks the connection pool.
<code>void setReauthentication(String <i>value</i>)</code>	Enables and disables reauthentication for the Pool Manager. To enable reauthentication, use <code>setReauthentication(enable)</code> . To disable reauthentication, use <code>setReauthentication(disable)</code> . See <a href="#">"Using Reauthentication with the Pool Manager" on page 790</a> for more information.
<code>void setTracing(boolean <i>value</i>)</code>	Enables or disables tracing. If set to true, tracing is enabled; if false, it is disabled. If enabled, tracing information is sent to the PrintWriter that is passed to the <code>setLogWriter()</code> method or the standard output <code>System.out</code> if the <code>setLogWriter()</code> method is not called.

## ConnectionPoolMonitor Class

This class is used to return information that is useful for monitoring the status of your connection pools.

<b>ConnectionPoolMonitor</b>	
<b>Methods</b>	<b>Description</b>
<code>public String getName()</code>	Returns the name of the connection pool associated with the monitor. The connection pool name has the form: <i>jndi_name-user_id</i> where <i>jndi_name</i> is the name used for the JNDI lookup of the PooledConnectionDataSource object from which the pooled connection was obtained and <i>user_id</i> is the user ID used to establish the connections contained in the pool.
<code>public int getNumActive()</code>	Returns the number of connections that have been checked out of the pool and are currently in use.
<code>public int getNumAvailable()</code>	Returns the number of connections that are idle in the pool (available connections).
<code>public int getInitialPoolSize()</code>	Returns the initial size of the connection pool (the number of available connections in the pool when the pool was first created).
<code>public int getMaxPoolSize()</code>	Returns the maximum number of available connection in the connection pool. If the number of available connections exceeds this value, the Pool Manager removes one or multiple available connections from the pool.
<code>public int getMinPoolSize()</code>	Returns the minimum number of available connections in the connection pool. When the number of available connections is lower than this value, the Pool Manager creates additional connections and makes them available.
<code>public int getPoolSize()</code>	Returns the current size of the connection pool, which is the total of active connections and available connections.



# J Statement Pool Monitor

You can use the DataDirect Statement Pool Monitor to add statements to load and save the contents of a connection's statement pool as well as generate information to help you troubleshoot statement pooling performance. The Statement Pool Monitor is an integrated component of the DataDirect Connect *for JDBC* drivers. It is implemented as a Java Management Extensions (JMX) MBean. Applications can access the Statement Pool Monitor using the standard JMX API or using methods in the DataDirect-specific ExtConnection interface. Because the Statement Pool Monitor is implemented as a JMX MBean, it can easily be used by any JMX-compliant tool, such as JConsole.

---

## Enabling Statement Pooling

To enable statement pooling, set the driver MaxPooledStatements (or MaxStatements) connection property to a positive integer. For example, the following connection URL enables statement pooling for the SQL Server driver and sets the maximum number of pooled prepared statements for this connection to 20.

```
jdbc:datadirect:sqlserver://server1:1433;User=test;  
Password=secret;MaxPooledStatements=20
```

---

# Accessing the Statement Pool Monitor

Your application cannot access the Statement Pool Monitor unless you enable statement pooling. See “[Enabling Statement Pooling](#)” on page 811 for more information.

Applications and tools can access the Statement Pool Monitor in either of the following ways:

- Using JMX API methods
- Using DataDirect-specific methods to access the Statement Pool Monitor using a JDBC connection

## Using JMX

DataDirect Connect for JDBC registers a single Statement Pool Monitor for each statement pool with the default MBean server of the JVM. The registered MBean name has the following form, where *monitor\_name* is the string returned by the ExtStatementPoolMonitor.getName() method:

```
com.ddtek.jdbc.type=StatementPoolMonitor,name=monitor_name
```

To return information about the statement pool, retrieve the names of all MBeans that are registered with the com.ddtek.jdbc domain and search through the list for the StatementPoolMonitor type attribute. The following code shows how to use the standard JMX API calls to return the state of all active statement pools in the JVM:

```
private void run(String[] args) {  
    if (args.length < 2) {  
  
        System.out.println("Not enough arguments supplied");  
        System.out.println("Usage: " + "ShowStatementPoolInfo hostname port");  
    }  
}
```

```
String hostname = args[0];
String port = args[1];

JMXServiceURL url = null;
JMXConnector connector = null;
MBeanServerConnection server = null;

try {
    url = new JMXServiceURL("service:jmx:rmi:///jndi/rmi://" +
                           hostname + ":" + port + "/jmxrmi");
    connector = JMXConnectorFactory.connect(url);
    server = connector.getMBeanServerConnection();

    System.out.println("Connected to JMX MBean Server at " +
                       args[0] + ":" + args[1]);

    // Get the MBeans that have been registered with the
    // com.ddtek.jdbc domain.
    ObjectName ddBMeans = new ObjectName("com.ddtek.jdbc:*");
    Set<ObjectName> mbeans = server.queryNames(ddBMeans, null);

    // For each statement pool monitor MBean, display statistics and
    // contents of the statement pool monitored by that MBean
    for (ObjectName name: mbeans) {

        if (name.getDomain().equals("com.ddtek.jdbc") &&
            name.getKeyProperty("type")
            .equals("StatementPoolMonitor")) {

            System.out.println("Statement Pool - " +
                               server.getAttribute(name, "Name"));
            System.out.println("Max Size: " +
                               server.getAttribute(name, "MaxSize"));
            System.out.println("Current Size: " +
                               server.getAttribute(name, "CurrentSize"));
            System.out.println("Hit Count: " +
                               server.getAttribute(name, "HitCount"));
            System.out.println("Miss Count: " +
                               server.getAttribute(name, "MissCount"));
            System.out.println("Statements:");
        }
    }
}
```

```
Object[] params = new Object[3];
params[0] = new Integer(-1);
params[1] = new Integer(-1);
params[2] = new Integer(-1);

String[] types = new String[3];
types[0] = "int";
types[1] = "int";
types[2] = "int";

ArrayList<String>statements = (ArrayList<String>)
    server.invoke(name,
                  "poolEntries",
                  params,
                  types);

for (String stmt : statements) {

    int index = stmt.indexOf(";");
    System.out.println("    " + stmt.substring(0, index));
}

}

}

}

catch (Throwable except) {
    System.out.println("ERROR: " + except);
}
}
```

**NOTE:** Because JMX is not supported by J2SE 1.4, applications running in these JVMs must use the DataDirect-specific method for accessing the Statement Pool Monitor (see “[Using DataDirect-Specific Methods](#)” on page 815).

## Using DataDirect-Specific Methods

The ExtConnection.getStatementPoolMonitor() method returns an ExtStatementPoolMonitor object for the statement pool associated with the connection. This method is provided by the ExtConnection interface in the com.ddtek.jdbc.extensions package. If the connection does not have a statement pool, the method returns null.

Once you have an ExtStatementPoolMonitor object, you can use the poolEntries() method of the ExtStatementPoolMonitorMBean interface implemented by the ExtStatementPoolMonitor to return a list of statements in the statement pool as an array.

### *Using the poolEntries() Method*

Using the poolEntries() method, your application can return all statements in the pool or filter the list based on the following criteria:

- Statement type (prepared statement or callable statement)
- Result set type (forward only, scroll insensitive, or scroll sensitive)
- Concurrency type of the result set (read only and updateable)

[Table J-1](#) lists the parameters and the valid values supported by the poolEntries() method.

**Table J-1. poolEntries() Parameters**

Parameter	Value	Description
statementType	ExtStatementPoolMonitor.TYPE_PREPARED_STATEMENT	Returns only prepared statements
	ExtStatementPoolMonitor.TYPE_CALLABLE_STATEMENT	Returns only callable statements
	-1	Returns all statements regardless of statement type
resultSetType	ResultSet.TYPE_FORWARD_ONLY	Returns only statements with forward-only result sets
	ResultSet.TYPE_SCROLL_INSENSITIVE	Returns only statements with scroll insensitive result sets
	ResultSet.TYPE_SCROLL_SENSITIVE	Returns only statements with scroll sensitive result sets
	-1	Returns statements regardless of result set type

**Table J-1. poolEntries() Parameters (cont.)**

Parameter	Value	Description
resultSetConcurrency	ResultSet.CONCUR_READ_ONLY	Returns only statements with a read-only result set concurrency
	ResultSet.CONCUR_UPDATABLE	Returns only statements with an updateable result set concurrency
	-1	Returns statements regardless of result set concurrency type

The result of the poolEntries() method is an array that contains a String entry for each statement in the statement pool using the format:

```
SQL_TEXT=[SQL_text];STATEMENT_TYPE=
TYPE_PREPARED_STATEMENT|TYPE_CALLABLE_STATEMENT;RESULTSET_TYPE=
TYPE_FORWARD_ONLY|TYPE_SCROLL_INSENSITIVE|TYPE_SCROLL_SENSITIVE;
RESULTSET_CONCURRENCY=CONCUR_READ_ONLY|CONCUR_UPDATABLE;
AUTOGENERATEDKEYSREQUESTED=true|false;REQUESTEDKEYCOLUMNS=
comma-separated_list
```

where *SQL\_text* is the SQL text of the statement and *comma-separated\_list* is a list of column names that will be returned as generated keys:

For example:

```
SQL_TEXT=[INSERT INTO emp(id, name) VALUES(99, ?)];STATEMENT_TYPE=Prepared
Statement;RESULTSET_TYPE=Forward Only;RESULTSET_CONCURRENCY=Read Only;
AUTOGENERATEDKEYSREQUESTED=false;REQUESTEDKEYCOLUMNS=id,name
```

## ***Example: Generating a List of Statements in the Statement Pool***

The following code shows how to return an ExtStatementPoolMonitor object using a connection and generate a list of statements in the statement pool associated with the connection.

```
private void run(String[] args) {  
  
    Connection con = null;  
    PreparedStatement prepStmt = null;  
    String sql = null;  
  
    try {  
  
        // Create the connection and enable statement pooling  
        Class.forName("com.ddtek.jdbc.sqlserver.SQLServerDriver");  
        con = DriverManager.getConnection(  
            "jdbc:datadirect:sqlserver://SMITH:1440;" +  
            "maxPooledStatements=10", "test", "test");  
  
        // Prepare a couple of statements  
        sql = "INSERT INTO employees (id, name) VALUES(?, ?)";  
        prepStmt = con.prepareStatement(sql);  
        prepStmt.close();  
  
        sql = "SELECT name FROM employees WHERE id = ?";  
        prepStmt = con.prepareStatement(sql);  
        prepStmt.close();  
  
        ExtStatementPoolMonitor monitor =  
            ((ExtConnection) con).getStatementPoolMonitor();  
  
        System.out.println("Statement Pool - " + monitor.getName());  
        System.out.println("Max Size:      " + monitor.getMaxSize());  
        System.out.println("Current Size: " + monitor.getCurrentSize());  
        System.out.println("Hit Count:     " + monitor.getHitCount());  
        System.out.println("Miss Count:    " + monitor.getMissCount());  
        System.out.println("Statements:");  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
        if (prepStmt != null) {  
            try {  
                prepStmt.close();  
            } catch (SQLException e) {  
                e.printStackTrace();  
            }  
        }  
        if (con != null) {  
            try {  
                con.close();  
            } catch (SQLException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

```

ArrayList statements = monitor.poolEntries(-1, -1, -1);
Iterator itr = statements.iterator();
while (itr.hasNext()) {

    String entry = (String)itr.next();
    System.out.println(entry);
}
}
catch (Throwable except) {

    System.out.println("ERROR: " + except);
}
finally {

    if (con != null) {

        try {
            con.close();
        }
        catch (SQLException except) {}
    }
}
}

```

In the previous code example, the `PoolEntries()` method returns all statements in the statement pool regardless of statement type, result set cursor type, and concurrency type by specifying the value `-1` for each parameter as shown in the following code:

```
ArrayList statements = monitor.poolEntries(-1, -1, -1);
```

We could have easily filtered the list of statements to return only prepared statements that have a forward-only result set with a concurrency type of updateable using the following code:

```
ArrayList statements =
monitor.poolEntries(ExtStatementPoolMonitor.TYPE_PREPARED_STATEMENT,
ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_UPDATABLE);
```

---

# Importing Statements into a Statement Pool

When importing statements into a statement pool, for each statement entry in the export file, a statement is added to the statement pool provided a statement with the same SQL text and statement attributes does not already exist in the statement pool. Existing statements in the pool that correspond to a statement entry are kept in the pool unless the addition of new statements causes the number of statements to exceed the maximum pool size. In this case, the driver closes and discards some statements until the pool size is shrunk to the maximum pool size.

For example, if the maximum number of statements allowed for a statement pool is 10 and the number of statements to be imported is 20, only the last 10 imported statements are placed in the statement pool. The other statements are created, closed, and discarded. Importing more statements than the maximum number of statements allowed in the statement pool can negatively affect performance because the driver unnecessarily creates some statements that are never placed in the pool.

**To import statements into a statement pool:**

- 1 Create a statement pool export file. See ["Statement Pool Export File Example" on page 859](#) for an example of a statement pool export file.

NOTE: The easiest way to create a statement pool export file is to generate an export file from the statement pool associated with the connection as described in ["Generating a Statement Pool Export File" on page 823](#).

- 2 Edit the export file to contain statements to be added to the statement pool.

- 3 Import the contents of the export file to the statement pool using either of the following methods to specify the path and file name of the export file:

- Use the `ImportStatementPool` property. For example:

```
jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc;User=test;  
Password=secret;ImportStatementPool=C:\\statement_pooling\\\\stmt_export.txt
```

- Use the `importStatements()` method of the `ExtStatementPoolMonitorMBean` interface. For example:

```
ExtStatementPoolMonitor monitor =  
((ExtConnection) con).getStatementPoolMonitor().  
monitor.importStatements("C:\\statement_pooling\\\\stmt_export.txt");
```

---

## Clearing All Statements in a Statement Pool

To close and discard all statements in a statement pool, use the `emptyPool()` method of the `ExtStatementPoolMonitorMBean` interface. For example:

```
ExtStatementPoolMonitor monitor =  
((ExtConnection) con).getStatementPoolMonitor().emptyPool();
```

---

## Freezing and Unfreezing the Statement Pool

Freezing the statement pool restricts the statements in the pool to those that were in the pool at the time the pool was frozen. For example, perhaps you have a core set of statements that you do not want replaced by new statements when your core statements are closed. You can freeze the pool using the `setFrozen()` method:

```
ExtStatementPoolMonitor monitor =
((ExtConnection) con).getStatementPoolMonitor().setFrozen(true);
```

Similarly, you can use the same method to unfreeze the statement pool:

```
ExtStatementPoolMonitor monitor =
((ExtConnection) con).getStatementPoolMonitor().setFrozen(false);
```

When the statement pool is frozen, your application can still clear the pool and import statements into the pool. In addition, if your application is using Java SE 6, you can use the `Statement.setPoolable()` method to add or remove single statements from the pool regardless of the pool's frozen state, assuming the pool is not full. If the pool is frozen and the number of statements in the pool is the maximum, no statements can be added to the pool.

To determine if a pool is frozen, use the `isFrozen()` method.

---

# Generating a Statement Pool Export File

You may want to generate an export file in the following circumstances:

- To import statements to the statement pool as described in ["Importing Statements into a Statement Pool" on page 820](#). You can create an export file, edit its contents, and import the file into the statement pool to import statements to the pool.
- To examine the characteristics of the statements in the statement pool to help you troubleshoot statement pool performance.

To generate a statement pool export file, use the `exportStatements()` method of the `ExtStatementPoolMonitorMBean` interface. For example, the following code exports the contents of the statement pool associated with the connection to a file named `stmt_export`:

```
ExtStatementPoolMonitor monitor =
((ExtConnection) con).getStatementPoolMonitor().
exportStatements("stmt_export.txt");
```

See ["Statement Pool Export File Example" on page 859](#) for information on interpreting the contents of an export file.

---

# DataDirect Statement Pool Monitor Classes

This section describes the methods used by the DataDirect Statement Pool Monitor classes and interfaces.

## ExtStatementPoolMonitor Class

This class is used to control and monitor a single statement pool. This class implements the ExtStatementPoolMonitorMBean interface. See “[ExtStatementPoolMonitorMBean Interface](#)” on [page 824](#) for a description of the methods in the ExtStatementPoolMonitorMBean interface.

## ExtStatementPoolMonitorMBean Interface

ExtStatementPoolMonitorMBean	
Methods	Description
String getName()	Returns the name of a Statement Pool Monitor instance associated with the connection. The name is comprised of the name of the driver that established the connection, and the name and port of the server to which the Statement Pool Monitor is connected, and the MBean ID of the connection. For example: <code>jdbc_datadirect_sqlserver_SMITH_1440_0</code>
int getCurrentSize()	Returns the total number of statements cached in the statement pool.

<b>ExtStatementPoolMonitorMBean</b>	
<b>Methods</b>	<b>Description</b>
<code>long getHitCount()</code>	<p>Returns the hit count for the statement pool. The hit count is the number of times a lookup is performed for a statement that results in a cache hit. A cache hit occurs when the Statement Pool Monitor successfully finds a statement in the pool with the same SQL text, statement type, result set type, result set concurrency, and requested generated key information.</p> <p>This method is useful to determine if your workload is using the statement pool effectively. For example, if the hit count is low, the statement pool is probably not being used to its best advantage.</p>
<code>long getMissCount()</code>	<p>Returns the miss count for the statement pool. The miss count is the number of times a lookup is performed for a statement that fails to result in a cache hit. A cache hit occurs when the Statement Pool Monitor successfully finds a statement in the pool with the same SQL text, statement type, result set type, result set concurrency, and requested generated key information.</p> <p>This method is useful to determine if your workload is using the statement pool effectively. For example, if the miss count is high, the statement pool is probably not being used to its best advantage.</p>
<code>int getMaxSize()</code>	Returns the maximum number of statements that can be stored in the statement pool.
<code>int setMaxSize(int value)</code>	Changes the maximum number of statements that can be stored in the statement pool to the specified value. See “ <a href="#">Importing Statements into a Statement Pool</a> ” on page 820 for more information about how the maximum size affects loading statements.

<b>ExtStatementPoolMonitorMBean</b>	
<b>Methods</b>	<b>Description</b>
void emptyPool()	Closes and discards all the statements in the statement pool. See “ <a href="#">Clearing All Statements in a Statement Pool</a> ” on page 821 for more information.
void resetCounts()	Resets the hit and miss counts to zero (0). See <a href="#">long getHitCount()</a> and <a href="#">long getMissCount()</a> for more information.
ArrayList poolEntries(int statementType, int resultSetType, int resultSetConcurrency)	Returns a list of statements in the pool. The list is an array that contains a String entry for each statement in the statement pool. See “ <a href="#">Using the poolEntries() Method</a> ” on page 815 for more information.
void exportStatements(File file_object)	Exports statements from the statement pool into the specified file. The file format contains an entry for each statement in the statement pool. See “ <a href="#">Generating a Statement Pool Export File</a> ” on page 823 for more information.
void exportStatements(String file_name)	Exports statements from statement pool into the specified file. The file format contains an entry for each statement in the statement pool. See “ <a href="#">Generating a Statement Pool Export File</a> ” on page 823 for more information.
void importStatements(File file_object)	Imports statements from the specified File object into the statement pool. See “ <a href="#">Importing Statements into a Statement Pool</a> ” on page 820 for more information.
void importStatements(String file_name)	Imports statements from the specified file into the statement pool. See “ <a href="#">Importing Statements into a Statement Pool</a> ” on page 820 for more information.

<b>ExtStatementPoolMonitorMBean</b>	
<b>Methods</b>	<b>Description</b>
boolean isFrozen()	Returns whether the state of the statement pool is frozen. When the statement pool is frozen, the statements that can be stored in the pool are restricted to those that were in the pool at the time the pool was frozen. Freezing a pool is useful if you have a core set of statements that you do not want replaced by other statements when the core statements are closed. See <a href="#">"Freezing and Unfreezing the Statement Pool" on page 822</a> for more information.
void setFrozen(boolean)	setFrozen(true) freezes the statement pool. setFrozen(false) unfreezes the statement pool. When the statement pool is frozen, the statements that can be stored in the pool are restricted to those that were in the pool at the time the pool was frozen. Freezing a pool is useful if you have a core set of statements that you do not want replaced by other statements when the core statements are closed. See <a href="#">"Freezing and Unfreezing the Statement Pool" on page 822</a> for more information.



# K Using DataDirect Bulk Load

The DB2, Oracle, SQL Server, and Sybase drivers support DataDirect Bulk Load, a feature that allows your application to send large numbers of rows of data to the database in a continuous stream instead of in numerous smaller database protocol packets. Similar to batch operations, performance improves because far fewer network round trips are required. Bulk load bypasses the data parsing usually done by the database, providing an additional performance gain over batch operations.

## NOTES:

- The Oracle driver supports bulk load only for Oracle 9*i* R2 and higher.
- Because a bulk load operation may bypass data integrity checks, your application must ensure that the data it is transferring does not violate integrity constraints in the database. For example, suppose you are bulk loading data into a database table and some of that data duplicates data stored as a primary key, which must be unique. The driver will not throw an exception to alert you to the error; your application must provide its own data integrity checks.

**IMPORTANT:** DataDirect Bulk Load requires a licensed installation of the drivers. If the drivers are installed with an evaluation license, the bulk load feature is available for prototyping with your applications, but with limited scope. Contact your sales representative or DataDirect SupportLink for further information.

You can perform bulk load operations in either of the following ways:

- Create a DDBulkLoad object and use the methods provided by the DDBulkLoad interface in the com.ddtek.jdbc.extensions package for bulk load. You may want to use this method if you are developing a new application that needs to perform bulk load operations. See “[Using a DDBulkLoad Object](#)” on page 830.
- Configure the driver to use the native bulk load protocol in the database when it encounters a request to execute a batch insert instead of the batch mechanism. This method allows existing applications to take advantage of bulk load without requiring changes to the application code. See “[Using Bulk Load for Batch Inserts](#)” on page 837.

---

## Using a DDBulkLoad Object

To create a DDBulkLoad object, create an instance of the DDBulkLoadFactory class as shown in the following code:

```
import com.ddtek.jdbc.extensions.*  
// Get Database Connection  
Connection con = DriverManager.getConnection  
  ("jdbc:datadirect:oracle://server3:1521;ServiceName=ORCL;User=test;  
  Password=secret");  
  
// Get a DDBulkLoad object  
DDBulkLoad bulkLoad =  
  com.ddtek.jdbc.extensions.DDBulkLoadFactory.getInstance(con);
```

Once your application has created a DDBulkLoad object, it can use the DDBulkLoad methods to provide the driver with the following types of instructions:

- Where to obtain the data to load. The driver can obtain data from either of the following sources:
  - ResultSet object generated from a query.
  - Comma-separated value (CSV) file. The driver can export data from a database table or ResultSet object into a CSV file, or the driver can use a CSV file created by another DataDirect Connect product.
- Where the data will be loaded (database table name).

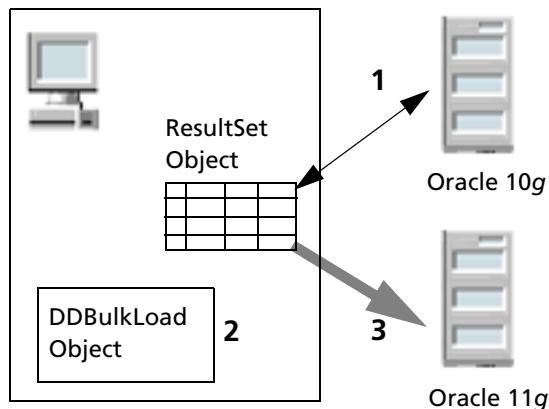
See “[DDBulkLoad Interface](#)” on page 603 for a description of the methods available for bulk load.

For example, suppose after upgrading from an Oracle 10g database to an Oracle 11g database, you need to migrate data from the old database to the new database. [Figure K-1](#) shows how your application would load the data from a ResultSet object created from a query. This scenario assumes that the Oracle driver is connected to both databases.

---

**Figure K-1. Loading Data Using a ResultSet Object**

---



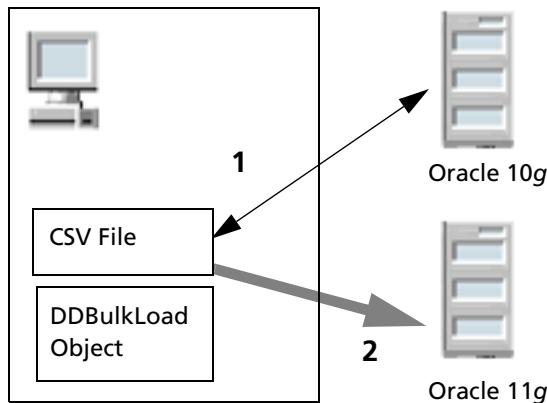
- 1 The application executes a query to the Oracle 10g database in the normal way and retrieves the results in a ResultSet object.
  - 2 The application creates a DDBulkLoad object and instructs the driver to load data from the ResultSet object into the Oracle 11g database.
  - 3 The driver loads the data in a continuous stream from the ResultSet object into the table in the Oracle 11g database.
- 

Using the same scenario, [Figure K-2](#) shows how you would load the data into the Oracle 11g database using a CSV file instead of a ResultSet object. The CSV file is created by exporting data from a database table in the Oracle 10g database.

---

**Figure K-2. Loading Data Using a CSV File**

---



- 1 The application creates a DDBulkLoad object and specifies that the driver export the data from a table in the Oracle 10g database into a CSV file.
  - 2 The application instructs the driver to load data from the CSV file into the Oracle 11g database. The driver loads the data in a continuous stream from the CSV file into the table in the Oracle 11g database.
-

## Exporting Data to a CSV File

Using the methods provided for bulk load (see “[DDBulkLoad Interface](#)” on page 603), the driver can export data from either of the following sources into a CSV file:

- Database table. Use the `setTableName()` method, specifying the table name. Then, use the `export()` method, specifying the CSV file. For example, to export data from a database table named GBMAXTABLE to a file named tmp.csv, you would specify:

```
bulkLoad.setTableName( "GBMAXTABLE" );
```

and

```
bulkLoad.export( "tmp.csv" );
```

**NOTE:** Alternatively, you can create a file reference to the CSV file, and use the `export()` method to specify the file reference:

```
File csvFile = new File ( "tmp.csv" );
bulkLoad.export(csvFile);
```

- `ResultSet` object. Create a file reference to a CSV file, and use the `export()` method, specifying the `ResultSet` object and the file reference. For example, to export data from a `ResultSet` object named `rs` to file named tmp.csv, you would specify:

```
File csvFile = new File ( "tmp.csv" );
bulkLoad.export(rs, csvFile);
```

If the CSV file does not already exist, the driver creates it when the `export()` method is executed. The driver also creates a bulk load configuration file, which describes the structure of the CSV file. For more information about using CSV files, see “[Using CSV Files](#)” on page 838.

## Loading Data

Using the methods provided for bulk load (see “[DDBulkLoad Interface](#)” on page 603), the driver can load data into a database table from either of the following sources:

- CSV file. Use the `setTableName()` method to specify the database table to load the data. Then, use the `load()` method, specifying the CSV file. For example, to load data from a file named `tmp.csv` into a database table named `GBMAXTABLE`, you would specify:

```
bulkLoad.setTableName( "GBMAXTABLE" );
```

and

```
bulkLoad.load( "tmp.csv" );
```

NOTE: Alternatively, you can create a file reference to the CSV file, and use the `load()` method to specify the file reference:

```
File csvFile = new File ( "tmp.csv" );
bulkLoad.load(csvFile);
```

- `ResultSet` object. Use the `setTableName()` method to specify the database table to load the data. Then, use the `load()` method, specifying the `ResultSet` object. For example, to load data from a `ResultSet` object named `rs` into a database table named `GBMAXTABLE`, you would specify:

```
bulkLoad.setTableName( "GBMAXTABLE" );
```

and

```
bulkLoad.load(rs);
```

Use the `BulkLoadBatchSize` property to specify the number of rows the driver loads 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.

## Logging

If logging is enabled for bulk load, a log file records information for each bulk load operation. Logging is enabled by specifying a file name and location for the log file using the `setLogFile()` method.

The log file records the following types of information about each bulk load operation:

- Total number of rows that were read
- Total number of rows that successfully loaded
- Total number of rows that failed to load

For example, the following log file shows that of 11 rows that were read, all successfully loaded:

```
/*----- Load Started: <Feb 25, 2009 11:20:09 AM EST>-----*/  
Total number of rows read 11  
Total number of rows successfully loaded 11  
Total number of rows that failed to load 0
```

### Example A: Enabling Logging on Windows

To enable logging using a log file named `bulk_load.log` located in the `C:\temp` directory, specify:

```
bulkLoad.setLogFile(C:\\temp\\bulk_load.log)
```

NOTE: 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: `C:\\temp\\bulk_load.log`.

### Example B: Enabling Logging on UNIX/Linux

To enable logging using a log file named `bulk_load.log` located in the `/tmp` directory, specify:

```
bulkLoad.setLogFile( /tmp/bulk_load.log)
```

---

# Using Bulk Load for Batch Inserts

When the `EnableBulkLoad` property is set to true, the driver uses the native bulk load protocol in the database for batch inserts instead of the batch mechanism. For example, if you set the `EnableBulkLoad` property to true, the driver would use bulk load for the following batch insert request.

```
PreparedStatement ps = conn.prepareStatement(  
    "INSERT INTO employees VALUES (?, ?, ?)");  
for (n = 0; n < 100; n++) {  
    ps.setString(name[n]);  
    ps.setLong(id[n]);  
    ps.setInt(salary[n]);  
    ps.addBatch();  
}  
ps.executeBatch();
```

In some cases, the driver may not be able to use bulk load because of restrictions enforced by the bulk load protocol in the database and will downgrade to a batch mechanism. For example, if the data being loaded has a data type that is not supported by the bulk load protocol, the driver cannot use bulk load, but will use the batch mechanism instead.

Use the `BulkLoadBatchSize` property to specify the number of rows the driver loads 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.

---

## Using CSV Files

As described in “[Exporting Data to a CSV File](#)” on page 834, the driver can create a CSV file by exporting data from a database table or ResultSet object. For example, suppose you want to export data from a 4-column database table named GBMAXTABLE into a CSV file. The contents of the CSV file, named GBMAXTABLE.csv, might look like the following example:

```
1,0x6263,"bc","bc"  
2,0x636465,"cde","cde"  
3,0x64656667,"defg","defg"  
4,0x6566676869,"efghi","efghi"  
5,0x666768696a6b,"fghijk","fghijk"  
6,0x6768696a6b6c6d,"ghijklm","ghijklm"  
7,0x68696a6b6c6d6e6f,"hijklmno","hijklmno"  
8,0x696a6b6c6d6e6f7071,"ijklmnopq","ijklmnopq"  
9,0x6a6b6c6d6e6f70717273,"jklmnopqrs","jklmnopqrs"  
10,0x6b,"k","k"
```

## Bulk Load Configuration File

Each time data is exported to a CSV file, a bulk load configuration file is created. This file has the same name as the CSV file, but with an .xml extension (for example, GBMAXTABLE.xml).

The bulk load configuration file defines in its metadata the names and data types of the columns in the CSV file based on those in the database table or ResultSet object. It also defines other data properties, such as the source code page for character data types, precision and scale for numeric data types, and nullability for all data types. The format of GBMAXTABLE.xml might look like the following example.

```
<?xml version="1.0" encoding="utf-8"?>  
<table codepage="UTF-8" xsi:noNamespaceSchemaLocation=  
"http://www.datadirect.com/ns/bulk/BulkData.xsd" xmlns:xsi=
```

```

"http://www.w3.org/2001/XMLSchema-instance">
<row>
  <column datatype="DECIMAL" precision="38" scale="0" nullable=
    "false">INTEGERCOL</column>
  <column datatype="VARBINARY" length="10" nullable=
    "true">VARBINCOL</column>
  <column datatype="VARCHAR" length="10" sourcecodepage="Windows-1252"
    externalfilecodepage="Windows-1252" nullable="true">VCHARCOL</column>
  <column datatype="VARCHAR" length="10" sourcecodepage="Windows-1252"
    externalfilecodepage="Windows-1252" nullable="true">UNIVCHARCOL</column>
</row>
</table>

```

If the driver cannot read a bulk load configuration file (for example, because it was inadvertently deleted), the driver reads all data as character data. The character set used by the driver is UTF-8.

## **Bulk Load Configuration File Schema**

The bulk load configuration file is supported by an underlying XML Schema defined at:

<http://www.datadirect.com/ns/bulk/BulkData.xsd>

The bulk load configuration file must conform to the bulk load configuration XML schema. The driver throws an exception if either of the following circumstances occur:

- If the driver performs a data export and the CSV file cannot be created
- If the driver performs a bulk load operation and the driver detects that the CSV file does not comply with the XML Schema described in the bulk load configuration file

## ***Verifying the Bulk Load Configuration File***

Before performing a bulk load operation, your application can verify the metadata in the bulk load configuration file against the structure of the target database table using the `validateTableFromFile()` method. This ensures that the data in the CSV file is compatible with the target database table structure.

Because the verification does not check the actual data in the CSV file, it is possible that the load can fail even if the verification succeeds. For example, suppose your bulk load configuration file has an Integer column that matches an Integer column in the target database. If you modified the data for the Integer column in the CSV file to contain non-digit characters, the bulk load operation would fail even though a verification using `validateTableFromFile()` would succeed.

Not all error messages or warnings generated by the verification process mean that the load will fail. Some messages notify you about possible incompatibilities between the source and target tables. For example, if the CSV file has a column defined as an integer and the column in the target table is defined as smallint, the load may still succeed if the values in the source column are small enough to fit in a smallint column.

## **Character Set Conversions**

When you export data from a database to a CSV file, the CSV file uses the same code page as the database from which the data was exported. If the CSV file and the target database use different code pages, performance for bulk load operations can suffer because the driver must perform a character set conversion.

To avoid character set conversions, your application can specify which code page to use for the CSV file when exporting data. Specify any code page listed in [Table K-1](#) using the `setCodePage()`

method. For example, if the source database table uses a SHIFT-JIS code page and the target database uses a EUC-JP code page, specify `setCodePage("EUC_JP")` to ensure that the CSV file will use the same code page as the target database. If the code page you need to use is not listed, contact Technical Support to request support for that code page.

---

**Table K-1. Supported Code Pages**

---

US_ASCII	IBM273	IBM01140
ISO_8859_1	IBM277	IBM01141
ISO_8859_2	IBM278	IBM01142
ISO_8859_3	IBM280	IBM01143
ISO_8859_4	IBM284	IBM01144
ISO_8859_5	IBM285	IBM01145
ISO_8859_6	IBM290	IBM01146
ISO_8859_7	IBM297	IBM01147
ISO_8859_8	IBM420	IBM01148
ISO_8859_9	IBM424	IBM01149
JIS_Encoding	IBM500	WINDOWS-1250
Shift_JIS	IBM851	WINDOWS-1251
EUC_JP	IBM855	WINDOWS-1252
KS_C_5601	IBM857	WINDOWS-1253
ISO_2022_KR	IBM860	WINDOWS-1254
EUC_KR	IBM861	WINDOWS-1255
ISO_2022_JP	IBM863	WINDOWS-1256
GB2312	IBM864	WINDOWS-1257
ISO_8859_13	IBM865	WINDOWS-1258
ISO_8859_15	IBM869	WINDOWS-854
GBK	IBM870	IBM-939
IBM850	IBM871	IBM-943_P14A-2000
IBM852	IBM1026	IBM-4396
IBM437	KOI8_R	IBM-5026
IBM862	HZ_GB_2312	IBM-5035
Big5	IBM866	UTF-8
MACINTOSH	IBM775	UTF-16LE
IBM037	IBM00858	UTF-16BE

---

## External Overflow Files

When you export data into a CSV file, you can choose to create one large file or multiple smaller files. For example, if you are exporting BLOB data that is a total of several GB, you may want to break the data into multiple smaller files of 100 MB each.

If the values set by the `setCharacterThreshold()` or `setBinaryThreshold()` methods are exceeded, separate files are generated to store character or binary data, respectively. Overflow files are located in the same directory as the CSV file.

The format for overflow file names is:

`CSV_file_name.xxxxxx.lob`

where:

`CSV_file_name` is the name of the CSV file.

`xxxxxx` is a 6-digit number that increments an overflow file.

For example, if multiple overflow files are created for a CSV file named CSV1, the file names would look like this:

`CSV1.000001.lob`

`CSV1.000002.lob`

`CSV1.000003.lob`

...

If the overflow file contains character data, the code page used by the file is the code page specified in the bulk load configuration file for the CSV file.

## Discard File

If the driver was unable to load rows into the database for a bulk load operation from a CSV file, it can record all the rows that were unable to be loaded into the database in the discard file. The contents of the discard file is in the same format as that of the CSV file. After fixing reported issues in the discard file, the bulk load can be reissued, using the discard file as the CSV file.

A discard file is created by specifying a file name and location for the discard file using the `setDiscardFile()` method.

### Example A: Creating a Discard File on Windows

To create a discard file named `discard.csv` located in the `C:\temp` directory, specify:

```
bulkLoad.setLogFile(C:\\temp\\discard.csv)
```

NOTE: 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:  
`C:\\temp\\discard.csv`.

### Example B: Creating a Discard File on UNIX/Linux

To create a discard file named `discard.csv` located in the `/tmp` directory, specify:

```
bulkLoad.setLogFile(/tmp/discard.csv)
```



# L Troubleshooting

This appendix provides information that can help you troubleshoot problems when they occur.

---

## Troubleshooting Your Application

To help you troubleshoot any problems that occur with your application, you can use DataDirect Spy to log detailed information about calls issued by a DataDirect Connect for JDBC driver on behalf of your application. When you enable DataDirect Spy for a connection, you can customize DataDirect Spy logging by setting one or multiple options. See [Appendix H “Tracking JDBC Calls with DataDirect Spy™” on page 783](#) for information about using DataDirect Spy and instructions on enabling and customizing logging.

### Turning On and Off DataDirect Spy<sup>TM</sup> Logging

Once DataDirect Spy logging is enabled for a connection, you can turn on and off the logging at runtime using the `setEnableLogging()` method in the `com.ddtek.jdbc.extensions.ExtLogControl` interface. When DataDirect Spy logging is enabled, all `Connection` objects returned to an application provide an implementation of the `ExtLogControl` interface.

For example, the following code turns off logging using `setEnableLogging(false)`:

```
import com.ddtek.jdbc.extensions.*  
  
// Get Database Connection  
Connection con = DriverManager.getConnection  
("jdbc:datadirect:sqlserver://server1:1433;User=TEST;Password=secret;  
SpyAttributes=(log=(file)/tmp/spy.log");  
  
((ExtLogControl) con).setEnableLogging(false);  
...
```

The `setEnableLogging()` method only turns on and off logging if DataDirect Spy logging has already been enabled for a connection; it does not set or change DataDirect Spy attributes. See ["Enabling DataDirect Spy™" on page 784](#) for information about enabling and customizing DataDirect Spy logging.

## DataDirect Spy™ Log Example

This section provides information to help you understand the content of your own DataDirect Spy logs. For example, suppose your application executes the following code and performs some operations:

```
Class.forName("com.ddtek.jdbc.sqlserver.SQLServerDriver");  
DriverManager.getConnection("jdbc:datadirect:sqlserver:// nc-myserver\  
\sqlserver2005;useServerSideUpdatableCursors=true;resultSetMetaDataOptions=1;  
sendStringParametersAsUnicode=true;alwaysReportTriggerResults=false;  
spyAttributes=(log=(file)c:\\temp\\\\spy.log)","test04", "test04");
```

The log file generated by DataDirect Spy would look similar to the following example. Notes provide explanations for the referenced text.

```
spy>> Connection[1].getMetaData()
spy>> OK (DatabaseMetaData[1])

spy>> DatabaseMetaData[1].getURL()
spy>> OK (jdbc:datadirect:sqlserver://nc-
myserver\sqlserver2005:1433;CONNECTIONRETRYCOUNT=5;RECEIVESTRINGPARAMETERTYPE
=nvarchar;ALTERNATESERVERS=;DATABASENAME=;PACKETSIZE=16;INITIALIZATIONSTRING=
;ENABLECANCELTIMEOUT=false;BATCHPERFORMANCEWORKAROUND=false;AUTHENTICATIONMET
HOD=auto;SENDSTRINGPARAMETERSASUNICODE=true;LOGINTIMEOUT=0;WSID=;SPYATTRIBUTE
S=(log=(file)c:\temp\spy.log);RESULTSETMETADATAOPTIONS=1;ALWAYSREPORTTRIGGERR
ESULTS=false;TRANSACTIONMODE=implicit;USESERVERSIDEUPDATABLECURSORS=true;SNAP
SHOTSERIALIZABLE=false;JAVADOUBLETOSTRING=false;SELECTMETHOD=direct;LOADLIBRA
RYPATH=;CONNECTIONRETRYDELAY=1;INSENSITIVERESULTSETBUFFERSIZE=2048;MAXPOOLEDS
TATEMENTS=0;DESCRIBEPARAMETERS=noDescribe;CODEPAGE OVERRIDE=;NETADDRESS=000000
000000;PROGRAMNAME=;LOADBALANCING=false;HOSTPROCESS=0) 1

spy>> DatabaseMetaData[1].getDriverName()
spy>> OK (SQLServer)

spy>> DatabaseMetaData[1].getDriverVersion()
spy>> OK (3.60.0 (000000.000000.000000))

spy>> DatabaseMetaData[1].getDatabaseProductName()
spy>> OK (Microsoft SQL Server)

spy>> DatabaseMetaData[1].getDatabaseProductVersion()
spy>> OK (Microsoft SQL Server Yukon - 9.00.1399)

spy>> Connection Options : 2
spy>> CONNECTIONRETRYCOUNT=5
spy>> RECEIVESTRINGPARAMETERTYPE=nvarchar
spy>> ALTERNATESERVERS=
```

---

1.The combination of the URL specified by the application and the default values of all connection properties not specified.

2.The combination of the connection properties specified by the application and the default values of all connection properties not specified.

```
spy>> DATABASENAME=
spy>> PACKETSIZE=16
spy>> INITIALIZATIONSTRING=
spy>> ENABLECANCELTIMEOUT=false
spy>> BATCHPERFORMANCEWORKAROUND=false
spy>> AUTHENTICATIONMETHOD=auto
spy>> SENDSTRINGPARAMETERSASUNICODE=true
spy>> LOGINTIMEOUT=0
spy>> WSID=
spy>> SPYATTRIBUTES=(log=(file)c:\temp\spy.log)
spy>> RESULTSETMETADATAOPTIONS=1
spy>> ALWAYSREPORTTRIGGERRESULTS=false
spy>> TRANSACTIONMODE=implicit
spy>> USESERVERSIDEUPDATABLECURSORS=true
spy>> SNAPSHOTSERIALIZABLE=false
spy>> JAVADOUBLETOSTRING=false
spy>> SELECTMETHOD=direct
spy>> LOADLIBRARYPATH=
spy>> CONNECTIONRETRYDELAY=1
spy>> INSENSITIVERESULTSETBUFFERSIZE=2048
spy>> MAXPOOLEDSTATEMENTS=0
spy>> DESCRIBEPARAMETERS=noDescribe
spy>> CODEPAGE OVERRIDE=
spy>> NETADDRESS=000000000000
spy>> PROGRAMNAME=
spy>> LOADBALANCING=false
spy>> HOSTPROCESS=0

spy>> Driver Name = SQLServer 1
spy>> Driver Version = 3.60.0 (000000.000000.000000) 2
spy>> Database Name = Microsoft SQL Server 3
spy>> Database Version = Microsoft SQL Server Yukon - 9.00.1399 4
```

---

1.The name of the driver.

2.The version of the driver.

3.The name of the database server to which the driver connects.

4.The version of the database to which the driver connects.

```
spy>> Connection[1].getWarnings()
spy>> OK1

spy>> Connection[1].createStatement
spy>> OK (Statement[1])

spy>> Statement[1].executeQuery(String sql)
spy>> sql = select empno,ename,job from emp where empno=7369
spy>> OK (ResultSet[1])2

spy>> ResultSet[1].getMetaData()
spy>> OK (ResultSetMetaData[1])3

spy>> ResultSetMetaData[1].getColumnCount()
spy>> OK (3)4

spy>> ResultSetMetaData[1].getColumnName(int column)
spy>> column = 1
spy>> OK (EMPNO)5

spy>> ResultSetMetaData[1].getColumnName(int column)
spy>> column = 2
spy>> OK (ENAME)6

spy>> ResultSetMetaData[1].getColumnName(int column)
spy>> column = 3
spy>> OK (JOB)7

spy>> ResultSet[1].next()
spy>> OK (true)8
```

- 
- 1.The application checks to see if there are any warnings. In this example, no warnings are present.
  - 2.The statement select empno,ename,job from emp where empno=7369 is created.
  - 3.Some metadata is requested.
  - 4.Some metadata is requested.
  - 5.Some metadata is requested.
  - 6.Some metadata is requested.
  - 7.Some metadata is requested.
  - 8.The first row is retrieved and the application retrieves the result values.

## 850 Appendix L Troubleshooting

```
spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 1
spy>> OK (7369) 1

spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 2
spy>> OK (SMITH) 2

spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 3
spy>> OK (CLERK) 3

spy>> ResultSet[1].next()
spy>> OK (false) 4

spy>> ResultSet[1].close()
spy>> OK 5

spy>> Connection[1].close()
spy>> OK 6
```

- 
- 1.The first row is retrieved and the application retrieves the result values.
  - 2.The first row is retrieved and the application retrieves the result values.
  - 3.The first row is retrieved and the application retrieves the result values.
  - 4.The application attempts to retrieve the next row, but only one row was returned for this query.
  - 5.After the application has completed retrieving result values, the result set is closed.
  - 6.The application finishes and disconnects.

---

# Troubleshooting Connection Pooling

Connection pooling allows connections to be reused rather than created each time a connection is requested. If your application is using connection pooling through the DataDirect Connection Pool Manager, you can generate a trace file that shows all the actions taken by the Pool Manager. See [Appendix I “Connection Pool Manager” on page 789](#) for information about using the Pool Manager.

## Enabling Pool Manager Tracing

You can enable Pool Manager logging by calling `setTracing(true)` on the `PooledConnectionDataSource` connection. To disable tracing, call `setTracing(false)` on the connection.

By default, the DataDirect Connection Pool Manager logs its pool activities to the standard output `System.out`. You can change where the Pool Manager trace information is written by calling the `setLogWriter()` method on the `PooledConnectionDataSource` connection.

## Pool Manager Trace File Example

The following example shows a DataDirect Connection Pool Manager trace file. Notes provide explanations for the referenced text to help you understand the content of your own Pool Manager trace files.

```
jdbc/SQLServerNCMarkBPool: *** ConnectionPool Created
( jdbc/SQLServerNCMarkBPool,
  com.ddtek.jdbcx.sqlserver.SQLServerDataSource@1835282, 5, 5, 10, scott) 1
jdbc/SQLServerNCMarkBPool: Number pooled connections = 0.
jdbc/SQLServerNCMarkBPool: Number free connections = 0.

jdbc/SQLServerNCMarkBPool: Enforced minimum! 2
NrFreeConnections was: 0
jdbc/SQLServerNCMarkBPool: Number pooled connections = 5.
jdbc/SQLServerNCMarkBPool: Number free connections = 5.

jdbc/SQLServerNCMarkBPool: Reused free connection. 3
jdbc/SQLServerNCMarkBPool: Number pooled connections = 5.
jdbc/SQLServerNCMarkBPool: Number free connections = 4.
```

---

- 1.The Pool Manager creates a connection pool. In this example, the characteristics of the connection pool are shown using the following format:

*(JNDI\_name,DataSource\_class,initial\_pool\_size,min\_pool\_size,max\_pool\_size,user)*

where:

*JNDI\_name* is the JNDI name used to look up the connection pool (for example, `jdbc/SQLServerNCMarkBPool`).

*DataSource\_class* is the `DataSource` class associated with the connection pool (for example `com.ddtek.jdbcx.sqlserver.SQLServerDataSource`).

*initial\_pool\_size* is the number of physical connections created when the connection pool is initialized (for example, 5).

*min\_pool\_size* is the minimum number of physical connections be kept open in the connection pool (for example, 5).

*max\_pool\_size* is the maximum number of physical connections allowed within a single pool at any one time. When this number is reached, additional connections that would normally be placed in a connection pool are closed (for example, 10).

*user* is the name of the user establishing the connection (for example, `scott`).

- 2.The Pool Manager checks the pool size. Because the minimum pool size is five connections, the Pool Manager creates new connections to satisfy the minimum pool size.

- 3.The driver requests a connection from the connection pool. The driver retrieves an available connection.

```
jdbc/SQLServerNCMarkBPool: Reused free connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 5.  
jdbc/SQLServerNCMarkBPool: Number free connections = 3.  
  
jdbc/SQLServerNCMarkBPool: Reused free connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 5.  
jdbc/SQLServerNCMarkBPool: Number free connections = 2.  
  
jdbc/SQLServerNCMarkBPool: Reused free connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 5.  
jdbc/SQLServerNCMarkBPool: Number free connections = 1.  
  
jdbc/SQLServerNCMarkBPool: Reused free connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 5.  
jdbc/SQLServerNCMarkBPool: Number free connections = 0.  
  
jdbc/SQLServerNCMarkBPool: Created new connection.1  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 6.  
jdbc/SQLServerNCMarkBPool: Number free connections = 0.  
  
jdbc/SQLServerNCMarkBPool: Created new connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 7.  
jdbc/SQLServerNCMarkBPool: Number free connections = 0.  
  
jdbc/SQLServerNCMarkBPool: Created new connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 8.  
jdbc/SQLServerNCMarkBPool: Number free connections = 0.  
  
jdbc/SQLServerNCMarkBPool: Created new connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 9.  
jdbc/SQLServerNCMarkBPool: Number free connections = 0.  
  
jdbc/SQLServerNCMarkBPool: Created new connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 10.  
jdbc/SQLServerNCMarkBPool: Number free connections = 0.
```

---

1.The driver requests a connection from the connection pool. Because a connection is unavailable, the Pool Manager creates a new connection for the request.

```
jdbc/SQLServerNCMarkBPool: Created new connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 0.  
  
jdbc/SQLServerNCMarkBPool: Connection was closed and added to the cache.1  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 1.  
  
jdbc/SQLServerNCMarkBPool: Connection was closed and added to the cache.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 2.  
  
jdbc/SQLServerNCMarkBPool: Connection was closed and added to the cache.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 3.  
  
jdbc/SQLServerNCMarkBPool: Connection was closed and added to the cache.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 4.  
  
jdbc/SQLServerNCMarkBPool: Connection was closed and added to the cache.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 5.  
  
jdbc/SQLServerNCMarkBPool: Connection was closed and added to the cache.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 6.  
  
jdbc/SQLServerNCMarkBPool: Connection was closed and added to the cache.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 7.  
  
jdbc/SQLServerNCMarkBPool: Connection was closed and added to the cache.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 8.
```

---

1.A connection is closed by the application and returned to the connection pool.

```
jdbc/SQLServerNCMarkBPool: Connection was closed and added to the cache.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 9.
```

```
jdbc/SQLServerNCMarkBPool: Connection was closed and added to the cache.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 10.
```

```
jdbc/SQLServerNCMarkBPool: Connection was closed and added to the cache.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 11.
```

```
jdbc/SQLServerNCMarkBPool: Enforced minimum! 1  
NrFreeConnections was: 11  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 11.  
jdbc/SQLServerNCMarkBPool: Number free connections = 11.
```

```
jdbc/SQLServerNCMarkBPool: Enforced maximum! 2  
NrFreeConnections was: 11  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 10.  
jdbc/SQLServerNCMarkBPool: Number free connections = 10.
```

```
jdbc/SQLServerNCMarkBPool: Enforced minimum!  
NrFreeConnections was: 10  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 10.  
jdbc/SQLServerNCMarkBPool: Number free connections = 10.
```

```
jdbc/SQLServerNCMarkBPool: Enforced maximum!  
NrFreeConnections was: 10  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 10.  
jdbc/SQLServerNCMarkBPool: Number free connections = 10.
```

```
jdbc/SQLServerNCMarkBPool: Enforced minimum!  
NrFreeConnections was: 10
```

---

1.The Pool Manager checks the pool size. Because the number of connections in the connection pool is greater than the minimum pool size, five connections, no action is taken by the Pool Manager.

2.The Pool Manager checks the pool size. Because the number of connections in the connection pool is greater than the maximum pool size, 10 connections, a connection is closed and discarded from the pool.

```
jdbc/SQLServerNCMarkBPool: Number pooled connections = 10.  
jdbc/SQLServerNCMarkBPool: Number free connections = 10.  
  
jdbc/SQLServerNCMarkBPool: Enforced maximum!  
NrFreeConnections was: 10  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 10.  
jdbc/SQLServerNCMarkBPool: Number free connections = 10.  
  
jdbc/SQLServerNCMarkBPool: Dumped free connection.1  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 9.  
jdbc/SQLServerNCMarkBPool: Number free connections = 9.  
  
jdbc/SQLServerNCMarkBPool: Dumped free connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 8.  
jdbc/SQLServerNCMarkBPool: Number free connections = 8.  
  
jdbc/SQLServerNCMarkBPool: Dumped free connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 7.  
jdbc/SQLServerNCMarkBPool: Number free connections = 7.  
  
jdbc/SQLServerNCMarkBPool: Dumped free connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 6.  
jdbc/SQLServerNCMarkBPool: Number free connections = 6.  
  
jdbc/SQLServerNCMarkBPool: Dumped free connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 5.  
jdbc/SQLServerNCMarkBPool: Number free connections = 5.  
  
jdbc/SQLServerNCMarkBPool: Dumped free connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 4.  
jdbc/SQLServerNCMarkBPool: Number free connections = 4.  
  
jdbc/SQLServerNCMarkBPool: Dumped free connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 3.  
jdbc/SQLServerNCMarkBPool: Number free connections = 3.  
  
jdbc/SQLServerNCMarkBPool: Dumped free connection.  
jdbc/SQLServerNCMarkBPool: Number pooled connections = 2.
```

---

1.The Pool Manager detects that a connection was idle in the connection pool longer than the maximum idle timeout. The idle connection is closed and discarded from the pool.

```

jdbc/SQLServerNCMarkBPool: Number free connections = 2.

jdbc/SQLServerNCMarkBPool: Dumped free connection.
jdbc/SQLServerNCMarkBPool: Number pooled connections = 1.
jdbc/SQLServerNCMarkBPool: Number free connections = 1.

jdbc/SQLServerNCMarkBPool: Dumped free connection.
jdbc/SQLServerNCMarkBPool: Number pooled connections = 0.
jdbc/SQLServerNCMarkBPool: Number free connections = 0.

jdbc/SQLServerNCMarkBPool: Enforced minimum! 1
NrFreeConnections was: 0
jdbc/SQLServerNCMarkBPool: Number pooled connections = 5.
jdbc/SQLServerNCMarkBPool: Number free connections = 5.

jdbc/SQLServerNCMarkBPool: Enforced maximum!
NrFreeConnections was: 5
jdbc/SQLServerNCMarkBPool: Number pooled connections = 5.
jdbc/SQLServerNCMarkBPool: Number free connections = 5.

jdbc/SQLServerNCMarkBPool: Closing a pool of the group
    jdbc/SQLServerNCMarkBPool 2
jdbc/SQLServerNCMarkBPool: Number pooled connections = 5.
jdbc/SQLServerNCMarkBPool: Number free connections = 5.

jdbc/SQLServerNCMarkBPool: Pool closed 3
jdbc/SQLServerNCMarkBPool: Number pooled connections = 0.
jdbc/SQLServerNCMarkBPool: Number free connections = 0.

```

- 1.The Pool Manager detects that the number of connections dropped below the limit set by the minimum pool size, five connections. The Pool Manager creates new connections to satisfy the minimum pool size.
- 2.The Pool Manager closes one of the connection pools in the pool group. A pool group is a collection of pools created from the same PooledConnectionDataSource call. Different pools are created when different user IDs are used to retrieve connections from the pool. A pool group is created for each user ID that requests a connection. In our example, because only one user ID was used, only one pool group is closed.
- 3.The Pool Manager closed all the pools in the pool group. The connection pool is closed.

---

## Troubleshooting Statement Pooling

Similar to connection pooling, statement pooling provides performance gains for applications that execute the same SQL statements multiple times in the life of the application. The DataDirect Statement Pool Monitor provides the following functionality to help you troubleshoot problems that may occur with statement pooling:

- You can generate a statement pool export file that shows you all statements in the statement pool. Each statement pool entry in the file includes information about statement characteristics such as the SQL text used to generate the statement, statement type, result set type, and result set concurrency type.
- You can use the following methods of the ExtStatementPoolMonitorMBean interface to return useful information to determine if your workload is using the statement pool effectively:
  - The getHitCount() method returns the hit count for the statement pool. The hit count should be high for good performance.
  - The getMissCount() method returns the miss count for the statement pool. The miss count should be low for good performance.

See [Appendix J “Statement Pool Monitor” on page 811](#) for more information about using the Statement Pool Monitor.

## Generating a Statement Pool Export File

You can generate an export file by calling the `exportStatements()` method of the `ExtStatementPoolMonitorMBean` interface. For example, the following code exports the contents of the statement pool associated with the connection to a file named `stmt_export`:

```
ExtStatementPoolMonitor monitor =
((ExtConnection) con).getStatementPoolMonitor();
exportStatements(stmt_export.txt)
```

## Statement Pool Export File Example

The following example shows a sample export file. Notes provide explanations for the referenced text to help you understand the content of your own statement pool export files.

```
DDTEK_STMT_POOL1
VERSION 12

[STMT_ENTRY}3
SQL_TEXT=[  
INSERT INTO emp(id, name) VALUES(?,?)  
]  
STATEMENT_TYPE=Prepared Statement  
RESULTSET_TYPE=Forward Only  
RESULTSET_CONCURRENCY=Read Only  
AUTOGENERATEDKEYSREQUESTED=false  
REQUESTEDKEYCOLUMNS=
```

---

1.A string that identifies the file as a statement pool export file.

2.The version of the export file.

3.The first statement pool entry. Each statement pool entry lists the SQL text, statement type, result set type, result set concurrency type, and generated keys information.

```
[STMT_ENTRY] 1
SQL_TEXT=[  
INSERT INTO emp(id, name) VALUES(99,?)  
]  
STATEMENT_TYPE=Prepared Statement  
RESULTSET_TYPE=Forward Only  
RESULTSET_CONCURRENCY=Read Only  
AUTOGENERATEDKEYSREQUESTED=false  
REQUESTEDKEYCOLUMNNS=id,name
```

---

1.The next statement pool entry.

# Glossary

<b>authentication</b>	The process of identifying a user, typically based on a user ID and password. Authentication ensures that the user is who they claim to be. See also <i>client authentication</i> , <i>NTLM authentication</i> , <i>OS authentication</i> , and <i>user ID/password authentication</i> .
<b>bulk load</b>	The process of sending large numbers of rows of data to the database in a continuous stream instead of in numerous smaller database protocol packets. This process also is referred to as bulk copy.
<b>client authentication</b>	Client authentication uses the user ID and password of the user logged onto the system on which the driver is running to authenticate the user to the database. The database server depends on the client to authenticate the user and does not provide additional authentication. See also <i>authentication</i> .
<b>client load balancing</b>	Client load balancing distributes new connections in a computing environment so that no one server is overwhelmed with connection requests.
<b>connection pooling</b>	Connection pooling allows you to reuse connections rather than create a new one every time a driver needs to establish a connection to the database. Connection pooling manages connection sharing across different user requests to maintain performance and reduce the number of new connections that must be created. See also <i>DataDirect Connection Pool Manager</i> .
<b>connection retry</b>	Connection retry defines the number of times the driver attempts to connect to the primary and, if configured, alternate database servers after an initial unsuccessful connection attempt. Connection retry can be an important strategy for system recovery.
<b>connection URL</b>	A connection URL is a string passed by an application to the Driver Manager that contains information required to establish a connection. See also <i>Driver Manager</i> .
<b>DataDirect DB2 Package Manager</b>	A Java graphical tool shipped with DataDirect Connect for JDBC for creating, dropping, and replacing DB2 packages for DB2.

<b>DataDirect Connection Pool Manager</b>	The DataDirect Connection Pool Manager is a component shipped with DataDirect Connect <i>for JDBC</i> that allows applications to use connection pooling.
<b>DataDirect Spy</b>	DataDirect Spy allows you to track and log detailed information about JDBC calls made by DataDirect Connect <i>for JDBC</i> drivers at runtime. This functionality is built into the drivers.
<b>DataDirect Test</b>	DataDirect Test is a menu-driven component shipped with DataDirect Connect <i>for JDBC</i> that helps you debug your applications and learn how to use the drivers. DataDirect Test displays the results of all JDBC function calls in one window, while displaying fully commented, Java JDBC code in an alternate window.
<b>data source</b>	A data source is a DataSource object that provides the connection information needed to connect to a database. The main advantage of using a data source is that it works with the Java Naming Directory Interface (JNDI) naming service, and it is created and managed apart from the applications that use it.
<b>failover</b>	Failover allows an application to connect to an alternate, or backup, database server. DataDirect Connect <i>for JDBC</i> provides different levels of failover: connection failover, extended connection failover, and select failover.
<b>Driver Manager</b>	The main purpose of the Driver Manager is to load drivers for the application. The Driver Manager also processes JDBC initialization calls and maps data sources to a specific driver.
<b>index</b>	A database structure used to improve the performance of database activity. A database table can have one or more indexes associated with it.
<b>isolation level</b>	An isolation level represents a particular locking strategy employed in the database system to improve data consistency. The higher the isolation level number, the more complex the locking strategy behind it. The isolation level provided by the database determines how a transaction handles data consistency.  The American National Standards Institute (ANSI) defines four isolation levels: <ul style="list-style-type: none"><li>■ Read uncommitted (0)</li><li>■ Read committed (1)</li><li>■ Repeatable read (2)</li><li>■ Serializable (3)</li></ul>

<b>J2EE</b>	J2EE (Java 2 Platform, Enterprise Edition) technology and its component-based model simplify enterprise development and deployment. The J2EE platform manages the infrastructure and supports the Web services to enable development of secure, robust and interoperable business applications.
<b>JDBC data source</b>	See <i>data source</i> .
<b>JNDI</b>	The Java Naming and Directory Interface (JNDI) is a standard extension to the Java platform, providing Java technology-enabled applications with a unified interface to multiple naming and directory services in the enterprise. As part of the Java Enterprise API set, JNDI enables seamless connectivity to heterogeneous enterprise naming and directory services. Developers can now build powerful and portable directory-enabled applications using this industry standard.
<b>JTA</b>	JTA (Java Transaction API) specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications.
<b>Kerberos</b>	Kerberos is an OS authentication protocol that provides authentication using secret key cryptography. See also <i>authentication</i> and <i>OS authentication</i> .
<b>load balancing</b>	See <i>client load balancing</i> .
<b>locking level</b>	Locking is a database operation that restricts a user from accessing a table or record. Locking is used in situations where more than one user might try to use the same table at the same time. By locking the table or record, the system ensures that only one user at a time can affect the data.
<b>NTLM authentication</b>	NTLM (NT LAN Manager) is an OS authentication protocol that provides security for connections between Windows NT clients and servers. See also <i>authentication</i> and <i>OS authentication</i> .
<b>OS authentication</b>	OS authentication can take advantage of the user name and password maintained by the operating system to authenticate users to the database or use another set of user credentials specified by the application. By allowing the database to share the user name and password used for the operating system, users with a valid operating system account can log into the database without supplying a user name and password. See also <i>authentication</i> , <i>Kerberos authentication</i> , and <i>NTLM authentication</i> .

<b>Performance Tuning Wizard</b>	An applet shipped with DataDirect Connect for JDBC that leads you step-by-step through a series of questions about your application. Based on your answers, the Wizard provides the optimal settings for DataDirect Connect for JDBC connection properties that affect performance.
<b>reauthentication</b>	The process of switching the user associated with a connection to another user to help minimize the number of connections required in a connection pool.
<b>resource adapter</b>	A resource adapter is a system-level software driver used by an application server to connect to an Enterprise Information Service (EIS). The resource adapter communicates with the server to provide the underlying transaction, security, and connection pooling mechanisms.
<b>Secure Socket Layer (SSL)</b>	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 SSL client/SSL server authentication. See also <i>SSL client/server authentication</i> .
<b>SSL client/server authentication</b>	SSL works by allowing the client and server to send each other encrypted data that only they can decrypt. SSL negotiates the terms of the encryption in a sequence of events known as the SSL handshake. The handshake involves the following types of authentication: <ul style="list-style-type: none"><li>■ SSL server authentication requires the server to authenticate itself to the client.</li><li>■ SSL client authentication is optional and requires the client to authenticate itself to the server after the server has authenticated itself to the client.</li></ul> See also <i>Secure Socket Layer (SSL)</i> .
<b>Unicode</b>	A standard for representing characters as integers. Unlike ASCII, which uses 7 bits for each character, Unicode uses 16 bits, which means that it can represent more than 65,000 unique characters. This is necessary for many languages, such as Greek, Chinese, and Japanese.
<b>user ID/password authentication</b>	User ID/password authentication authenticates the user to the database using a database user name and password. See also <i>authentication</i> .

# Index

## A

about  
 authentication 67  
 bulk load 75  
 connection failover 59  
 connection pooling 52  
 DataDirect Connection Pool Manager 37  
 DataDirect DB2 Package Manager 134  
 DataDirect Spy 37  
 DataDirect Statement Pool Monitor 57  
 DataDirect Test 36  
 drivers 36  
 encryption 69  
 extended connection failover 61  
 failover 58  
 load balancing 65  
 reauthentication 55  
 resource adapters 37  
 select failover 62  
 SSL 70  
 statement pooling 57  
 accessing the DataDirect Statement Pool  
 Monitor 812, 815  
**AccountingInfo**  
 DB2 88  
 Informix 179  
 MySQL 219  
 Oracle 268  
 SQL Server 376  
 Sybase 473  
**addresses, IP** 50  
**AddToCreateTable (DB2)** 89  
**AllowImplicitResultSetCloseForXA property (DB2)** 89

**alternate database servers**  
**AlternateServers**  
 DB2 90  
 Informix 180  
 MySQL 219  
 Oracle 268  
 SQL Server 376  
 Sybase 474  
 specifying  
 DB2 90  
 Informix 180  
 MySQL 219  
 Oracle 268  
 SQL Server 376  
 Sybase 474  
**alternate server, specifying**  
 guidelines for 63  
**AlternateID (DB2)** 90  
**AlternateServers**  
 DB2 90  
 Informix 180  
 MySQL 219  
 Oracle 268  
 SQL Server 376  
 Sybase 474  
**AlwaysReportTriggerResults (SQL Server)** 377  
**ANSI\_NULLS option (SQL Server)** 462  
**ansinull option (Sybase)** 527  
**applet**  
 Java 2 Platform, permissions for 77  
 Performance Tuning Wizard 33  
 untrusted 46, 86, 178, 266, 374, 472  
 using drivers with 35, 42  
**application**  
 connecting with drivers 29  
 connection pooling 21, 37, 811  
 data source, calling in 49  
 DataDirect Spy, using to troubleshoot 845

- multi-lingual 82
- optimizing for performance
  - See performance optimization
- resource adapters, permissions for 42
- statement pooling 57
- troubleshooting problems 845
- Unicode support 82
- using
  - drivers with 35
  - resource adapters directly 40
- application server
  - Enterprise Information System (EIS) 37
  - example, setting up resource adapters 40
  - J2EE Connector Architecture 38
  - resource archive (RAR) file 39
  - Service Provider Interface (SPI) 38
  - using
    - drivers with 35, 42
    - resource adapters with 39
- ApplicationName
  - DB2 91
  - Informix 181
  - MySQL 220
  - Oracle 269
  - SQL Server 378
  - Sybase 474
- Array object, methods 536
- attributes
  - DataDirect Spy 787
  - DB2 Workload Manager (WLM) 628
- authentication
  - about 67
  - client
    - DB2 147
    - Oracle 341
  - client (SSL) 73
  - DB2 146
  - Kerberos
    - DB2 147
    - Oracle 340
    - SQL Server 428
    - Sybase 512
- NTLM
  - Oracle 341
  - SQL Server 429
- server (SSL) 71
- support 68
- Sybase 512
- types of 67
- user ID/password
  - DB2 146
  - Oracle 340
  - Sybase 512
- AuthenticationMethod
  - DB2 92, 147
  - Oracle 270, 341
  - SQL Server 379, 429
  - Sybase 475, 513
- Autocommit mode 713
- auto-generated keys
  - DB2 169
  - example, retrieving 710
  - Informix 210
  - MySQL 257
  - Oracle 364
  - performance optimization 710
  - SQL Server 460
  - Sybase 526

## B

- batch inserts and updates
  - DB2 93, 163
  - example
    - batch execution on a prepared statement (DataDirect Test) 751
    - performance gain 706
  - Oracle 271, 356
  - SQL Server 455
  - Sybase 476, 523
  - using instead of prepared statements 705
- batch inserts, using bulk load for 837

BatchPerformanceWorkaround  
 DB2 93  
 Oracle 271  
 Sybase 476  
**BatchUpdateException (SQL Server)** 455  
**binary stream data (DB2)** 124  
**Blob**  
 and binary stream data (DB2) 124  
 object 536  
 retrieving data (DataDirect Test) 776  
 searches (Oracle) 209  
 using to return long data  
   MySQL 252  
   SQL Server 455  
   Sybase 522  
**bulk load**  
 character set conversions 840  
 CSV files, using 838  
 discard file 843  
 exporting data to a CSV file from  
   database table 834  
   ResultSet object 834  
 external overflow files 842  
 loading data  
   from a CSV file 835  
   from a ResultSet object 835  
 logging 836  
 overview 75  
 using for batch inserts 837  
**bulk load configuration file**  
 about 838  
 schema 839  
 verifying 840  
**BulkLoadBatchSize**  
 DB2 94  
 Oracle 272  
 SQL Server 380  
 Sybase 476  
**BulkLoadOptions (SQL Server)** 381

**C**

caching long data  
 SQL Server 400  
 Sybase 493  
**CallableStatement object, methods** 539  
**cancel request timeout**  
 DB2 104  
 Oracle 282  
 SQL Server 388  
 Sybase 482  
**catalog table views (DB2)** 157  
**CatalogIncludesSynonyms**  
 DB2 95  
 Oracle  
   See CatalogOptions (Oracle)  
**CatalogOptions**  
 DB2 95  
 Oracle 273  
**CatalogSchema (DB2)** 96  
**certificate**  
 SSL 71  
 validation  
 DB2 128  
 MySQL 246  
 Oracle 312  
 SQL Server 414  
 Sybase 506  
**Certificate Authority (CA), SSL** 71  
**Character data, converting**  
 DB2 98  
 Informix 182  
 MySQL 221  
 Oracle 275  
 SQL Server 383  
 Sybase 478  
**character set conversions, bulk load** 840  
**CharsetFor65535 (DB2)** 97  
**checking version of DataDirect Connection Pool Manager** 793

classes  
 data source and driver  
   DB2 86  
   Informix 177  
   MySQL 217  
   Oracle 265  
   SQL Server 373  
   Sybase 471  
 DataDirect Connection Pool Manager 803  
 DataDirect Statement Pool Monitor 824  
 J2EE Connector Architecture resource  
   adapter  
   DB2 87  
   Informix 178  
   MySQL 218  
   Oracle 267  
   SQL Server 375  
   Sybase 472  
 CLASSPATH, setting  
   example  
     on UNIX 30  
     on Windows 30  
   variable 29  
 clear text user ID and password  
   (DB2) 146, 147  
 clearing statements in the statement  
   pool 821  
 client authentication  
   about  
     DB2 147  
     Oracle 341  
   configuring  
     DB2 155  
     Oracle 352  
   SSL 73  
 client information  
   about 621  
   DB2 160  
   how databases store 622  
   Informix 204  
   location used for storing 624  
   maximum length of a value,  
     determining 627  
   metadata, returning 627  
   MySQL 251  
   Oracle 355  
   returning 626  
   SQL Server 446  
   storing 623  
   Sybase 521  
 ClientHostName  
   DB2 97  
   Informix 181  
   MySQL 221  
   Oracle 274  
   SQL Server 382  
   Sybase 477  
 ClientUser  
   DB2 98  
   Informix 181  
   MySQL 221  
   Oracle 274  
   SQL Server 383  
   Sybase 477  
 Clob  
   object 549  
   retrieving data (DataDirect Test) 776  
   searches (Oracle) 209  
   using to return long data  
     MySQL 252  
     SQL Server 455  
     Sybase 522  
 Clob data, converting  
   DB2 98  
   MySQL 221  
 closing the connection pool 802  
 code page  
   converting Character data  
     DB2 98  
     Informix 182  
     MySQL 221  
     Oracle 275  
     SQL Server 383  
     stored as bit data (DB2) 97  
     Sybase 478  
   converting Clob data  
     DB2 98  
     Informix 182

MySQL 221  
 SQL Server 383  
 Sybase 478  
 overriding  
   DB2 98  
   Informix 182  
   MySQL 221  
   Oracle 275  
   SQL Server 383  
   Sybase 478  
**CodePageOverride**  
   DB2 98  
   Informix 182  
   MySQL 221  
   Oracle 275  
   SQL Server 383  
   Sybase 478  
**CollectionId (DB2)**  
   See **PackageCollection (DB2)**  
**com.ddtek.jdbc.extensions package** 20, 601  
**Comma-Separated Value (CSV) files**  
   See **CSV files**  
**CommitBehavior (Oracle)** 277  
**committing transactions** 713  
**concurrency type of result set in statement pool** 815  
**configuring**  
   client authentication  
     DB2 155  
     Oracle 352  
   encryption  
     DB2 155  
     MySQL 250  
     Oracle 353  
     SQL Server 441  
     Sybase 520  
   failover  
     DB2 171  
     Informix 211  
     MySQL 258  
     Oracle 365  
     SQL Server 463  
     Sybase 529  
**Kerberos authentication**  
   DB2 149  
   Oracle 343  
   SQL Server 431  
   Sybase 514  
**NTLM authentication**  
   Oracle 350  
   SQL Server 437  
**user ID/password authentication**  
   DB2 149  
   Oracle 343  
   SQL Server 430  
   Sybase 514  
**connect descriptor parameters in tnsnames.ora file (Oracle)** 323  
**connecting**  
   connection URL example  
     for all drivers 47  
     for SQL Server named instances 420  
   data sources 48  
   DataDirect Test  
     using a data source 734  
     using driver/database selection 736  
   Driver Manager, JDBC 44  
   example  
     DataDirect Test 734  
     Driver Manager and DataDirect Spy 784  
     Driver Manager and drivers 44  
     JDBC data source 49  
   quick start guide to 29  
   SQL Server named instances 420  
   tnsnames.ora file (Oracle) 266  
   using  
     connection pool 799  
     URLs 47  
**connection failover**  
   about 59  
   connection retry and 60  
**connection management** 711  
**Connection object**  
   ExtLogControl interface 845  
   managing connections 711  
   methods 552, 556, 598  
   transaction model 714

- connection pool
  - See connection pooling
- Connection Pool Monitor 806
- connection pooling 851
  - connection pool
    - closing 802
    - configuring 789
    - connecting using 799
    - creating 796
  - PooledConnectionDataSource object,
    - creating 796
  - with reauthentication, example 792
  - without reauthentication, example 791
- Connection Pool Monitor 806
- ConnectionEventListener interface 54
- driver DataSource object for connection pooling, creating 794
- example
  - DataDirect Connection Pool Manager
    - trace file 851
  - pooled data source 794
  - reauthentication 791, 792
  - transactions with connection pooling 53
  - transactions without connection pooling 52
- javax.sql.ConnectionPoolDataSource interface 48
- performance optimization 711
- PooledConnectionDataSource object, creating 796
- using 52
- connection properties
  - DB2 88
  - driver 36
  - Informix 179
  - MySQL 219
  - Oracle 267
  - Performance Tuning Wizard, using 33
  - specifying 43
  - SQL Server 375
  - Sybase 473
- connection retry
  - ConnectionRetryCount
    - DB2 99
    - Informix 182
    - MySQL 222
    - Oracle 278
    - SQL Server 384
    - Sybase 478
  - ConnectionRetryDelay
    - DB2 100
    - Informix 183
    - MySQL 223
    - Oracle 279
    - SQL Server 385
    - Sybase 479
  - DB2 174
  - Informix 214
  - MySQL 261
  - Oracle 369
  - overview 66
  - SQL Server 467
  - Sybase 532
- connection URL
  - DB2 86
  - Informix 177
  - MySQL 218
  - Oracle 266
  - SQL Server 374
  - Sybase 472
- ConnectionPoolDataSource object
  - connecting with 54
  - methods 556
- ConnectionPoolMonitor class, methods 809
- ConnectionRetryCount
  - DB2 99
  - Informix 182
  - MySQL 222
  - Oracle 278
  - SQL Server 384
  - Sybase 478
- ConnectionRetryDelay
  - DB2 100
  - Informix 183
  - MySQL 223

- Oracle 279
- SQL Server 385
- Sybase 479
- connections**
  - client information for
    - returning 75
    - storing 75
  - connection pooling 52
  - connection properties, specifying 43
  - connection URLs 44
  - data sources 48
  - JDBC Driver Manager 44
  - orphaned (SQL Server) 452
  - permissions 77
  - testing 76
- contacting Technical Support 26
- conventions, typographical 25
- converting
  - Character data
    - DB2 98
    - Informix 182
    - MySQL 221
    - Oracle 275
    - SQL Server 383
    - Sybase 478
  - Clob data
    - DB2 98
    - MySQL 221
  - null values
    - DB2 101
    - Informix 184
    - MySQL 224
    - Oracle 280
    - SQL Server 385
    - Sybase 480
  - UTF-16 encoding 82
- ConvertNull**
  - DB2 101
  - Informix 184
  - MySQL 224
  - Oracle 280
  - SQL Server 385
  - Sybase 480
- CREATE statements (DB2)** 89
- CreateDefaultPackage (DB2)** 101
- creating**
  - connection pool 796
  - DB2 package 134
  - driver DataSource object for connection
    - pooling 794
  - PooledConnectionDataSource object 796
- CSV files**
  - example 838
  - using 838
- CURRENT SCHEMA register (DB2)** 90
- CURRENT SQLID register (DB2)** 90
- CurrentFunctionPath (DB2)** 102
- cursors**
  - choosing for performance 707
  - cursor behavior in transactions (DB2) 130
  - DB2 162
  - Informix 205
  - MySQL 252
  - Oracle 356
  - SQL Server 448
  - Sybase 522

## D

- data sources**
  - application, calling in 49
  - connecting with 29
  - connection pooling 52
  - creating
    - data sources 49
    - driver DataSource object for connection
      - pooling 794
    - PooledConnectionDataSource object 796
  - DataSource interface 48
  - example
    - using File System JNDI Provider 49
    - using in JDBC application 49
    - using JNDI Provider for LDAP 49
  - implementing 48
  - registering with JNDI 54

- data types
  - date/time (Oracle 9i and higher) 332
  - DB2 139
  - Informix 202
  - MySQL 248
  - Oracle 330
  - performance, choosing for 702
  - SQL Server 421
  - Sybase 510
- database
  - errors 83
  - metadata, retrieving (DataDirect Test) 745
  - table characteristics, using dummy query to determine 698
- Database (alias for DatabaseName)
  - DB2 102
  - Informix 184, 185
  - MySQL 224
  - Oracle 280, 281
  - SQL Server 386
  - Sybase 481
- DATABASE SQL statement (Informix) 185
- DatabaseMetaData object, methods 557
- DatabaseName
  - DB2 102
  - Informix 185
  - MySQL 224
  - Oracle 281
  - SQL Server 386
  - Sybase 481
- DataDirect Bulk Load
  - See bulk load
- DataDirect Connect for JDBC drivers
  - connecting with and using 35
  - MySQL 217
  - SQL Server 373
  - Sybase 471
- DataDirect Connection Pool Manager
  - about 37
  - checking the version 793
  - trace file 851
    - example 851
    - using 851
- tracing, enabling 793, 851
- using reauthentication with 790
- DataDirect DB2 Package Manager 134, 135
- DataDirect Spy
  - about 37
  - attributes 787
  - enabling
    - using JDBC data sources 785
    - using JDBC Driver Manager 784
  - example
    - JDBC data source connection 786, 787
    - JDBC Driver Manager connection 785
  - logging
    - generating a log 845
    - JDBC calls to System.out 787
    - log example 846
    - turning on and off 845
  - setEnableLogging() method 845
  - SpyAttributes connection property
    - DB2 125
    - Informix 199
    - MySQL 243
    - Oracle 306
    - SQL Server 410
    - Sybase 502
    - using 784
- DataDirect Statement Pool Monitor
  - about 57
  - accessing
    - using DataDirect-specific methods 815
    - using the JMX API 812
  - classes and interfaces 824
  - using 21, 811
- DataDirect Test
  - about 36
  - batches 751
  - configuring 730
  - connecting with 734
  - database metadata, retrieving 745
  - example 729
  - executing
    - prepared statement 741
    - Select statement 739
  - LOB support 776

- ParameterMetaData, returning 755
- result set, scrolling through 748
- rows, deleting 764
- savepoints, establishing 757
- starting 731
- tutorial 729
- updatable result sets 764
- using 729
- DataSource object
  - calling in an application 50
  - connecting with 48
  - connection pooling
    - ConnectionPoolDataSource interface 54
    - creating DataSource object for 794
    - obtaining a connection 789
    - Pool Monitor 806
    - referencing for 796, 804, 806, 807
  - definition 862
  - methods 566
  - retrieving 49
- date format (Informix) 186
- date, time, timestamp escape sequences 718
- date/time data types (Oracle 9i and higher) 332
- DB2
  - alternate database servers, specifying 172
  - authentication 146
  - auto-generated keys 169
  - batch inserts and updates 93, 163
  - binary stream data 124
  - cancel request timeout 104
  - catalog table views 157
  - classes, data source and driver 86
  - clear text user ID and password 146, 147
  - client authentication
    - about 147
    - configuring 155
  - client information 160
  - closing result sets in distributed transactions automatically 89
  - connection properties
    - AccountingInfo 88
    - AddToCreateTable 89
    - AllowImplicitResultSetCloseForXA 89
  - AlternateId 90
  - AlternateServers 90
  - ApplicationName 91
  - AuthenticationMethod 92, 147
  - BatchPerformanceWorkaround 93
  - BulkLoadBatchSize 94
  - CatalogIncludesSynonyms 95
  - CatalogOptions 95
  - CatalogSchema 96
  - CharsetFor65535 97
  - ClientHostName 97
  - ClientUser 98
  - CodePageOverride 98
  - ConnectionRetryCount 99
  - ConnectionRetryDelay 100
  - ConvertNull 101
  - CreateDefaultPackage 101
  - CurrentFunctionPath 102
  - Database (alias for DatabaseName) 102
  - DatabaseName 102
  - DynamicSections 103
  - EnableBulkLoad 103
  - EnableCancelTimeout 104
  - EncryptionMethod 105
  - FailoverGranularity 106
  - FailoverMode 107
  - FailoverPreconnect 108
  - for connection failover 175
  - for DB2 packages 136
  - GrantExecute 109
  - HostNameInCertificate 110
  - ImportStatementPool 111
  - InitializationString 111
  - InInsensitiveResultSetBufferSize 112
  - JavaDoubleToString 113
  - JDBCBehavior 114
  - KeyPassword 114
  - KeyStore 115
  - KeyStorePassword 115
  - LoadBalancing 116
  - LocationName 117
  - LoginTimeout 118
  - MaxPooledStatements 118
  - PackageCollection 99, 119

- PackageOwner 120
- Password 120
- PortNumber 120
- ProgramID 121
- QueryTimeout 122
- ReplacePackage 122
- ResultSetMetaDataOptions 123
- SendStreamAsBlob 124
- ServerName 125
- SpyAttributes 125
- StripNewlines 126
- TrustStore 126
- TrustStorePassword 127
- UseCurrentSchema 127
- User 128
- ValidateServerCertificate 128
- WithHoldCursors 129
- XMLEDescribeType 130
- connection retry 99, 174
- connection URL 86
- CREATE statements 89
- CURRENT SCHEMA register 90
- CURRENT SQLID register 90
- cursor behavior 130
- data types 139
- DataDirect DB2 Package Manager 135
- DB2 packages
  - creating by copying packages (z/OS and iSeries) 139
  - creating using connection properties 136
  - creating using DB2 Package Manager 135
  - creating using list files 137
- default catalog schema 157
- delegation of credentials 152
- dynamic sections 103, 134, 137, 139
- encryption
  - configuring 155
  - DES 155
  - user ID and password 146
- encryption, user ID and password 147
- failover 171
- getTypeInfo() method 631
- High Availability Disaster Recovery (HADR) 64, 91, 171
- isolation levels 162
- J2EE Connector Architecture resource adapter class 87
- JSR 114, Rowsets 169
- JTA support 163
- Kerberos authentication
  - about 147
  - configuring 149
  - javax.security.auth.Subject 152
  - Kerberos configuration file 150
  - kinit command (UNIX and Linux) 154
  - MIT Kerberos 147
  - permissions 80
  - product requirements 149
  - service principal name 80
  - Ticket Granting Ticket (TGT) 154
  - user credentials, specifying 152
  - Windows Active Directory Kerberos 147
- load balancing 116
- LOB support 163
- newline characters 126
- non-default schema, using 157
- null values, handling for data
  - conversions 101
- parameter metadata 164
- performance considerations 131
- prepared statement pool size, setting 118
- primary server, specifying 172
- reauthentication 158
- restrictions on procedure calls 727
- ResultSet metadata 123, 167
- Rowsets 169
- schemas for catalog methods 96, 157
- scrollable cursors, using 162
- SSL encryption, configuring 156
- synonyms in result sets 95
- table name information in ResultSet
  - metadata, returning 123
- timeout, connection 118
- timeout, query 122
- unqualified function names 102
- URL 86

- user ID/password authentication
  - about 146
  - configuring 149
- Workload Manager (WLM) 75, 160, 621
- Workload Manager (WLM) service
  - class 160
- workloads
  - about 160
  - performance 161
- XML
  - inserting 142
  - returning 142
  - updating 142
- DB2 packages
  - connection properties
    - for creating 136
  - GrantExecute 109
  - PackageCollection 99, 119
  - PackageOwner 120
  - ReplacePackage 136
- creating
  - automatically 134
  - copying packages (zOS and iSeries) 139
  - using connection properties 136
  - using DataDirect DB2 Package Manager 134
    - Manager 134
    - using list files 137
- DataDirect DB2 Package Manager 135
- dynamic sections 103
- example
  - DB2 for iSeries 137
  - DB2 for Linux/UNIX/Windows 137
  - DB2 for z/OS 137
- DB2 Workload Manager (WLM)
  - attributes 628
- DBDate (Informix) 186
- DBBulkLoad interface, methods 603
- DBBulkLoad object, using 830
- default catalog schema (DB2) 157
- delegation of credentials for Kerberos
  - DB2 152
  - Oracle 346
  - SQL Server 434
  - Sybase 517
- DELIMIDENT environment variable, setting (Informix) 200
- deployment descriptor, resource adapters 39, 42
- DescribeParameters (SQL Server) 386
- designing JDBC applications
  - See performance optimization
- discard file for bulk load 843
- distributed transactions
  - See JTA support
- DLL, NTLM authentication
  - Oracle 352
  - SQL Server 439
- DML with results (SQL Server) 442
- documentation, about 22
- double quote ("") characters in SQL statements (Informix) 200
- driver
  - DataSource object for connection pooling 794
  - errors 83
- driver classes
  - DB2 86
  - Informix 177
  - MySQL 217
  - Oracle 265
  - SQL Server 373
  - Sybase 471
- Driver object, methods 567
- dummy query, using to determine table characteristics 698
- DynamicSections (DB2) 103

## E

- EnableBulkLoad
  - DB2 103
  - Oracle 281
  - SQL Server 388
  - Sybase 481

EnableCancelTimeout  
 DB2 104  
 Oracle 282  
 SQL Server 388  
 Sybase 482  
**EnableServerResultCache (Oracle 11g)** 283  
 enabling statement pooling 811  
**encryption**  
 about 69  
 DB2 155  
 MySQL 250  
 Oracle 352  
 performance optimization 69  
 SQL Server 439  
 support 70  
 Sybase 520  
 types of 69  
**EncryptionMethod**  
 DB2 105  
 MySQL 225  
 Oracle 283  
 SQL Server 389  
 Sybase 483  
**Enterprise Information System (EIS)** 37  
**error handling** 82  
**ErrorBehavior (Sybase)** 484  
**escape sequences**  
 date, time, and timestamp 718  
 LIKE escape character for wildcards 727  
 outer join 725  
 procedure call 727  
 establishing savepoints (DataDirect Test) 757  
**example**  
 auto-generated keys, retrieving 710  
 batches, using 706  
 connection pool with  
   reauthentication 792  
 connection pool without  
   reauthentication 791  
 connection pooling data source 54  
**connection URL**  
 for SQL Server named instances 420  
 specifying 32  
**CSV file** 838  
**data source**  
 File System JNDI Provider, using 49  
 for connection pooling 794  
 in JDBC application 49  
 JNDI Provider for LDAP, using 49  
**DataDirect Spy**  
 JDBC data source connection 786, 787  
 JDBC Driver Manager connection 785  
 log 846  
**DataDirect Test** 729  
**date, time, and timestamp escape**  
 sequence 718  
**DB2 packages**  
 DB2 for iSeries 137  
 DB2 for Linux/UNIX/Windows 137  
 DB2 for z/OS 137  
**driver connection URL** 47  
**Driver Manager**, using to  
 connect 44, 784, 785  
**executing a prepared statement multiple**  
 times 706  
**IPv4/IPv6** 51  
**logging**, enabling for bulk load 836  
**obtaining a javax.security.auth.Subject for**  
 authentication  
 DB2 152  
 Oracle 347  
 SQL Server 434  
 Sybase 517  
**outer join escape sequence** 725  
**permissions**, granting access to temporary  
 files 78  
**reauthentication** 56  
**REF CURSOR (Oracle)** 338, 339  
**registering driver using Class.forName** 30  
**resource adapters**  
 application using directly 40  
 using with a J2EE application server 40  
**returning REF CURSORs (Oracle)** 338  
**scalar functions** 718  
**setting CLASSPATH**  
 on UNIX 30  
 on Windows 30  
**statement pool export file** 859

trace file, DataDirect Connection Pool Manager 851  
 transactions  
   with connection pooling 53  
   without connection pooling 52  
 executing  
   prepared statement (DataDirect Test) 741  
   Select statement (DataDirect Test) 739  
 export file for statement pool  
   example 859  
   generating 823  
 exporting data for bulk load 834  
 ExtConnection interface, methods 613  
 ExtDatabaseMetaData interface,  
   methods 619  
 extended connection failover 61  
 external overflow files, bulk load 842  
 ExtLogControl class, methods 620  
 ExtStatementPoolMonitor class 824  
 ExtStatementPoolMonitorMBean  
   interface 824

## F

failover  
   about 58  
 AlternateServers  
   DB2 90  
   Informix 180, 211  
   MySQL 219  
   Oracle 268  
   SQL Server 376, 463  
   Sybase 474, 529  
 DB2 171  
 Informix 211  
 load balancing and 65  
 MySQL 258  
 Oracle 365  
 SQL Server 463  
 Sybase 529  
 types of 58

FailoverGranularity  
   DB2 106  
   Informix 187  
   MySQL 226  
   Oracle 284  
   SQL Server 390  
   Sybase 484  
 FailoverMode  
   DB2 107  
   Informix 188  
   MySQL 227  
   Oracle 285  
   SQL Server 391  
   Sybase 485  
 FailoverPreconnect  
   DB2 108  
   Informix 189  
   MySQL 228  
   Oracle 286  
   SQL Server 392  
   Sybase 486  
 FetchBufferSize (Informix) 189  
 FetchTSWTSasTimestamp (Oracle) 287  
 File System JNDI Provider 49  
 forward-only cursor 707  
 freezing the statement pool 822

## G

generating statement pool export  
   file 823, 859  
 getBestRowIdentifier method, using 715  
 getBlob() method 701  
 getClob() method 701  
 getColumns(), type of metadata returned  
   DB2 95  
   Oracle 273  
 getObject method 708  
 getPooledConnection method 53  
 getTypeInfo() method  
   DB2 631  
   Informix 642

- MySQL 650
- Oracle 662
- SQL Server 670
- Sybase 684
- glossary** 861
- GrantExecute (DB2)** 109
- guidelines for primary and alternate servers 63
  
- H**
- handshake, SSL 70, 864
- help, online 22
- High Availability Disaster Recovery (HADR)** 64, 91, 171
- HostNameInCertificate**
  - DB2 110
  - MySQL 228
  - Oracle 287
  - SQL Server 393
  - Sybase 487
- HostProcess** (alias for **ProgramID**) (SQL Server) 394
  
- I**
- importing statements into the statement pool 820
- ImportStatementPool**
  - DB2 111
  - Informix 190
  - MySQL 230
  - Oracle 289
  - SQL Server 394
  - Sybase 488
- Informix**
  - auto-generated keys 210
  - classes, data source and driver 177
  - client information 204
  - connection properties
- AccountingInfo** 179
- AlternateServers** 180
- ApplicationName** 181
- ClientHostName** 181
- ClientUser** 181
- CodePageOverride** 182
- ConnectionRetryCount** 182
- ConnectionRetryDelay** 183, 223, 479
- ConvertNull** 184
- Database** (alias for **DatabaseName**) 184, 185
- DatabaseName** 185
- DBDate** 186
- FailoverGranularity** 187
- FailoverMode** 188
- FailoverPreconnect** 189
- FetchBufferSize** 189
- for connection failover 215
- ImportStatementPool** 190
- InformixServer** 191
- InitializationString** 191
- InsensitiveResultSetBufferSize** 192
- JavaDoubleToString** 193
- JDBCBehavior** 193
- LoadBalancing** 194
- LoginTimeout** 195
- MaxPooledStatements** 195
- Password** 196
- PortNumber** 197
- ProgramID** 197
- QueryTimeout** 198
- ResultSetMetaDataOptions** 198
- ServerName** 199
- SpyAttributes** 199
- UseDelimitedIdentifier** 200
- User** 201
- connection retry 182
- connection URL 177
- data types 202
- DATABASE** SQL statement, using 185
- date format 186
- DELIMIDENT** environment variable, setting 200

- double quote ("") characters in
  - SQL statements, interpreting 200
- failover 211
- getTypeInfo() method 642
- isolation levels 204
- J2EE Connector Architecture resource
  - adapter class 178
- JSR 114, Rowsets 209
- load balancing 194
- null values, handling for data
  - conversions 184
- parameter metadata 205
- performance considerations 201
- prepared statement pool size, setting 195
- primary server, specifying 212
- ResultSet metadata 198, 207
- Rowsets 209
- scrollable cursors 205
- table name information in ResultSet
  - metadata, returning 198, 242
- timeout, connection 195
- timeout, query 198
- URL 177
- InformixServer (Informix) 191
- initial column buffer size (Oracle) 289
- initial pool size 790
- InitialColumnBufferSize (Oracle) 289
- InitializationString
  - DB2 111
  - Informix 191
  - MySQL 230
  - Oracle 290
  - SQL Server 395
  - Sybase 489
- InOrderColumnAccess (MySQL) 231
- InputStream object (DataDirect Spy) 786, 788
- insensitive cursors
  - access to temporary files required 78
  - performance implications 707
- InsensitiveResultSetBufferSize
  - DB2 112
  - Informix 192
  - MySQL 232
  - Oracle 291
- SQL Server 395
- Sybase 489
- inserting a row (DataDirect Test) 768
- inserting XML
  - DB2 142
  - Oracle 335
  - SQL Server 424
- interactive\_timeout variable (MySQL) 233
- InteractiveClient (MySQL) 233
- interfaces, DataDirect Statement Pool
  - Monitor 824
- IP addresses (IPv4/IPv6) 50
- isolation levels
  - Snapshot isolation level (SQL Server) 447
  - support
    - DB2 162
    - Informix 204
    - MySQL 251
    - Oracle 355
    - SQL Server 447
    - Sybase 522
  - TRANSACTION\_SNAPSHOT constant (SQL Server) 448

## J

- J2EE application server, example using
  - resource adapters with 40
- J2EE Connector Architecture
  - resource adapter class
    - DB2 87
    - Informix 178
    - MySQL 218
    - Oracle 267
    - SQL Server 375
    - Sybase 472
  - Service Provider Interface (SPI) 38

- Java 2 Platform
  - batch execution on a prepared statement (DataDirect Test) 751
  - permissions
    - for establishing connections 77
    - granting access to temporary files 78
    - Java properties, accessing 78
    - Kerberos authentication 79
  - scrollable result sets (DataDirect Test) 748
  - Security Manager 77
  - using drivers on 77
- Java Authentication and Authorization Service (JAAS) login module
- DB2 150
- Oracle 345
- SQL Server 432
- Sybase 515
- Java Management Extensions (JMX) MBean, DataDirect Statement Pool Monitor 57, 811
- Java Naming Directory Interface (JNDI)
  - data source
    - application, using in 49
    - connecting with 29
    - example 49
    - registering with 54
  - initializing environment 50, 54
  - naming service 48
- Java properties, accessing 78
- Java Security Architecture 42, 43
- Java Transaction API 49
- JavaDoubleToString
  - DB2 113
  - Informix 193
  - MySQL 234
  - Oracle 292
  - SQL Server 396
  - Sybase 490
- javax.security.auth.Subject
  - DB2 152
  - Oracle 347
  - SQL Server 434
  - Sybase 517
- JConsole 57, 811
- JDBC 4.0 36
- JDBC data sources
  - See data sources
- JDBC data type mappings
  - DB2 139
  - Informix 202
  - MySQL 248
  - Oracle 330
  - SQL Server 421
  - Sybase 510
- JDBC Driver Manager
  - connecting with 29, 43, 44
  - connection URL format 46
  - registering
    - DataDirect Connect for JDBC drivers 30
    - drivers 45
  - SpyAttributes property, specifying with 784
- JDBC extensions 20, 601
- JDBC support
  - functionality supported 536
  - JDBC specification versions supported 535
  - methods supported for each JDBC object 536
- JDBCBehavior
  - DB2 114
  - Informix 193
  - MySQL 234
  - Oracle 292
  - SQL Server 397
  - Sybase 491
- JNDI Provider for LDAP 49
- JSR 114, Rowsets
  - DB2 169
  - Informix 209
  - MySQL 257
  - Oracle 364
  - SQL Server 460
  - Sybase 526
- JTA support
  - DB2 163
  - Oracle 356
  - SQL Server 449
- transaction

**K**

Kerberos authentication

about

Oracle 340

SQL Server 428

Sybase 512

configuring

DB2 149

Oracle 343

SQL Server 431

Sybase 514

delegation of credentials

DB2 152

Oracle 346

SQL Server 434

Sybase 517

`javax.security.auth.Subject`

DB2 152

Oracle 347

SQL Server 434

Sybase 517

`kinit` command (UNIX and Linux)

DB2 154

Oracle 349

SQL Server 436

Sybase 519

MIT Kerberos

DB2 147

Oracle 341

Sybase 512

permissions

DB2 80

SQL Server 80

Sybase 81

product requirements

DB2 149

Oracle 343, 344

SQL Server 431

Sybase 514

service principal name

DB2 80

Sybase 81, 501

Ticket Granting Ticket (TGT)

DB2 154

Oracle 348

SQL Server 436

Sybase 519

user credentials, specifying

DB2 152

Oracle 346

SQL Server 434

Sybase 517

using with reauthentication 55

Windows Active Directory Kerberos

DB2 147

Oracle 341

SQL Server 429

Sybase 512

Kerberos configuration file

DB2 150

Oracle 344

SQL Server 432

Sybase 515

Kerberos Key Distribution Center (KDC)

name, specifying

DB2 150, 151

Oracle 344, 345

SQL Server 432

Sybase 515, 516

service principal name 80, 81

Kerberos realm 80, 81

KeyPassword  
 DB2 114  
 MySQL 235  
 Oracle 293

keyset-driven cursor  
 See sensitive cursors

KeyStore  
 DB2 115  
 MySQL 235  
 Oracle 293

keystore, SSL 73

KeyStorePassword  
 DB2 115  
 MySQL 236  
 Oracle 294

kinit command, Kerberos authentication  
 DB2 154  
 Oracle 349  
 SQL Server 436  
 Sybase 519

**L**

large binary object (DB2) 133

LDAP 49

LIKE escape character for wildcards escape sequence 727

linked servers (Oracle) 307

literals  
 arguments, using parameter markers 703  
 escape sequences 718

load balancing  
 about 65  
 DB2 116  
 Informix 194  
 MySQL 237  
 Oracle 294  
 SQL Server 398  
 Sybase 492

LoadBalancing  
 DB2 116  
 Informix 194

MySQL 237  
 Oracle 294  
 SQL Server 398  
 Sybase 492

MySQL 237  
 Oracle 294  
 SQL Server 398  
 Sybase 492

Oracle 294  
 SQL Server 398  
 Sybase 492

loading data using bulk load 835

LoadLibraryPath  
 Oracle 295  
 SQL Server 398

LOB support  
 DB2 163  
 executing a query that uses Clob, Blobs (DataDirect Test) 776  
 MySQL 252  
 SQL Server 455  
 Sybase 522

LobCommandSize (MySQL) 237

location used for storing client information for a connection 624

LocationName (DB2) 117

log for DataDirect Spy, using 845

logging for bulk load 836

LoginTimeout  
 DB2 118  
 Informix 195  
 MySQL 238  
 Oracle 296  
 SQL Server 399  
 Sybase 493

long data  
 caching  
 SQL Server 400  
 Sybase 493

retrieving and performance 700

using Blobs and Clob to return  
 MySQL 252  
 SQL Server 455  
 Sybase 522

LongDataCacheSize  
 SQL Server 400  
 Sybase 493

lost connections  
 extended connection failover 61  
 recovering work in progress 62  
 select failover 62

# M

ManagedConnectionFactory class  
 DB2 87  
 Informix 178  
 MySQL 218  
 Oracle 267  
 SQL Server 375  
 Sybase 472  
 man-in-the-middle (MITM) attacks 110, 228, 287, 393, 487  
 mapping Oracle driver properties to tnsnames.ora 323  
 max\_allowed\_packet system variable (MySQL) 237  
 maximum idle time 790  
 maximum pool size 789  
 MaxPooledStatements  
 DB2 118  
 Informix 195  
 MySQL 238  
 Oracle 297  
 SQL Server 401  
 Sybase 494  
 MBean name, registering 812  
 memory usage, tuning  
 BulkLoadBatchSize  
 DB2 94  
 Oracle 272  
 SQL Server 380  
 Sybase 476  
 FetchBufferSize (Informix) 189  
 InitialColumnBufferSize (Oracle) 289  
 InsensitiveResultSetBufferSize  
 DB2 112  
 Informix 192  
 MySQL 232  
 Oracle 291  
 SQL Server 395  
 Sybase 489  
 LongDataCacheSize  
 SQL Server 400  
 Sybase 493

metadata  
 about client information 627  
 methods, minimizing use of 696  
 methods  
 Array object 536  
 Blob object 536  
 CallableStatement object 539  
 Clob object 549  
 Connection object 552  
 ConnectionEventListener object 556  
 ConnectionPoolDataSource object 556  
 DatabaseMetaData object 557  
 DataSource object 566  
 Driver object 567  
 ExtConnection class 613  
 ExtDatabaseMetaData class 619  
 ExtLogControl class 620  
 ExtStatementPoolMonitorMBean interface 824  
 ParameterMetaData object 567  
 PooledConnection object 569  
 PreparedStatement object 570  
 Ref object 576  
 ResultSet object 576  
 ResultSetMetaData object 590  
 RowSet object 592  
 SavePoint object 592  
 StatementEventListener object 598  
 Struct object 598  
 XAConnection object 598  
 XADataSource object 599  
 XAResource object 599  
 Microsoft Cluster Server (MSCS) 64, 376, 463  
 Microsoft SQL Server  
 See SQL Server  
 minimum pool size 789  
 MIT Kerberos  
 Oracle 341  
 Sybase 512  
 MITM attacks 110, 228, 287, 393, 487  
 modifyThreadGroup permission 42  
 multi-lingual applications 82

MySQL  
  alternate server, specifying 259  
  auto-generated keys 257  
  Blobs and Clobs, using to return long  
    data 252  
  certificate validation 246  
  classes, data source and driver 217  
  client information, storing and  
    returning 251  
  connection properties  
    AccountingInfo 219  
    AlternateServers 219  
    ApplicationName 220

**N**

named instances (SQL Server)  
     connecting to 420  
     permissions required for 78  
     specifying connections to for connection failover 465  
**NetAddress (SQL Server)** 402  
**newline characters (DB2)** 126  
**NewPassword (Oracle)** 298  
**NLS\_TIMESTAMP\_FORMAT session parameter (Oracle 9i and higher)** 332  
**NLS\_TIMESTAMP\_TZ\_FORMAT session parameter (Oracle 9i and higher)** 332  
**NTLM authentication**  
     about  
         Oracle 341  
         SQL Server 429  
     configuring  
         Oracle 350  
         SQL Server 437  
**DLL**  
     Oracle 352  
     SQL Server 439  
**product requirements**  
     Oracle 350  
     SQL Server 437  
**null values**  
     handling for data conversions  
         DB2 101  
         Informix 184  
         MySQL 224  
         Oracle 280  
         SQL Server 385  
         Sybase 480  
     interpreting  
         SQL Server 462  
         Sybase 527

**O**

**object**  
     **Array** 536  
     **Blob**, methods 536  
     **CallableStatement**, methods 539  
     **Clob**, methods 549  
**Connection**  
     **ExtLogControl interface** 845  
     managing connections 711  
     methods 552  
     transaction model 714  
**ConnectionEventListener**, methods 556  
**ConnectionPoolDataSource**  
     connecting with 54  
     methods 556  
**DatabaseMetaData** methods 557  
**DataSource**  
     calling in an application 50  
     connecting with 48  
     connection pooling 54, 789, 794, 796, 804, 806, 807  
     definition 862  
     methods 566  
     retrieving 49  
**DBBlob (DB2)** 163  
**DDBulkLoad** 830  
**DDBulkLoad**, methods 603  
**Driver**, methods 567  
**ExtConnection**, methods 613  
**ExtControl**, methods 620  
**ExtDatabaseMetaData**, methods 619  
**InputStream (DataDirect Spy)** 786, 788  
**large binary (DB2)** 133  
**Long (DB2)** 581  
**ParameterMetaData**, methods 567  
**pooled data source** 794  
**PooledConnection**, methods 569  
**PooledConnectionDataSource**  
     connecting with 799  
     connection pooling 794, 796, 803  
**ConnectionPoolMonitor** 809

- creating 794, 796, 803
- methods 804
- PooledConnectionDataSourceFactory
  - methods 803
  - Reference object, obtaining 805
- PreparedStatement
  - methods 570
  - using addBatch() instead of 705
  - using Statement object instead of 704
- Reader (DataDirect Spy) 786, 788
- Ref, methods 576
- Reference, connection pooling 803, 805
- ResultSet
  - database metadata 696
  - generating 696
  - methods 576
  - updating data 714
- ResultSetMetaData, methods 590
- RowSet, methods 592
- SavePoint, methods 592
- Statement 711
  - methods 592
  - using instead of PreparedStatement object 704
  - using multiple 711
- StatementEventListener, methods 598
- Struct, methods 598
- XAConnection, methods 598
- XADatasource, methods 599
- XAResource, methods 599
- ORA\_CHAR vs ORA\_VARCHAR bindings for non-output parameters in a Where clause (Oracle) 307
- Oracle
  - authentication 340
  - auto-generated keys 364
  - batch inserts and updates 271, 356
  - Blob and Clob searches 209
  - cancel request timeout 282
  - classes, data source and driver 265
  - client authentication
    - about 341
    - configuring 352
  - client information 355
- configuring
  - client authentication 352
  - driver to retrieve connection information from tnsnames.ora file 321
  - Kerberos authentication 343
  - tnsnames.ora file 322
  - user ID/password authentication 343
- connection information, retrieving from tnsnames.ora files 319
- connection properties
  - AccountingInfo 268
  - AlternateServers 268
  - ApplicationName 269
  - AuthenticationMethod 270, 341
  - BatchPerformanceWorkaround 271
  - BulkLoadBatchSize 272
  - CatalogOptions 273
  - ClientHostName 274
  - ClientUser 274
  - CodePageOverride 275
  - CommitBehavior 277
  - ConnectionRetryCount 278
  - ConnectionRetryDelay 279
  - ConvertNull 280
  - Database (alias for DatabaseName) 280, 281
  - DatabaseName 281
  - EnableBulkLoad 281
  - EnableCancelTimeout 282
  - EnableServerResultCache 283
  - EncryptionMethod 283
  - FailoverGranularity 284
  - FailoverMode 285
  - FailoverPreconnect 286
  - FetchTSWTSasTimestamp 287
  - for connection failover 369
  - HostNameInCertificate 287
  - ImportStatementPool 289
  - InitialColumnBufferSize 289
  - InitializationString 290
  - InsensitiveResultSetBufferSize 291
  - JavaDoubleToString 292
  - JDBCBehavior 292

KeyPassword 293  
 KeyStore 293  
 KeyStorePassword 294  
 LoadBalancing 294  
 LoadLibraryPath 295  
 LoginTimeout 296  
 MaxPooledStatements 297  
 NewPassword 298  
 Password 299  
 PortNumber 299  
 ProgramID 300  
 QueryTimeout 300  
 ResultSetMetaDataOptions 301  
 SDUSize 302  
 SendFloatParametersAsString 303  
 ServerName 303  
 ServerType 304  
 ServiceName 305  
 SID 306  
 SpyAttributes 306  
 StringParamsMustMatchCharColumns 307  
 SupportLinks 307  
 SysLoginRole 308  
 TNSNamesFile 309  
 TNSServerName 310  
 TrustStore 311  
 TrustStorePassword 311  
 User 312  
 ValidateServerCertificate 312  
 WireProtocolMode 313  
 connection retry 278, 369  
 connection URL 266  
 data types 330  
 date/time data types 332  
 delegation of credentials 346  
 double parameters 303  
 encryption 352  
 encryption, configuring 353  
 example, REF CURSOR 338, 339  
 failover 365  
 float parameters 303  
 getTypeInfo() method 662  
 initial column buffer size 289  
 isolation levels 355  
 J2EE Connector Architecture resource adapter class 267  
 JSR 114, Rowsets 364  
 JTA support 356  
 Kerberos authentication  
     configuring 343  
     Java Authentication and Authorization Service (JAAS) login module 345  
     javax.security.auth.Subject 347  
     Kerberos configuration file 344  
     kinit command (UNIX and Linux) 349  
     MIT Kerberos 341  
     product requirements 343, 344  
     Ticket Granting Ticket (TGT) 348  
     user credentials, specifying 346  
     Windows Active Directory Kerberos 341  
 linked servers 307  
 load balancing 294  
 NLS\_TIMESTAMP\_FORMAT session parameter 332  
 NLS\_TIMESTAMP\_TZ\_FORMAT session parameter 332  
 NTLM authentication  
     about 341  
     configuring 350  
     DLL 352  
     product requirements 350  
 null values, handling for data conversions 280  
 ORA\_CHAR vs ORA\_VARCHAR bindings  
     for non-output parameters in a Where clause 307  
 parameter metadata 357  
 password, changing when a connection is established 298  
 prepared statement pool size, setting 297  
 primary server, specifying 366  
 Real Application Clusters (RAC) 64, 269, 305, 318, 366  
 reauthentication 354  
 REF CURSOR data types 338  
 restricted mode, connecting to 329  
 RESTRICTED SESSION system privilege 329

ResultSet metadata 301, 359  
 RETURNING clause 361  
 Rowsets 364  
 scrollable cursors 356  
 server-side resultset caching 283  
 Session Data Unit (SDU) 302  
 SYSDBA privilege 308  
 SYSOPER privilege 308  
 table name information in ResultSet  
     metadata, returning 301  
 TIME\_ZONE session parameter 332  
 timeout, connection 296  
 timeout, query 300  
 TIMESTAMP data type 333  
 timestamp data types 287  
 TIMESTAMP WITH LOCAL TIME ZONE  
     data type 333  
 TIMESTAMP WITH TIME ZONE data  
     type 333  
 timestamps 332  
 tnsnames.ora file  
     configuring driver to reference 321  
     configuring file 322  
     connect descriptor parameters 323  
     driver property mappings to 323  
     permissions for Java 2 Platform 79  
 TNSServerName 310  
 transaction redo changes 277  
 URL 266  
 user ID/password authentication  
     about 340  
     configuring 343  
 using scrollable cursors 356  
 XML  
     inserting 335  
     updating 335  
 XMLType data type 335  
 orphaned connections (SQL Server) 452  
 outer join escape sequence, example 725  
 overflow files, bulk load 842

## P

package (DB2), creating automatically 134  
 PackageCollection (DB2) 99, 119  
 PackageOwner (DB2) 120  
 PacketSize  
     SQL Server 402  
     Sybase 495  
 parameter markers, using as arguments to  
     stored procedures 703  
 parameter metadata  
     DB2 164  
     Informix 205  
     MySQL 253  
     Oracle 357  
     returning (DataDirect Test) 755  
     SQL Server 456  
     Sybase 524  
 ParameterMetaData object, methods 567  
 Password  
     DB2 120  
     Informix 196  
     MySQL 240  
     Oracle 299  
     SQL Server 403  
     Sybase 496  
 password, changing when a connection is  
     established (Oracle) 298  
 performance optimization  
     application 44  
     auto-generated keys, retrieving 709  
     batches, using instead of prepared  
         statements 705  
     commits, managing 713  
     connection  
         management 711  
         pooling 711  
     database metadata methods 696  
     DB2 workloads 161  
     designing JDBC applications 711  
     encryption 69  
     general 695  
     get methods, using effectively 708

getBestRowIdentifier, using 715  
**Performance Tuning Wizard** 33  
 restricting results to tables in current catalog (DB2) 127  
 result sets, retrieving 702  
 retrieving long data 700  
 scrollable cursors 702  
 selecting JDBC objects and methods 703  
 transaction model, choosing 714  
 update methods of the `ResultSet` object 714  
 updating data 714  
**Performance Tuning Wizard** 33  
**permissions**  
 for establishing connections 77  
 granting access to temporary files, example 78  
 Java properties, granting access 78  
 Kerberos authentication 79  
 named instances (SQL Server) 78  
**PooledConnection** object, methods 569  
**PooledConnectionDataSource** object  
 connecting with 799  
`ConnectionPoolMonitor` 809  
 creating 794, 796, 803  
 methods 804  
 Reference object, obtaining 805  
**PooledConnectionDataSourceFactory** object  
 methods 803  
**poolEntries()** parameters 816  
**PortNumber**  
 DB2 120  
 Informix 197  
 MySQL 240  
 Oracle 299  
 SQL Server 404  
 Sybase 496  
**prepared statement pooling**  
 performance optimization 712  
 setting size of statement pool  
 DB2 118  
 Informix 195  
 MySQL 238  
 Oracle 297  
 SQL Server 401  
 Sybase 494  
**prepared statements**  
 creating stored procedures for (Sybase) 497  
 example, executing multiple times 706  
 using batches instead of 705  
**PreparedStatement** object  
 methods 570  
**performance**  
 implications of using `Statement` object instead 704  
 of prepared statement pool 712  
**prepared statement pooling** 712  
 using `addBatch()` instead of 705  
 using `Statement` object instead of 704  
**PrepareMethod (Sybase)** 497  
**primary server, specifying**  
 DB2 172  
 guidelines for 63  
 Informix 212  
 MySQL 259  
 Oracle 366  
 SQL Server 464  
 Sybase 530  
**procedure call** 727  
**product requirements**  
 Kerberos authentication  
 DB2 149  
 Oracle 343  
 SQL Server 431  
 Sybase 514  
 NTLM authentication  
 Oracle 350  
 SQL Server 437  
**ProgramID**  
 DB2 121  
 Informix 197  
 MySQL 240  
 Oracle 300  
 SQL Server 404  
 Sybase 498  
**ProgramName (alias for ApplicationName)**  
 (SQL Server) 405

## Q

QueryTimeout  
 DB2 122  
 Informix 198  
 MySQL 241  
 Oracle 300  
 SQL Server 405  
 Sybase 499  
 quick start guide to connecting 29

## R

Reader object (DataDirect Spy) 786, 788  
 Real Application Clusters (RAC)  
 (Oracle) 305, 318  
 realm, Kerberos 80, 81  
 reauthentication  
   about 55  
   DB2 158  
   enabling in DataDirect Connection Pool  
     Manager 790  
 example  
   connection pool with 792  
   connection pool without 791  
   performing on a connection  
     explicitly 56  
 Kerberos, using with 55  
 Oracle 354  
 performing explicitly 56  
 SQL Server 444  
 support 55  
 using 55  
 using with the DataDirect Connection Pool  
     Manager 790  
 ReceiveStringParameterType  
   (SQL Server) 406  
 REF CURSOR data types (Oracle) 338  
 Ref object, methods 576  
 Reference object

obtaining for  
 PooledConnectionDataSource 805  
 using to create a  
 PooledConnectionDataSource  
 object 803  
 registering  
   drivers 45  
   MBean name 812  
 ReplacePackage (DB2) 122  
 resource adapters  
   about 37  
   deployment descriptor 39, 42  
   example  
     application using resource adapters  
       directly 40  
       using with a J2EE application server 40  
   J2EE Connector Architecture 37  
   modifyThreadGroup permission 42  
   resource archive (RAR) file 38, 39  
   Security Manager 42  
   security permissions 42  
   Sun Microsystems JDBC Connector 39  
 resource archive (RAR) file 38, 39  
 restricted mode, connecting to (Oracle) 329  
 RESTRICTED SESSION system privilege  
 (Oracle) 329  
 result set type in statement pool 815  
 result sets, scrollable  
   Informix 205  
   Oracle 356  
   performance optimization 702  
   requirements for 748  
   scrolling through a result set  
     (DataDirect Test) 748  
   SQL Server 448  
   Sybase 522  
 ResultSet metadata  
   DB2 167  
   Informix 207  
   MySQL 255  
   Oracle 359  
   SQL Server 458  
   Sybase 524

- ResultSet object
    - database metadata 696
    - generating 696
    - methods 576
    - updating data 714
  - ResultSetMetaData object, methods 590
  - ResultSetMetaDataOptions
    - DB2 123
    - Informix 198
    - MySQL 242
    - Oracle 301
    - SQL Server 407
    - Sybase 499
  - retrying connections
    - See connection retry
  - RETURNING clause (Oracle) 361
  - returning client information 626
  - returning XML
    - DB2 142
    - SQL Server 424
  - rows, deleting (DataDirect Test) 764
  - RowSet object, methods 592
  - Rowsets
    - DB2 169
    - Informix 209
    - MySQL 257
    - Oracle 364
    - SQL Server 460
    - Sybase 526
- 
- ## S
- SavePoint object, methods 592
  - savepoints
    - establishing (DataDirect Test) 757
    - object 592
  - scalar functions 718
  - schemas (DB2)
    - default schema for unqualified objects in dynamically prepared SQL 90
    - for catalog methods 96, 157
    - non-default 157
  - scrollable cursors
    - DB2 162
    - Informix 205
    - MySQL 252
    - Oracle 356
    - SQL Server 448
    - Sybase 522
  - scroll-insensitive result sets
    - DB2 162
    - Informix 205
    - MySQL 252
    - Oracle 356
    - SQL Server 448
    - Sybase 522
  - scroll-sensitive result sets
    - Informix 205
    - Oracle 356
    - SQL Server 448
    - Sybase 522
  - SDUSize (Oracle) 302
  - search patterns, avoiding 697
  - Secure Sockets Layer (SSL)
    - See SSL
  - security
    - types of 67
    - using 67
  - Security Manager
    - and resource adapters 42
    - for Java 2 Platform 77
  - security permissions, resource adapters 42
  - select failover 62
  - SelectMethod
    - SQL Server 407
    - Sybase 500
  - SendFloatParametersAsString (Oracle) 303
  - SendStreamAsBlob (DB2) 124
  - SendStringParametersAsUnicode (SQL Server) 408
  - sensitive cursors
    - Oracle 356
    - performance implications 707
    - SQL Server 448
  - server authentication, SSL 71

ServerName  
 DB2 125  
 Informix 199  
 MySQL 242  
 Oracle 303  
 SQL Server 409  
 Sybase 501  
 server-side  
 RPCs 703  
   updatable cursors (SQL Server) 448  
 server-side resultset caching (Oracle) 283  
 ServerType (Oracle) 304  
 service class, Workload Manager (WLM) 160  
 service principal name for Kerberos  
   authentication  
   DB2 80  
   Sybase 81, 501  
 Service Provider Interface (SPI) 38  
 ServiceName (Oracle) 305  
 ServicePrincipalName (Sybase) 501  
 Session Data Unit (SDU) (Oracle) 302  
 setEnableLogging(), using to turn on and off  
   DataDirect Spy logging 845  
 SID (Oracle) 306  
 Snapshot isolation level, using  
   (SQL Server) 447  
 SpyAttributes  
   DB2 125  
   Informix 199  
   MySQL 243  
   Oracle 306  
   SQL Server 410  
   Sybase 502  
 SQL escape sequences  
   date, time, timestamp 718  
   general 717  
   LIKE escape character for wildcards 727  
   MySQL 251  
   outer join 725  
   procedure call 727  
   scalar functions 718  
 SQL Server  
   alternate database servers, specifying 464  
   ANSI\_NULLS option 462  
   auto-generated keys 460  
   batch inserts and updates 455  
   Blobs and Clobs, using to return long  
    data 455  
   caching long data 400  
   cancel request timeout 388  
   classes, data source and driver 373  
   client information 446  
   connecting to named instances 420  
   connection properties  
     AccountingInfo 376  
     AlternateServers 376  
     AlwaysReportTriggerResults 377  
     ApplicationName 378  
     AuthenticationMethod 379, 429  
     BatchUpdateException 455  
     BulkLoadBatchSize 380  
     BulkLoadOptions 381  
     ClientHostName 382  
     ClientUser 383  
     CodePageOverride 383  
     ConnectionRetryCount 384  
     ConnectionRetryDelay 385  
     ConvertNull 385  
     Database (alias for DatabaseName) 386  
     DatabaseName 386  
     DescribeParameters 386  
     EnableBulkLoad 388  
     EnableCancelTimeout 388  
     EncryptionMethod 389  
     FailoverGranularity 390  
     FailoverMode 391  
     FailoverPreconnect 392  
     for connection failover 468, 533  
     HostNameInCertificate 393  
     HostProcess (alias for ProgramID) 394  
     ImportStatementPool 394  
     InitializationString 395  
     InsensitiveResultSetBufferSize 395  
     JavaDoubleToString 396  
     JDBCBehavior 397  
     LoadBalancing 398  
     LoadLibraryPath 398  
     LoginTimeout 399

LongDataCacheSize 400  
MaxPooledStatements 401  
NetAddress 402  
PacketSize 402  
Password 403  
PortNumber 404  
ProgramID 404  
ProgramName (alias for ApplicationName) 405  
QueryTimeout 405  
ReceiveStringParameterType 406  
ResultSetMetaDataOptions 407  
SelectMethod 407  
SendStringParametersAsUnicode 408  
ServerName 409  
SpyAttributes 410  
TransactionMode 411  
TruncateFractionalSeconds 412  
TrustStore 412  
TrustStorePassword 413  
User 413  
UseServerSideUpdatableCursors 414  
ValidateServerCertificate 414  
WSID (alias for ClientHostName) 415  
XATransactionGroup 415  
XMLDescribeType 416  
connection retry 384, 467  
connection SQL Server 374  
data types 421  
delegation of credentials 434  
distributed transaction cleanup 452  
DML with results 442  
encryption 439  
encryption, configuring 441  
explicit transaction cleanup 453  
failover 463  
getTypeInfo method 670  
getTypeInfo() method 670  
isolation levels 447  
J2EE Connector Architecture resource adapter class 375  
JDBC XA procedures 449  
JSR 114, Rowsets 460  
JTA support 449  
Kerberos authentication 428  
about 428  
configuring 431  
javax.security.auth.Subject 434  
Kerberos configuration file 432  
kinit command (UNIX and Linux) 436  
permissions 80  
product requirements 431  
Ticket Granting Ticket (TGT) 436  
user credentials, specifying 434  
Windows Active Directory Kerberos 429  
load balancing 398  
LOB support 455  
Microsoft Cluster Server (MSCS) 64, 376, 463  
named instances  
connecting to 420  
permissions required for 78  
NTLM authentication  
about 429  
configuring 437  
DLL 439  
product requirements 437  
null values, handling for data conversions 385  
null values, interpreting 462  
orphaned connections 452  
parameter metadata 456  
prepared statement pool size, setting 401  
primary server, specifying 464  
reauthentication 444  
ResultSet metadata 407, 458  
Rowsets 460  
scrollable cursors, using 448  
server-side updatable cursors 448  
Snapshot isolation level, using 447  
SQL Server authentication  
about 428  
configuring 430  
String stored procedure output  
parameters, describing 406  
table name information in ResultSet  
metadata, returning 407  
timeout, connection 399

timeout, query 405  
transaction cleanup  
  methods 452  
  using timeout 452  
  using transaction group identifier 453  
**TRANSACTION\_SNAPSHOT** constant 448  
triggers, filtering results from 377  
URL 374  
user ID/password authentication  
  See SQL Server authentication  
XML  
  inserting 424  
  returning 424  
  updating 424  
SQL Server authentication  
  about 428  
  configuring 430  
sqljdbc.dll 453, 454  
SSL  
  about 70  
  certificate 71  
  Certificate Authority (CA) 71  
client authentication 73  
configuring  
  DB2 156  
  MySQL 250  
  Oracle 353  
  SQL Server 441  
  Sybase 520  
handshake 70, 864  
keystore 73  
server authentication 71  
truststore 71  
  using with Microsoft SQL Server 440  
**Statement** object  
  Connection object association 711  
  methods 592  
  using instead of PreparedStatement  
    object 704  
  using multiple 711  
  when to use 704  
statement pool  
  clearing statements 821  
  concurrency type of result set in 815  
  export file, generating 823  
  freezing 822  
  importing statements to 820  
  result set type in 815  
  statement type in 815  
  unfreezing 822  
statement pool export file  
  example 859  
  generating 859  
statement pooling  
  about 57  
  adding statements to the pool  
    DB2 111  
    Informix 190  
    MySQL 230  
    Oracle 289  
    SQL Server 394  
    Sybase 488  
  enabling 811  
  statement pool export file 859  
  troubleshooting problems 858  
  using 57  
statement type in statement pool 815  
stored procedures  
  errors returned from (Sybase) 484  
  escape sequences 727  
  installing for JTA 449  
  parameter markers as arguments, using  
    703  
  String output parameters, describing  
    (SQL Server) 406  
storing client information 75, 623  
**StringParamsMustMatchCharColumns**  
  (Oracle) 307  
**StripNewlines** (DB2) 126  
**Struct** object, methods 598  
Sun Microsystems JDBC Connector 39  
support  
  authentication 68  
  encryption 70  
**SupportLink** 26  
**SupportLinks** (Oracle) 307

- Sybase
- alternate database servers, specifying 530
  - ansinull option 527
  - authentication 512
  - auto-generated keys 526
  - batch inserts and updates 476, 523
  - Blobs and Clobs, using to return long data 522
  - caching long data 493
  - cancel request timeout 482
  - classes, data source and driver 471
  - client information 521
  - connection properties 473
    - AccountingInfo 473
    - AlternateServers 474
    - ApplicationName 474
    - AuthenticationMethod 475, 513
    - BatchPerformanceWorkaround 476
    - BulkLoadBatchSize 476
    - ClientHostName 477
    - ClientUser 477
    - CodePageOverride 478
    - ConnectionRetryCount 478
    - ConnectionRetryDelay 479
    - ConvertNull 480
    - Database (alias for DatabaseName) 481
    - DatabaseName 481
    - EnableBulkLoad 481
    - EnableCancelTimeout 482
    - EncryptionMethod 483
    - ErrorBehavior 484
    - FailoverGranularity 484
    - FailoverMode 485
    - FailoverPreconnect 486
    - HostNameInCertificate 487
    - ImportStatementPool 488
    - InitializationString 489
    - InsensitiveResultSetBufferSize 489
    - JavaDoubleToString 490
    - JDBCBehavior 491
    - LoadBalancing 492
    - LoginTimeout 493
    - LongDataCacheSize 493
    - MaxPooledStatements 494
  - PacketSize 495
  - Password 496
  - PortNumber 496
  - PrepareMethod 497
  - ProgramID 498
  - QueryTimeout 499
  - ResultSetMetaDataOptions 499
  - SelectMethod 500
  - ServerName 501
  - ServicePrincipalName 501
  - SpyAttributes 502
  - TransactionMode 503
  - TrustStore 504
  - TrustStorePassword 504
  - UseAlternateProductInfo 505
  - User 505
    - ValidateServerCertificate 506
  - connection retry 478, 532
  - connection Sybase 472
  - data types 510
  - delegation of credentials 517
  - driver class 471
  - encryption, configuring 520
  - failover 529
  - getTypeInfo() method 684
  - isolation levels 522
  - J2EE Connector Architecture resource adapter class 472
  - JSR 114, Rowsets 526
  - Kerberos authentication
    - configuring 514
    - Kerberos configuration file 515
    - kinit command (UNIX and Linux) 519
    - MIT Kerberos 512
    - obtaining a
      - javax.security.auth.Subject 517
    - permissions 81
    - product requirements 514
    - service principal name 81, 501
    - Ticket Granting Ticket (TGT) 519
    - user credentials, specifying 517
    - Windows Active Directory Kerberos 512
  - load balancing 492
  - LOB support 522

- native batches 523
  - null values, handling for data conversions 480
  - null values, interpreting 527
  - parameter metadata 524
  - prepared statement pool size, setting 494
  - prepared statements, creating stored procedures for 497
  - primary server, specifying 530
  - ResultSet metadata 499, 524
  - Rowsets 526
  - scrollable cursors, using 522
  - stored procedures
    - creating for prepared statements 497
    - errors returned from 484
  - table name information in ResultSet
    - metadata, returning 499
  - timeout, connection 493
  - timeout, query 499
  - URL 472
  - user ID/password authentication
    - about 512
    - configuring 513
  - synonyms in result sets (DB2) 95
  - SYSDBA privilege (Oracle) 308
  - SysLoginRole (Oracle) 308
  - SYSOPER privilege (Oracle) 308
  - System.out, logging JDBC calls to using DataDirect Spy 787
- 
- ## T
- table name information in ResultSet
    - metadata
      - DB2 123
      - Informix 198
      - MySQL 242
      - Oracle 301
      - SQL Server 407
      - Sybase 499
  - Technical Support, contacting 26
  - testing connections 76
  - Ticket Granting Ticket (TGT), Kerberos
    - authentication
    - DB2 154
    - Oracle 348
    - SQL Server 436
    - Sybase 519
  - time literal escape sequence 718
  - TIME\_ZONE session parameter (Oracle 9i and higher) 332
  - timeout, connection
    - DB2 118
    - Informix 195
    - MySQL 238
    - Oracle 296
    - SQL Server 399
    - Sybase 493
  - timeout, query
    - DB2 122
    - Informix 198
    - MySQL 241
    - Oracle 300
    - SQL Server 405
    - Sybase 499
  - TIMESTAMP data type (Oracle 9i and higher) 333
  - timestamp literal escape sequence 718
  - TIMESTAMP WITH LOCAL TIME ZONE data type (Oracle 9i and higher) 333
  - TIMESTAMP WITH TIME ZONE data type (Oracle 9i and higher) 333
  - timestamps (Oracle 9i and higher) 332
  - tnsnames.ora file
    - configuring
      - driver to reference file 321
      - options for 322
    - connect descriptor parameters 323
    - connection information, retrieving 319
    - mapping Oracle driver properties to 323
    - permissions for Java 2 Platform 79
    - setting path and filename of 309
  - TNSNamesFile (Oracle) 309
  - TNSServerName (Oracle) 310
  - tracing, enabling for Pool Manager 793
  - tracking JDBC calls 783

transaction redo changes (Oracle) 277  
**TRANSACTION\_SNAPSHOT** constant  
     (SQL Server) 448  
**TransactionMode**  
     SQL Server 411  
     Sybase 503  
**transactions**  
     See JTA support  
**TreatBinaryAsChar** (MySQL) 243  
**troubleshooting**  
     application problems 845  
     connection pooling problems 851  
     statement pooling problems 858  
**TruncateFractionalSeconds** (SQL Server) 412  
**TrustStore**  
     DB2 126  
     MySQL 244  
     Oracle 311  
     SQL Server 412  
     Sybase 504  
**truststore file, SSL** 71  
**TrustStorePassword**  
     DB2 127  
     MySQL 245  
     Oracle 311  
     SQL Server 413  
     Sybase 504  
**tuning application performance** 33  
**turning on and off DataDirect Spy**  
     logging 845  
**Type 4 JDBC drivers** 35  
**types of**  
     authentication 67  
     encryption 69  
     failover 58  
     security 67

# U

**unfreezing the statement pool** 822  
**Unicode**  
     string parameters in SQL Server 408  
     support 82  
**unqualified function names (DB2)** 102  
**updatable result sets**  
     DB2 162  
     deleting a row 764  
     Informix 205  
     inserting a row 768  
     MySQL 252  
     Oracle 356  
     SQL Server 448  
     Sybase 522  
**updating a row** 772  
**updating XML**  
     DB2 142  
     Oracle 335  
     SQL Server 424  
**URL**  
     DB2 86  
     Informix 177  
     MySQL 218  
     Oracle 266  
     SQL Server 374  
     Sybase 472  
**UseAlternateProductInfo** (Sybase) 505  
**UseCurrentSchema** (DB2) 127  
**UseDelimitedIdentifier** (Informix) 200  
**User**  
     DB2 128  
     Informix 201  
     MySQL 246  
     Oracle 312  
     SQL Server 413  
     Sybase 505  
**user credentials, specifying for Kerberos**  
     authentication  
     DB2 152  
     Oracle 346

SQL Server 434  
 Sybase 517  
 user ID and password  
   clear text (DB2) 146, 147  
   encryption (DB2) 146, 147  
 user ID/password authentication  
   about  
     DB2 146  
     Oracle 340  
     SQL Server 428  
     Sybase 512  
 configuring  
   DB2 149  
   Oracle 343  
   Sybase 513  
**UseServerSideUpdatableCursors**  
   (SQL Server) 414  
 using  
   bulk load for batch inserts 837  
   CSV files 838  
   DataDirect Spy log 845  
   DataDirect Statement Pool  
     Monitor 21, 811  
   DataDirect-specific methods 815  
   DDBulkLoad object 830  
   get methods effectively 708  
   getBestRowIdentifier method 715  
   getObject method 708  
   JMX API 812  
   reauthentication 55  
   scrollable cursors  
     DB2 162  
     Informix 205  
     MySQL 252  
     Oracle 356  
     SQL Server 448  
     Sybase 522  
   security 67  
   statement pooling 57  
   Wrapper methods 602  
 UTF-16 encoding, converting 82

**V**

ValidateServerCertificate  
   DB2 128  
   MySQL 246  
   Oracle 312  
   SQL Server 414  
   Sybase 506  
 verifying bulk load configuration file 840

**W**

wait\_timeout variable (MySQL) 233  
 Windows Active Directory Kerberos  
   DB2 147  
   Oracle 341  
   SQL Server 429  
   Sybase 512  
 Windows Domain 80, 81  
 WireProtocolMode (Oracle) 313  
 WithHoldCursors (DB2) 129  
 Wizard, performance tuning 33  
 Workload Manager (WLM) 75, 160, 621  
 Workload Manager (WLM) service class 160  
 Wrapper methods 602  
 WSID (alias for ClientHostName)  
   (SQL Server) 415

**X**

XAConnection object 598  
 XAConnection object, methods 598  
 XADataSource object 599  
 XADataSource object, methods 599  
 XAResource object 599  
 XAResource object, methods 599  
 XAResource.recover method 452, 454  
 XATransactionGroup (SQL Server) 415

XML  
  inserting  
    DB2 142  
    Oracle 335  
    SQL Server 424  
  returning  
    DB2 142  
    SQL Server 424  
  updating  
    DB2 142  
    Oracle 335  
    SQL Server 424  
XML data type (DB2) 142  
xml data type (SQL Server) 424  
XMLDescribeType  
  DB2 130  
  SQL Server 416  
XMLType data type (Oracle 9i and  
higher) 335

