Oracle® Fusion Middleware

Type 4 JDBC Drivers for Oracle WebLogic Server

11*g* Release 1 (10.3.3)

E13753-02

April 2010

This document is a resource for software developers and system administrators who develop and support applications that use the Java Database Connectivity (JDBC) API.

ORACLE®

Oracle Fusion Middleware Type 4 JDBC Drivers for Oracle WebLogic Server, 11*g* Release 1 (10.3.3)

E13753-02

Contents

Preface xi

Documentation Accessibility xi
Conventions xi

1 Introduction and Roadmap

1.1 Document Scope and Audience 1-1
1.2 Guide to this Document 1-1
1.3 Related Documentation 1-2
1.4 JDBC Samples and Tutorials 1-2
1.4.1 Avitek Medical Records Application (MedRec) and Tutorials 1-2
1.4.2 JDBC Examples in the WebLogic Server Distribution 1-2
1.5 New and Changed JDBC Features in This Release 1-3

2 Using WebLogic Type 4 JDBC Drivers

2.1 JDBC Specification Compliance 2-1
2.2 Installation 2-2
2.3 Supported Databases 2-2
2.4 Connecting Through WebLogic JDBC Data Sources 2-2
2.5 Specifying Connection Properties 2-2
2.5.1 Limiting Connection Creation Time with LoginTimeout 2-3
2.6 Using IP Addresses 2-3
2.7 Using Security 2-4
2.7.1 Authentication 2-4
2.7.1.1 Kerberos Authentication Requirements 2-5
2.7.1.2 NTLM Authentication Requirements 2-6
2.7.2 Data Encryption Across the Network 2-7
2.7.3 SSL Encryption 2-8
2.7.3.1 SSL Server Authentication 2-8
2.7.3.2 SSL Client Authentication (DB2 Drivers) 2-9
2.8 Required Permissions for the Java Security Manager 2-10
2.8.1 Permissions for Establishing Connections 2-10
2.8.2 Granting Access to Java Properties 2-11
2.8.3 Granting Access to Temporary Files 2-11
2.8.4 Permissions for Kerberos Authentication 2-12
2.8.4.1 DB2 2-12

2.8.4.2 Microsoft SQL Server 2-12
2.8.4.3 Sybase 2-13
2.9 XA Support 2-13
2.10 Unicode Support 2-13
2.11 Error Handling 2-14
2.11.1 Driver Errors 2-14
2.11.2 Database Errors 2-14

3 The DB2 Driver

3.1 DB2 Driver Classes 3-2
3.2 J2EE Connector Architecture Resource Adapter Class 3-2
3.3 DB2 URL 3-2
3.4 DB2 Connection Properties 3-2
3.5 Performance Considerations 3-29
3.5.1 CatalogIncludesSynonyms 3-29
3.5.2 CatalogOptions 3-29
3.5.3 CatalogSchema 3-30
3.5.4 EnableBulkLoad 3-30
3.5.5 EncryptionMethod 3-30
3.5.6 InsensitiveResultSetBufferSize 3-30
3.5.7 MaxPooledStatements 3-30
3.5.8 SendStreamAsBlob 3-31
3.5.9 StripNewLines 3-31
3.5.10 UseCurrentSchema 3-31
3.6 Setting the locationName on AS/400 3-31
3.7 Creating a DB2 Package 3-32
3.7.1 Creating a DB2 Package Using dbping 3-32
3.7.2 Creating a DB2 Package Using Connection Properties 3-32
3.7.2.1 Example for DB2 for Linux/UNIX/Windows: 3-33
3.7.2.2 Example for DB2 for z/OS and iSeries: 3-33
3.7.3 Notes About Increasing Dynamic Sections in the DB2 Package 3-33
3.8 Data Types 3-34
3.9 Returning and Inserting/Updating XML Data 3-36
3.9.1 Returning XML Data 3-36
3.9.1.1 Character Data 3-36
3.9.1.2 Binary Data 3-37
3.9.2 Inserting/Updating XML Data 3-37
3.9.2.1 Character Data 3-37
3.9.2.2 Binary Data 3-38
3.10 Authentication 3-38
3.10.1 Using the AuthenticationMethod Property 3-39
3.10.2 Configuring User ID/Password Authentication 3-39
3.10.3 Configuring Kerberos Authentication 3-40
3.10.3.1 Product Requirements 3-40
3.10.3.2 Configuring the Driver 3-40
3.10.4 Specifying User Credentials for Kerberos Authentication 3-41
3.10.5 Obtaining a Kerberos Ticket Granting Ticket 3-42

3.10.6 Configuring Client Authentication 3-43
3.11 Data Encryption 3-43
3.11.1 Configuring SSL Encryption 3-43
3.12 Non-Default Schemas for Catalog Methods 3-44
3.13 Reauthentication 3-45
3.14 SQL Escape Sequences 3-46
3.15 Isolation Levels 3-46
3.16 Using Scrollable Cursors 3-46
3.17 JTA Support 3-46
3.18 Large Object (LOB) Support 3-46
3.19 Batch Inserts and Updates 3-47
3.20 Parameter Metadata Support 3-47
3.20.1 Insert and Update Statements 3-47
3.20.2 Select Statements 3-48
3.20.3 Stored Procedures 3-49
3.21 ResultSet Metadata Support 3-49
3.22 Rowset Support 3-50
3.23 Auto-Generated Keys Support 3-50
3.24 Database Connection Property 3-51
3.25 DatabaseName Connection Property 3-51
3.26 New Data Types 3-51
3.27 SQL Procedures for z/OS 3-52
3.28 IPv6 Support 3-52
3.29 Bulk Load 3-52

4 The Informix Driver

4.1 Informix Driver Classes 4-1
4.2 Informix URL 4-1
4.3 Informix Connection Properties 4-2
4.3.1 Informix Limitation for Prepared Statements 4-14
4.4 Performance Considerations 4-15
4.4.1 FetchBufferSize 4-15
4.4.2 InsensitiveResultSetBufferSize 4-15
4.4.3 MaxPooledStatements 4-15
4.4.4 ResultSetMetaDataOptions 4-15
4.5 Data Types 4-16
4.6 Client Information for Connections 4-17
4.7 SQL Escape Sequences 4-17
4.8 Isolation Levels 4-17
4.9 Using Scrollable Cursors 4-17
4.10 Parameter Metadata Support 4-17
4.10.1 Insert and Update Statements 4-17
4.10.2 Select Statements 4-17
4.10.3 Stored Procedures 4-18
4.11 ResultSet MetaData Support 4-18
4.12 Rowset Support 4-19
4.13 Blob and Clob Searches 4-20

4.14 Auto-Generated Keys Support ... 4-20
4.15 Configuring Failover ... 4-20
4.15.1 Specifying Primary and Alternate Servers ... 4-21
4.15.2 Specify Connection Retry ... 4-22
4.15.3 Failover Properties ... 4-23

5 The Sybase Driver

5.1 Driver Classes ... 5-1
5.2 Sybase URL ... 5-2
5.3 J2EE Connector Architecture Resource Adapter Class ... 5-2
5.4 Sybase Connection Properties ... 5-2
5.5 Performance Considerations ... 5-24
5.5.1 BatchPerformanceWorkaround ... 5-25
5.5.2 EnableBulkLoad ... 5-25
5.5.3 EncryptionMethod ... 5-25
5.5.4 InsensitiveResultSetBufferSize ... 5-25
5.5.5 LongDataCacheSize ... 5-25
5.5.6 MaxPooledStatements ... 5-25
5.5.7 PacketSize ... 5-26
5.5.8 PrepareMethod ... 5-26
5.5.9 ResultSetMetaDataOptions ... 5-26
5.5.10 SelectMethod ... 5-26
5.6 Data Types ... 5-26
5.7 Authentication ... 5-28
5.7.1 Using the AuthenticationMethod Property ... 5-28
5.7.2 Configuring User ID/Password Authentication ... 5-29
5.7.3 Configuring Kerberos Authentication ... 5-29
5.7.3.1 Product Requirements ... 5-29
5.7.3.2 Configuring the Driver ... 5-29
5.7.4 Specifying User Credentials for Kerberos Authentication ... 5-30
5.7.5 Obtaining a Kerberos Ticket Granting Ticket ... 5-32
5.8 Data Encryption ... 5-32
5.9 Client Information for Connections ... 5-33
5.10 SQL Escape Sequences ... 5-33
5.11 Isolation Levels ... 5-33
5.12 Using Scrollable Cursors ... 5-33
5.13 Large Object (LOB) Support ... 5-33
5.14 Batch Inserts and Updates ... 5-34
5.15 Parameter Metadata Support ... 5-34
5.16 ResultSet MetaData Support ... 5-34
5.17 Rowset Support ... 5-35
5.18 Auto-Generated Keys Support ... 5-35
5.19 NULL Values ... 5-36
5.20 Sybase JTA Support ... 5-37
5.21 Configuring Failover ... 5-37
5.21.1 Specifying Primary and Alternate Servers ... 5-37
5.21.2 Specify Connection Retry ... 5-39

5.21.3 Failover Properties 5-39
5.22 Bulk Load 5-40

6 The MS SQL Server Driver

6.1 Driver Class 6-2
6.2 Microsoft SQL Server URL 6-2
6.3 Connecting to Named Instances 6-2
6.4 SQL Server Connection Properties 6-3
6.5 Performance Considerations 6-31
6.5.1 EnableBulkLoad 6-32
6.5.2 EncryptionMethod 6-32
6.5.3 InsensitiveResultSetBufferSize 6-32
6.5.4 LongDataCacheSize 6-32
6.5.5 MaxPooledStatements 6-32
6.5.6 PacketSize 6-32
6.5.7 ResultSetMetaDataOptions 6-33
6.5.8 SelectMethod 6-33
6.5.9 SendStringParametersAsUnicode 6-33
6.5.10 SnapshotSerializable 6-33
6.5.11 UseServerSideUpdatableCursors 6-33
6.6 Data Types 6-34
6.7 Returning and Inserting/Updating XML Data 6-36
6.7.1 Returning XML Data 6-36
6.7.1.1 Character Data 6-36
6.7.1.2 Binary Data 6-37
6.7.2 Inserting/Updating XML Data 6-37
6.7.2.1 Character Data 6-37
6.7.2.2 Binary Data 6-38
6.8 Authentication 6-38
6.8.1 Using the AuthenticationMethod Property 6-39
6.8.2 Configuring SQL Server Authentication 6-39
6.8.3 Configuring Kerberos Authentication 6-39
6.8.3.1 Product Requirements 6-39
6.8.3.2 Configuring the Driver 6-40
6.8.4 Specifying User Credentials for Kerberos Authentication 6-41
6.8.5 Obtaining a Kerberos Ticket Granting Ticket 6-42
6.8.6 Configuring NTLM Authentication 6-43
6.8.6.1 Product Requirements 6-43
6.8.6.2 Configuring the Driver 6-43
6.9 Data Encryption 6-44
6.9.1 Using SSL with Microsoft SQL Server 6-45
6.9.2 Configuring SSL Encryption 6-45
6.10 DML with Results (Microsoft SQL Server 2005 and Higher) 6-46
6.11 Reauthentication 6-47
6.12 Client Information for Connections 6-47
6.13 SQL Escape Sequences 6-47
6.14 Isolation Levels 6-47

6.15 Using the Snapshot Isolation Level (Microsoft SQL Server 2005 and Higher) 6-48
6.16 Using Scrollable Cursors 6-48
6.17 Server-Side Updatable Cursors 6-48
6.18 Installing Stored Procedures for JTA 6-49
6.19 Distributed Transaction Cleanup 6-50
6.19.1 Transaction Timeout 6-50
6.19.2 Explicit Transaction Cleanup 6-51
6.20 Large Object (LOB) Support 6-51
6.21 Batch Inserts and Updates 6-51
6.22 Parameter Metadata Support 6-52
6.22.1 Insert and Update Statements 6-52
6.22.2 Select Statements 6-52
6.22.3 Stored Procedures 6-53
6.23 ResultSet MetaData Support 6-53
6.24 Rowset Support 6-54
6.25 Auto-Generated Keys Support 6-54
6.26 Null Values 6-55
6.27 Configuring Failover 6-55
6.28 Specifying Primary and Alternate Servers 6-56
6.29 Specifying Connection Retry 6-58
6.30 Failover Properties 6-59
6.31 Bulk Load 6-59

A JDBC Support

A.1 JDBC Compatibility A-2
A.2 Supported Functionality A-2
A.2.1 Array Object A-2
A.2.2 Blob Object A-2
A.2.3 CallableStatement Object A-4
A.2.4 Clob Object A-12
A.2.5 Connection Object A-14
A.2.6 ConnectionEventListener Object A-16
A.2.7 ConnectionPoolDataSource Object A-17
A.2.8 DatabaseMetaData Object A-17
A.2.9 Data Source Object A-27
A.2.10 Driver Object A-27
A.2.11 ParameterMetaData Object A-28
A.2.12 PooledConnection Object A-29
A.2.13 PreparedStatement Object A-30
A.2.14 Ref Object A-34
A.2.15 ResultSet Object A-34
A.2.16 ResultSetMetaData Object A-44
A.2.17 RowSet Object A-46
A.2.18 SavePoint Object A-46
A.2.19 Statement Object A-46
A.2.20 StatementEventListener Object A-50
A.2.21 Struct Object A-50

A.2.22 XAConnection Object A-50
A.2.23 XADataSource Object A-51
A.2.24 XAResource Object A-51

B GetTypeInfo

B.1 DB2 Driver B-1
B.2 Informix Driver B-15
B.3 SQL Server Driver B-27
B.4 Sybase Driver B-47

C SQL Escape Sequences for JDBC

C.1 Date, Time, and Timestamp Escape Sequences C-1
C.2 Scalar Functions C-1
C.3 Outer Join Escape Sequences C-5
C.4 LIKE Escape Character Sequence for Wildcards C-6
C.5 Procedure Call Escape Sequences C-6

D Tracking JDBC Calls with WebLogic JDBC Spy

D.1 Configuring WebLogic JDBC Data Sources for WebLogic JDBC Spy D-1
D.2 WebLogic JDBC Spy URL Attributes D-2
D.3 WebLogic JDBC Spy Log Example D-3

Preface

This preface describes the document accessibility features and conventions used in this guide—*Type 4 JDBC Drivers for Oracle WebLogic Server*.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.

Convention	Meaning
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Introduction and Roadmap

This section describes the contents and organization of this guide—*WebLogic Server Type 4 JDBC Drivers.*

- Section 1.1, "Document Scope and Audience"
- Section 1.2, "Guide to this Document"
- Section 1.3, "Related Documentation"
- Section 1.4, "JDBC Samples and Tutorials"
- Section 1.5, "New and Changed JDBC Features in This Release"

1.1 Document Scope and Audience

This document is a resource for software developers and system administrators who develop and support applications that use the Java Database Connectivity (JDBC) API. It also contains information that is useful for business analysts and system architects who are evaluating WebLogic Server. The topics in this document are relevant during the evaluation, design, development, pre-production, and production phases of a software project.

It is assumed that the reader is familiar with Java EE and EJB concepts. This document emphasizes the value-added features provided by WebLogic Server EJBs and key information about how to use WebLogic Server features and facilities to get an EJB application up and running.

1.2 Guide to this Document

- This chapter, Chapter 1, "Introduction and Roadmap," introduces the organization of this guide.
- Chapter 2, "Using WebLogic Type 4 JDBC Drivers" provides information about connecting to a database with WebLogic Type 4 JDBC drivers.
- Chapter 3, "The DB2 Driver" provides detailed information about the DB2 driver.
- Chapter 4, "The Informix Driver" provides detailed information about the Informix driver.
- Chapter 6, "The MS SQL Server Driver" provides detailed information about the Microsoft SQL Server driver.
- Chapter 5, "The Sybase Driver," provides detailed information about the Sybase driver.

- Appendix A, "JDBC Support" lists support for standard and extension JDBC methods.
- Appendix B, "GetTypeInfo" provides results returned from the method DataBaseMetaData.getTypeinfo for all of the WebLogic Type 4 JDBC drivers.
- Appendix C, "SQL Escape Sequences for JDBC" describes the scalar functions supported for the WebLogic Type 4 JDBC drivers. Your data store may not support all of these functions.
- Appendix D, "Tracking JDBC Calls with WebLogic JDBC Spy" describes how to configure the WebLogic JDBC Spy, which logs JDBC usage.

1.3 Related Documentation

This document contains JDBC-specific driver information.

For comprehensive guidelines for developing, deploying, and monitoring WebLogic Server applications, see the following documents:

- *Programming JDBC for Oracle WebLogic Server* is a guide to designing and using JDBC connections in your applications.
- *Configuring and Managing JDBC for Oracle WebLogic Server* is a guide to JDBC configuration and management for WebLogic Server.
- *Developing Applications for Oracle WebLogic Server* is a guide to developing WebLogic Server applications.
- *Deploying Applications to Oracle WebLogic Server* is the primary source of information about deploying WebLogic Server applications.
- *Performance and Tuning for Oracle WebLogic Server* contains information on monitoring and improving the performance of WebLogic Server applications.

1.4 JDBC Samples and Tutorials

In addition to this document, Oracle provides a variety of JDBC code samples and tutorials that show JDBC configuration and API use, and provide practical instructions on how to perform key JDBC development tasks.

1.4.1 Avitek Medical Records Application (MedRec) and Tutorials

MedRec is an end-to-end sample Java EE application shipped with WebLogic Server that simulates an independent, centralized medical record management system. The MedRec application provides a framework for patients, doctors, and administrators to manage patient data using a variety of different clients.

MedRec demonstrates WebLogic Server and Java EE features, and highlights Oracle-recommended best practices. MedRec is included in the WebLogic Server distribution, and can be accessed from the Start menu on Windows machines. For Linux and other platforms, you can start MedRec from the *WL_HOME*`\samples\domains\medrec` directory, where *WL_HOME* is the top-level installation directory for WebLogic Server.

1.4.2 JDBC Examples in the WebLogic Server Distribution

WebLogic Server optionally installs API code examples in *WL_HOME*`\samples\server\examples\src\examples`, where *WL_HOME* is the top-level directory of your WebLogic Server installation. You can start the examples

server, and obtain information about the samples and how to run them from the WebLogic Server Start menu.

1.5 New and Changed JDBC Features in This Release

For a comprehensive listing of the new WebLogic Server features introduced in this release, see *What's New in Oracle WebLogic Server*

2

Using WebLogic Type 4 JDBC Drivers

Oracle WebLogic Type 4 JDBC drivers from DataDirect provide JDBC high-performance access through WebLogic Server to industry-leading data stores across the Internet and intranets. The WebLogic Type 4 JDBC drivers are optimized for the Java environment, allowing you to incorporate Java technology and extend the functionality and performance of your existing system.

The WebLogic Type 4 JDBC drivers from DataDirect are proven drivers that:

- Support performance-oriented and enterprise functionality such as distributed transactions, savepoints, multiple open result sets and parameter metadata.
- Are Java EE Compatibility Test Suite (CTS) certified and tested with the largest JDBC test suite in the industry.
- Include tools for testing and debugging JDBC applications.

The following sections provide more information about the WebLogic Type 4 JDBC drivers:

- Section 2.1, "JDBC Specification Compliance"
- Section 2.2, "Installation"
- Section 2.3, "Supported Databases"
- Section 2.4, "Connecting Through WebLogic JDBC Data Sources"
- Section 2.5, "Specifying Connection Properties"
- Section 2.6, "Using IP Addresses"
- Section 2.7, "Using Security"
- Section 2.8, "Required Permissions for the Java Security Manager"
- Section 2.9, "XA Support"
- Section 2.10, "Unicode Support"
- Section 2.11, "Error Handling"

2.1 JDBC Specification Compliance

Oracle WebLogic Type 4 JDBC drivers are compliant with the JDBC 3.0 specification In addition, the WebLogic Type 4 JDBC drivers support the following JDBC 4.0 specification features:

- Connection validation
- Client information storage and retrieval

- Auto-load driver classes (when using Java SE 6)

For details, see Appendix A, "JDBC Support."

2.2 Installation

WebLogic Type 4 JDBC drivers are installed with WebLogic Server in the *WL_HOME*\server\lib folder, where *WL_HOME* is the directory in which you installed WebLogic Server. Driver class files are included in the manifest classpath in weblogic.jar, so the drivers are automatically added to your classpath on the server.

> **Note:** The WebLogic Type 4 JDBC drivers are installed by default when you perform a complete installation of WebLogic Server. If you choose a custom installation, ensure that the WebLogic JDBC Drivers option is selected (checked). If this option is unchecked, the drivers are not installed.

The WebLogic Type 4 JDBC drivers are not included in the manifest classpath of the WebLogic client jar files (for example: wlclient.jar). To use the drivers with a WebLogic client, you must copy the following files to the client and add them to the classpath on the client:

- For DB2: wldb2.jar
- For Informix: wlinformix.jar
- For MS SQL Server: wlsqlserver.jar
- For Sybase: wlsybase.jar

2.3 Supported Databases

For information on database support, see http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html.

2.4 Connecting Through WebLogic JDBC Data Sources

To use the WebLogic Type 4 JDBC drivers, you create a JDBC data source in your WebLogic Server configuration and select the JDBC driver to create the physical database connections in the data source. Applications can then look up the data source on the JNDI tree and request a connection.

See the following related information:

- For information about JDBC and data sources in WebLogic Server, see *Configuring and Managing JDBC for Oracle WebLogic Server*.
- For information about requesting a connection from a data source, see "Obtaining a Client Connection Using a DataSource" in *Programming JDBC for Oracle WebLogic Server*.

2.5 Specifying Connection Properties

You specify connection properties for connections in a data source using the WebLogic Server Administration Console, command line interface, or JMX API. Connection

properties vary by DBMS. For the list of the connection properties specific to each WebLogic Type 4 JDBC driver, see the appropriate driver chapter:

- For the DB2 driver, see Section 3.4, "DB2 Connection Properties."
- For the Informix driver, see Section 4.3, "Informix Connection Properties."
- For the MS SQL Server driver, see Section 6.4, "SQL Server Connection Properties."
- For the Sybase driver, see Section 5.4, "Sybase Connection Properties."

2.5.1 Limiting Connection Creation Time with LoginTimeout

When creating database connections in a JDBC data source, if the database is unavailable, the request may hang until the default system timeout expires. On some systems this can be as long as 9 minutes. The request will hang for each connection in the JDBC data source. To minimize this hang time, you can specify a `LoginTimeout` value for the connection. All WebLogic Type 4 JDBC Drivers support the `LoginTimeout` connection property. When you specify a `LoginTimeout` connection property and the connection is not created immediately, the request waits for the time you specify. If the connection cannot be created within the time specified, the driver throws an SQL exception.

For details on configuring connection properties, see the appropriate driver chapter:

- Section 3.4, "DB2 Connection Properties"
- Section 4.3, "Informix Connection Properties"
- Section 6.4, "SQL Server Connection Properties"
- Section 5.4, "Sybase Connection Properties"

2.6 Using IP Addresses

The WebLogic Type 4 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 2–1). In addition, to connect to IPv6 addresses, the driver machine requires J2SE 5.0 or higher on Windows and J2SE 1.4 on UNIX/Linux.

Table 2–1 IP Address Formats Supported by the WebLogic Type 4 JDBC Drivers

Driver	IPv4	IPv6
DB2	All supported versions	DB2 v9.1 for z/OS DB2 V9.1 for Linux/UNIX/Windows and higher DB2 V5R2 for iSeries and higher
Informix	All supported versions	Informix 10 and higher
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:weblogic: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:weblogic: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 WebLogic Type 4 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:weblogic: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:weblogic: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

2.7 Using Security

The WebLogic Type 4 JDBC drivers support the following security features: authentication and data encryption.

2.7.1 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. WebLogic Type 4 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, and Sybase. For 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 2–2 shows the authentication methods supported by the WebLogic Type 4 JDBC drivers.

Table 2–2 Authentication Methods Supported by the WebLogic Type 4 JDBC Drivers

Driver	UserID/Password	Kerberos	Client	NTLM
DB2 for Linux/UNIX/Windows	X	X	X	N/A
DB2 for z/OS	X	X	X	N/A
DB2 for iSeries	X	N/A	X	N/A
Informix	X	N/A	N/A	N/A
Microsoft SQL Server	X	X[1]	N/A	X
Sybase	X	X	N/A	N/A

[1] Supported for Microsoft SQL Server 2000 and higher.

2.7.1.1 Kerberos Authentication Requirements

Verify that your environment meets the requirements listed in Table 2–3 before you configure your driver for Kerberos authentication.

Table 2–3 Kerberos Authentication Requirements for the Drivers

Component	Requirements
Database server	The database server must be running one of the following databases: DB2: ■ DB2 v8.1 or higher for Linux/UNIX/Windows Microsoft SQL Server: ■ Microsoft SQL Server 2005 ■ Microsoft SQL Server 2000 ■ Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher Sybase: ■ Sybase 12.0 or higher

Table 2–3 (Cont.) Kerberos Authentication Requirements for the Drivers

Component	Requirements
Kerberos server	The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos Key Distribution Center (KDC). If using Windows Active Directory, this machine is also the domain controller. DB2 and Sybase: Network authentication must be provided by one of the following methods: ■ Windows Active Directory on one of the following operating systems: Windows Server 2003 Windows 2000 Server Service Pack 3 or higher ■ MIT Kerberos 1.4.2 or higher Microsoft SQL Server: Network authentication must be provided by Windows Active Directory on one of the following operating systems: ■ Windows Server 2003 ■ Windows 2000 Server Service Pack 3 or higher
Client	J2SE 1.4.2 or higher must be installed.

To use Kerberos authentication, some configuration is required after installation of the WebLogic JDBC Type 4 drivers. See the individual driver chapters for details about configuring authentication.

2.7.1.2 NTLM Authentication Requirements

Verify that your environment meets the requirements listed in Table 2–4 before you configure the driver for NTLM authentication.

Table 2–4 NTLM Authentication Requirements for the Drivers

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: Microsoft SQL Server: ■ Microsoft SQL Server 2005 ■ Microsoft SQL Server 2000 Service Pack 3 or higher ■ Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher
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: ■ Windows Server 2003 ■ Windows 2000 Server Service Pack 3 or higher

Table 2–4 (Cont.) NTLM Authentication Requirements for the Drivers

Component	Requirements
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: ■ Windows Vista ■ Windows Server 2003 ■ Windows XP Service Pack 1 or higher ■ Windows 2000 Service Pack 4 or higher ■ Windows NT 4.0 In addition, J2SE 1.3 or higher must be installed.

To use NTLM authentication, minimal configuration is required after installation of the WebLogic JDBC Type 4 drivers. See the individual driver chapters for details about configuring authentication.

2.7.2 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.

WebLogic Type 4 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 for these databases. See Section 3.11, "Data Encryption" 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 WebLogic Type 4 JDBC drivers.

Table 2–5 Data Encryption Methods Supported by the WebLogic Type 4 JDBC Drivers

Driver	Database-Specific	SSL
DB2 for Linux/UNIX/Windows	X	X[1]

Table 2–5 (Cont.) Data Encryption Methods Supported by the WebLogic Type 4 JDBC

Driver	Database-Specific	SSL
DB2 for z/OS	X	NA
DB2 for iSeries	NA	NA
Informix	N/A	N/A
Microsoft SQL Server	N/A	X[2]
Sybase	N/A	X

[1] Supported for DB2 V5R3 and higher for iSeries
[2] Supported for Microsoft SQL Server 2000 and higher.

2.7.3 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: SSL client authentication is supported with DB2 only.

See the individual driver chapters for details about configuring SSL.

2.7.3.1 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:\Certficates\MyTruststore
  ```

and

```
TrustStorePassword=MyTruststorePassword
```

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

Alternatively, you can configure the WebLogic Type 4 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.

2.7.3.2 SSL Client Authentication (DB2 Drivers)

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 WebLogic Type 4 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:\Certficates\MyKeyStore
  ```

  and

  ```
  KeyStorePassword=MyKeystorePassword
  ```

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

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

2.8 Required Permissions for the Java Security Manager

Using the WebLogic Type 4 JDBC drivers with the Java Security Manager enabled requires certain permissions to be set in the security policy file of the domain. WebLogic Server provides a sample security policy file that you can edit and use. The file is located at *WL_HOME*`\server\lib\weblogic.policy`. The `weblogic.policy` file includes all necessary permissions for the drivers. If you use the `weblogic.policy` file without changes, you may not need to grant any further permissions. If you use another security policy file or if you use driver features that require additional permissions, see the following sections for details about required permissions.

Note: Web browser applets running in the Java 2 plug-in are always running in a JVM with the Java Security Manager enabled.

For more information about using the Java Security Manager with WebLogic Server, see "Using Java Security to Protect WebLogic Resources" in *Programming Security for Oracle WebLogic Server*.

2.8.1 Permissions for Establishing Connections

To establish a connection to the database server, the WebLogic Type 4 JDBC drivers must be granted the permissions as shown in the following examples. You must grant permissions to the jar for your specific database management system. You can grant the permissions to all JAR files in the directory or just to the specific files.

For all JAR files in the directory:

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

For individual JAR files:

```
//For DB2:
grant codeBase "file:WL_HOME${/}server${/}lib${/}wldb2.jar" {
```

```
   permission java.net.SocketPermission "*", "connect";
};
//For Informix:
grant codeBase "file:WL_HOME${/}server${/}lib${/}wlinformix.jar" {
   permission java.net.SocketPermission "*", "connect";
};
//For MS SQL Server:
grant codeBase "file:WL_HOME${/}server${/}lib${/}wlsqlserver.jar" {
   permission java.net.SocketPermission "*", "connect";
};
//For Sybase:
grant codeBase "file:WL_HOME${/}server${/}lib${/}wlsybase.jar" {
   permission java.net.SocketPermission "*", "connect";
};
```

where *WL_HOME* is the directory in which you installed WebLogic Server.

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:WL_HOME${/}server${/}lib${/}-" {
   permission java.net.SocketPermission "*", "listen, connect, accept";
};
```

2.8.2 Granting Access to Java Properties

To allow the WebLogic Type 4 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:WL_HOME${/}server${/}lib${/}-" {
   permission java.util.PropertyPermission "false", "read";
   permission java.util.PropertyPermission "user.name", "read";
   permission java.util.PropertyPermission "user.language", "read";
   permission java.util.PropertyPermission "user.country", "read";
   permission java.util.PropertyPermission "os.name", "read";
permission java.util.PropertyPermission "os.arch", "read";
permission java.util.PropertyPermission "java.specification.version",
      "read";
};
```

where WL_HOME is the directory in which you installed WebLogic Server.

You can also grant these permissions to individual files as shown in Section 2.8.1, "Permissions for Establishing Connections."

2.8.3 Granting Access to Temporary Files

Access to the temporary directory specified by the JVM configuration must be granted in the security policy file, typically in the security policy file used by the JVM in the *JAVA_HOME*`/jre/lib/security` folder. To use insensitive scrollable cursors or to perform client-side sorting of DatabaseMetaData result sets, all code bases must have access to temporary files. The following example shows permissions that have been granted for the `C:\TEMP` directory:

```
// permissions granted to all domains
grant codeBase "file:WL_HOME${/}server${/}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 *WL_HOME* is the directory in which you installed WebLogic Server.

You can also grant these permissions to individual files as shown in Section 2.8.1, "Permissions for Establishing Connections."

2.8.4 Permissions for Kerberos Authentication

To use Kerberos authentication with the WebLogic Type 4 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 WebLogic Type 4 JDBC drivers, see the appropriate driver chapters.

2.8.4.1 DB2

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 example.

```
grant codeBase "file:/WL_HOME/server/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:

- *WL_HOME* is the directory in which you installed WebLogic Server.
- *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.

2.8.4.2 Microsoft SQL Server

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 example.

```
grant codeBase "file:/WL_HOME/server/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:

- *WL_HOME* is the directory in which you installed WebLogic Server.
- *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.

2.8.4.3 Sybase

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 example.

```
grant codeBase "file:/WL_HOME/server/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:

- *WL_HOME* is the directory in which you installed WebLogic Server.
- *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.

2.9 XA Support

Although the WebLogic Type 4 JDBC drivers support XA, you may need to configure your database to support XA with the drivers. See the following sections for more details:

- For DB2, see Section 3.17, "JTA Support."
- For Microsoft SQL Server, see Section 6.18, "Installing Stored Procedures for JTA."
- For Sybase, see Section 5.20, "Sybase JTA Support."

2.10 Unicode Support

Multi-lingual applications can be developed on any operating system platform with JDBC using the WebLogic Type 4 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 WebLogic Type 4 JDBC 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.

2.11 Error Handling

The WebLogic Type 4 JDBC drivers report errors to the calling 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

2.11.1 Driver Errors

An error generated by a WebLogic Type 4 JDBC driver has the following format:

```
[OWLS][WebLogic Type 4 JDBC driver name]message
```

For example:

```
[OWLS][SQLServer 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.

2.11.2 Database Errors

An error generated by the database has the following format:

```
[OWLS][WebLogic Type 4 JDBC driver name][DBMS name] message
```

For example:

```
[OWLS][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 following sections describe how to configure and use the WebLogic Type 4 JDBC driver for DB2:

- Section 3.1, "DB2 Driver Classes"
- Section 3.2, "J2EE Connector Architecture Resource Adapter Class"
- Section 3.3, "DB2 URL"
- Section 3.4, "DB2 Connection Properties"
- Section 3.5, "Performance Considerations"
- Section 3.6, "Setting the locationName on AS/400"
- Section 3.7, "Creating a DB2 Package"
- Section 3.8, "Data Types"
- Section 3.9, "Returning and Inserting/Updating XML Data"
- Section 3.10, "Authentication"
- Section 3.11, "Data Encryption"
- Section 3.12, "Non-Default Schemas for Catalog Methods"
- Section 3.13, "Reauthentication"
- Section 3.14, "SQL Escape Sequences"
- Section 3.15, "Isolation Levels"
- Section 3.16, "Using Scrollable Cursors"
- Section 3.17, "JTA Support"
- Section 3.18, "Large Object (LOB) Support"
- Section 3.19, "Batch Inserts and Updates"
- Section 3.20, "Parameter Metadata Support"
- Section 3.21, "ResultSet Metadata Support"
- Section 3.22, "Rowset Support"
- Section 3.23, "Auto-Generated Keys Support"
- Section 3.24, "Database Connection Property"
- Section 3.25, "DatabaseName Connection Property"
- Section 3.26, "New Data Types"

- Section 3.27, "SQL Procedures for z/OS"
- Section 3.28, "IPv6 Support"
- Section 3.29, "Bulk Load"

3.1 DB2 Driver Classes

The driver classes for the WebLogic Type 4 JDBC DB2 driver are as follows:

```
XA: weblogic.jdbcx.db2.DB2DataSource
Non-XA: weblogic.jdbc.db2.DB2Driver
```

Use these driver classes when configuring a JDBC data source in your WebLogic Server domain.

3.2 J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the Informix resource adapter is:

```
com.weblogic.resource.spi.InformixManagedConnectionFactory
```

3.3 DB2 URL

The connection URL format for the DB2 driver is:

```
jdbc:weblogic: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 Section 2.6, "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 Section 3.4, "DB2 Connection Properties."

For example:

DB2 UDB for Linux, UNIX, and Windows

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

DB2 UDB for z/OS and iSeries

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

3.4 DB2 Connection Properties

Table 3–1 lists the JDBC connection properties supported by the DB2 driver, and describes each property. You can use these connection properties in a JDBC data source configuration in your WebLogic Server domain.

Note: 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.

To specify a property, use the following form in the JDBC data source configuration: *property=value*.

Table 3–1 DB2 Connection Properties

Property	Description
AccountingInfo	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 for database administration/monitoring purposes Data Type: String Valid Values: *string* where *string* is the accounting information. The default value is an empty string.
AddToCreateTable	A string that is appended to the end of all CREATE statements. This field is primarily for users who need to add an "in database" clause. Data Type: String Valid Values: *string* where *string* is the set of characters appended to all CREATE statements.
AllowImplicitResultSetCloseForXA	True or false. DB2 provides a mechanism that automatically closes a result set when all rows of the result set have been fetched. This mechanism increases application performance by reducing the number of database round trips. The WebLogic DB2 driver uses this mechanism by default. **Note:** Problems have been noted when using this mechanism. As a workaround, you should add AllowImplicitResultSetCloseForXA=false to the properties in your data source configuration. The default is true.
AlternateID	Sets the default DB2 schema used by unqualified SQL identifiers to the specified value. 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.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description

AlternateServers	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.
	Data type: String
	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 `DatabaseName` property.
	`jdbc:weblogic:db2://server1:50000;DatabaseName=TEST;User=test;Password=secret;AlternateServers=(server2:50000;DatabaseName=TEST2,server3:50000;DatabaseName=TEST3)`
	Default: None

ApplicationName	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. Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. Data Type: String Default is empty string.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
AuthenticationMethod	Determines which authentication method the driver uses when establishing a connection. Valid Values: `Kerberos`, `encryptedUIDPassword`, `encryptedPassword`, `clearText` , or `client`. If `kerberos`, the driver uses Kerberos authentication. The driver ignores any user ID or password specified. If `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 (see the `EncryptionMethod` property for details). If `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 (see the `EncryptionMethod` property for details). If set to `clearText` (the default), 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 (see the `EncryptionMethod` property for details). If `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. The `User` property provides the user ID. The `Password` property provides the password. If the specified authentication method is not supported by the DB2 server, the connection fails and the driver throws an exception. The default is `clearText`. See Section 3.10, "Authentication" for more information about using authentication with the DB2 driver.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
BatchPerformanceWorkaround	The DB2 driver uses the native DB2 batch mechanism. This property determines whether certain restrictions are enforced to facilitate data conversions. Valid Values: `true` | `false` ■ When set to `false`, the methods used to set the parameter values of a batch operation performed using a `PreparedStatement` must match the database data type of the column the parameter is associated with. This is because DB2 servers do not perform implicit data conversions. ■ When set to `true`, restrictions are removed; however, parameter sets may not be executed in the order they were specified. The default is `false`. See Section 3.19, "Batch Inserts and Updates" for more information. **Note:** For data sources used as a JMS JDBC store that use the WebLogic Type 4 JDBC driver for DB2, the `BatchPerformanceWorkaround` property must be set to `true`.
BulkLoadBatchSize	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. Valid Values: *x* where *x* is a positive integer. The default is 2048. Data Type: long

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
CatalogIncludesSynonyms	Determines whether synonyms are included in the result sets returned from the `DatabaseMetaData.getColumns()` method. Data Type: `True` or `false`. If set to `true` (the default), synonyms are included in the result sets returned from the `DatabaseMetaData.getColumns()` method. If set to `false`, synonyms are omitted from result sets returned from the `DatabaseMetaData.getColumns()` method. See Section 3.5, "Performance Considerations" for information about configuring this property for optimal performance. The default is `true`.
CatalogOptions	Determines which type of metadata information is included in result sets when an application calls DatabaseMetaData methods. Valid Values: 0 | 2 | 6 and the default value is 2. Data Type: int If 0, result sets do not contain synonyms. If 2, result sets contain synonyms that are returned from the following `DatabaseMetaData` methods: `getColumns()`, `getExportedKeys()`, `getFunctionColumns()`, `getFunctions()`, `getImportedKeys()`, `getIndexInfo()`, `getPrimaryKeys()`, `getProcedureColumns()`, and `getProcedures()`. If 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.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
CatalogSchema	The DB2 schema to use for catalog functions. The value must be the name of a valid DB2 schema. The default depends on the platform of the DB2 database. Valid Values: *string* where *string* is the name of a valid DB2 schema. The default is SYSCAT (DB2 for Linux/UNIX/Windows); SYSIBM (DB2 for z/OS); or QSYS2 (DB2 for iSeries). Data Type: String 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 Section 3.12, "Non-Default Schemas for Catalog Methods" for the required views of catalog tables. See Section 3.5, "Performance Considerations" for information about configuring this property for optimal performance.
CharsetFor65535	The code page to use to convert character data 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 stored as bit data retrieved from the database using columns defined with *CCSID 65535* is converted using the specified code page. The value must be a string containing the name of a valid code page supported by your JVM, for example, *CharsetFor65535=CP950*. This property has no effect when writing data to character columns defined with *CCSID 65535*.
ClientHostName	The host name of the client machine to be stored in the database. This value sets the CURRENT CLIENT_WRKSTNNAME register (DB2 for Linux/UNIX/Windows) or CLIENT WRKSTNNAME 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 host name of the client machine. 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. Default is empty string. Data Type: String

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
ClientUser	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: *string* where *string* is a valid user ID. 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. Default is empty string. Data Type: String
CodePageOverride	A code page to be used to convert `Character` and `Clob` data. The specified code page overrides the default database code page or column collation. All `Character` and `Clob` data retrieved from or written to the database is converted using the specified code page. The value must be a string containing the name of a valid code page supported by your JVM, for example, `CodePageOverride=CP950`. Data Type: String 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.
CollectionId (DEPRECATED)	This property is recognized for backward compatibility, but we recommend that you use the `PackageCollection` property instead to specify the name of the collection or library (group of packages) to which DB2 packages are bound.
ConnectionRetryCount	The number of times the driver retries connection attempts until a successful connection is established. Valid Values: 0 | *x* where *x* is any postive integer. Data Type: int If 0, the driver does not retry connections if a successful connection is not established on the driver's first attempt to create a connection. 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. The `ConnectionRetryDelay` property specifies the wait interval, in seconds, used between retry attempts. The default is 5.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
ConnectionRetryDelay	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 0, the driver does not delay between retries. If *x*, the driver waits between connection retry attempts the specified number of seconds. The default is 1.
ConvertNull	Controls how data conversions are handled for null values. Valid Values: 0 | 1 Data Type: int If 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 1 (the default), 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.
CreateDefaultPackage	Determines whether the driver automatically creates required DB2 packages. Valid Values: `true` | `false` If `true`, the DB2 driver automatically creates required DB2 packages, even if they already exist. Existing DB2 packages are replaced by the new packages. If `false` (the default), the driver determines if the required DB2 packages exist. If they do not, the driver automatically creates them. For DB2 for Linux/UNIX/Windows, this property must be used in conjunction with the `ReplacePackage` property. For DB2 for z/OS and DB2 for iSeries, DB2 packages are created in the collection or library specified by the `PackageCollection` property. For more information about creating DB2 packages, see Section 3.7, "Creating a DB2 Package."

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
CurrentFunctionPath	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. Data Type: String Default is null.
Database	An alias for the `DatabaseName` property.
DatabaseName (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: *string* where *string* is the name of a DB2 database. Default is 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:weblogic:db2://server1:50000;DatabaseName=jdbc;Database=acct;User=test;Password=secret` See also Section 3.24, "Database Connection Property."
DynamicSections	The maximum number of prepared statements that the DB2 driver can have open at any time. The value must be a positive integer. The default is 200.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
EnableBulkLoad	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 `true`, the driver uses the native bulk load protocols for batch inserts. Any value set for `BatchPerformanceWorkaround` is ignored. If `false`, the driver uses the batch mechanism for batch inserts. Data Type: boolean Default is `false`.
EnableCancelTimeout	Determines whether a cancel request 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` Data Type: boolean If `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 `false` (the default), the cancel request does not time out.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
EncryptionMethod	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 LoginTimeout property to avoid problems when connecting to a server that does not support SSL. Valid Values: `noEncryption` | `DBEncryption` | `requestDBEncryption` | `SSL` If `noEncryption`, data is not encrypted or decrypted. If `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 `AuthenticationMethod` property must be set to a value of `clearText`, `encryptedPassword`, or `encryptedUIDPassword`. This value is not supported for DB2 for iSeries. If `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 `AuthenticationMethod` property must be set to a value of `clearText`, `encryptedPassword`, or `encryptedUIDPassword`. This value is not supported for DB2 for iSeries. If `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)` ■ `KeyStore (for SSL client authentication)` ■ `KeyStorePassword (for SSL client authentication)` ■ `KeyPassword (for SSL client authentication)` ■ `TrustStore` ■ `TrustStorePassword` ■ `ValidateServerCertificate` Data Type: String The default is `noEncryption`.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
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. This property is ignored if `FailoverMode=connect`. Valid Values: `nonAtomic` | `atomic` | `atomicWithRepositioning` | `disableIntegrityCheck` If `nonAtomic`, the driver continues with the failover process and posts any exceptions on the statement on which they occur. If `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 `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 `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`. Data Type: String Default is `nonAtomic`.
FailoverMode	Specifies the type of failover method the driver uses. Valid Values: `connect` | `extended`| `select` If `connect`, the driver provides failover protection for new connections only. If `extended`, the driver provides failover protection for new and lost connections, but not any work in progress. If `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. Data Type: String Default is `connect`.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
FailoverPreconnect	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 `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 `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. Data Type: boolean Default is `false`.
Grantee	The name of the schema to which you want to grant `EXECUTE` privileges for DB2 packages. This property is ignored if the `GrantExecute` property is set to `false`. Valid Values: *`string`* where *`string`* is a valid DB2 schema. Data Type: String 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 is `PUBLIC`.
GrantExecute	Determines which DB2 schema is granted `EXECUTE` privileges for DB2 packages. Valid Values: `true` | `false` Data Type: boolean If `true`, `EXECUTE` privileges are granted to the schema that is specified by the `Grantee` property. If `false`, `EXECUTE` privileges are granted to the schema that created the DB2 packages. The default is `true`.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
HostNameInCertificate	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`, 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. Data Type: String Default is empty string.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
ImportStatementPool	Specifies the path and file name of the file to be used to load the contents of the statement pool. When this property is specified, statements are imported into the statement pool from the specified file. If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver throws an exception. Valid Values: *string* where *string* is the path and file name of the file to be used to load the contents of the statement pool. Data Type: String Default is empty string.
InitializationString	Specifies one or multiple SQL commands to be executed by the driver after it has established the connection to the database and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver throws an exception indicating which SQL command or commands failed. Valid Values: *command*[[;*command*]...] where *command* is a SQL command. 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. Data Type: String

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
InsensitiveResultSetBufferSize	Determines the amount of memory used by the driver to cache insensitive result set data. Valid Values -1 | 0 | *x* Data Type: int If -1, the driver caches all insensitive result set data in memory. If the size of the result set exceeds available memory, an `OutOfMemoryException` is generated. Because the need to write result set data to disk is eliminated, the driver processes the data more efficiently. If 0, the driver caches all 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 *x*, where *x* is a positive integer that specifies the size (in KB) of the memory buffer used to cache insensitive result set data. If the size of the result set data exceeds the buffer size, 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 more efficient memory use. The default is 2048 (KB)
JavaDoubleToString	Determines whether the driver uses its internal conversion algorithm or the JVM conversion algorithm when converting double or float values to string values. Valid Values: `true` | `false` If `true`, the driver uses the JVM algorithm when converting double or float values to string values. If `false`, the driver uses its internal algorithm when converting double or float values to string values. Using this value improves performance; however, slight rounding differences can occur when compared to the same conversion using the JVM algorithm. These differences are within the allowable error of the double and float data types. The default is `false`.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
JDBCBehavior	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 Data Type: int If 0, the driver describes the data types as JDBC 4.0 data types when using Java SE 6. If 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 `ResultSet.getHoldability()` returns the value of the JDBC 3.0 method `Connection.getHoldability()`. Default is 1.
KeyPassword	Specifies the password that is used to access the individual keys in the keystore file when SSL is enabled (`EncryptionMethod=SSL`) and SSL client authentication is enabled on the database server. This property is useful when individual keys in the keystore file have a different password than the keystore file. Valid Values: *`string`* where *`string`* is a valid password. Data Type: String Default: None
KeyStore	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. Data Type: String Default: None

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
KeyStorePassword	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. Data Type: String Default: None
LoadBalancing	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 `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 `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

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
LocationName (REQUIRED)	The name of the DB2 location that you want to access. For DB2 for z/OS, your system administrator can determine the name of your DB2 location using the following command: `DISPLAY DDF`. 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. For `WRKRDBDIRE`, this property is supported only for DB2 for z/OS and DB2 for iSeries. Valid Values: stringwhere string is the DB2 location. 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.
LoginTimeout	The amount of time, in seconds, the driver waits for a connection to be established before returning control to the application and throwing a timeout exception. Valid Values 0 | x where x is a positive integer. If 0, the driver does not time out a connection request. If 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

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
MaxPooledStatements	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 0, the driver's internal prepared statement pooling is not enabled. If set to *x*, the driver enables the 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`
MaxStatements	An alias for the `MaxPooledStatements` property.
PackageCollection	The name of the collection or library (group of packages) to which DB2 packages are bound. This property is ignored for DB2 for Linux/UNIX/Windows. **Note:** This property replaces the `CollectionId` property; however, the `CollectionId` property is still recognized for backward compatibility. If both the `PackageCollection` and `CollectionId` properties are specified, the `CollectionId` property is ignored. See Section 3.7, "Creating a DB2 Package" for more information about creating DB2 packages. The default is NULLID.
PackageOwner	The owner to be used for any DB2 packages that are created. See Section 3.7, "Creating a DB2 Package" for more information about creating DB2 packages. The default is NULL. Data Type: String

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
Password	A case-sensitive password used to connect to your DB2 database. A password is required only if security is enabled on your database. If so, contact your system administrator to get your password. Valid Values: *string* where *string* is a valid password. The password is case-sensitive. Default: None Data Type: String
PortNumber	The TCP port on which the database server listens for connections. The default is 50000.
ProgramID	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 *DDJVVRRM* where: ■ *DD*J 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
QueryTimeout	Positive integer, -1, or zero (0). Sets the default query timeout (in seconds) for all statements created by a connection. If set to a positive integer, 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. 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), the default query timeout is infinite (the query does not time out). The default is 0.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
ReplacePackage	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 `CreateDefaultPackage` property. Valid Values: `true` | `false` If `true`, the current bind process replaces the existing DB2 packages that are used by the driver. If `false`, the current bind process does not replace the existing DB2 packages. Default: `false` Data Type: boolean
ResultSetMetaDataOptions	The DB2 driver can return table name information in the ResultSet metadata for Select statements if your application requires that information. Valid Values 0 | 1 If set to 0 (the default) and the `ResultSetMetaData.getTableName()` method is called, the DB2 driver does not perform additional processing to determine the correct table name for each column in the result set. In this case, 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 DB2 driver performs additional processing to determine the correct table name for each column in the result set. 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 information about configuring this property for optimal performance, see Section 3.5, "Performance Considerations." The default is 0. Data Type: int
SecurityMechanism (DEPRECATED)	This property is recognized for backward compatibility, but we recommend that you use the `AuthenticationMethod` property to set the authentication method used by the driver.

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
SendStreamAsBlob	Determines whether binary stream data that is less than 32K bytes is sent to the database as Long Varchar for Bit Data or Blob data. Binary streams that are larger than 32K bytes can only be inserted into a Blob column. The driver always sends binary stream data larger than 32K bytes to the database as Blob data. Valid Values: `true` | `false` If `true`, the driver sends binary stream data that is less than 32K 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 `false`, the driver sends binary stream data that is less than 32K 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. See Section 3.5, "Performance Considerations" for information about configuring this property for optimal performance. The default is `false`. Data Type: boolean
ServerName (REQUIRED)	Specifies either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the 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 DB2Server Default: None Data Type: String

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
SpyAttributes	Enables DataDirect Spy to log detailed information about calls that are issued by the driver on behalf of the application. Spy is not enabled by default. Valid Values: `(`*`spy_attribute`*`[;`*`spy_attribute`*`]...)` where *`spy_attribute`* is any valid Spy attribute. See Section D, "Tracking JDBC Calls with WebLogic JDBC Spy" 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	Specifies whether new-line characters in a SQL statement are sent to the DB2 server. If you know that the SQL statements used in your application do not contain newline characters, instructing the driver to not remove them eliminates parsing by the DB2 server and improves performance. Valid Values: `true` | `false` If `true`, the driver removes all newline characters from SQL statements. If `false`, the driver does not remove any newline characters from SQL statements. Default: `false` Data Type: boolean See Section 3.5, "Performance Considerations" for information about configuring this property for optimal performance.
TrustStore	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. Defaul:t None Data Type: String

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
TrustStorePassword	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	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 `true`, results that are returned from the `getTables()` and `getColumns()` methods are restricted to tables and views in the current schema. If `false` (the default), results of the `getTables()` and `getColumns()` methods are not restricted. See Section 3.5, "Performance Considerations" for information about configuring this property for optimal performance. The default is false. Data Type: boolean
User	The case-sensitive user name 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

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
ValidateServerCertificate	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 `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 `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
WithHoldCursors	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 `true`, the cursor behavior is `Preserve`. If `false`, the cursor behavior is `Delete`. The default is `true`. Data Type: boolean

Table 3–1 (Cont.) DB2 Connection Properties

Property	Description
XMLDescribeType	Determines whether the driver maps XML data to the CLOB or BLOB data type. Valid Values: `clob | blob` If `clob`, the driver maps XML data to the `CLOB` data type. If `blob`, the driver maps XML data to the `BLOB` data type. See Section 3.9, "Returning and Inserting/Updating XML Data" for more information. The default is `clob`. Data Type: String

3.5 Performance Considerations

Setting the following connection properties for the DB2 driver as described in the following list can improve performance for your applications:

- Section 3.5.1, "CatalogIncludesSynonyms"
- Section 3.5.2, "CatalogOptions"
- Section 3.5.3, "CatalogSchema"
- Section 3.5.4, "EnableBulkLoad"
- Section 3.5.5, "EncryptionMethod"
- Section 3.5.6, "InsensitiveResultSetBufferSize"
- Section 3.5.7, "MaxPooledStatements"
- Section 3.5.8, "SendStreamAsBlob"
- Section 3.5.9, "StripNewLines"
- Section 3.5.10, "UseCurrentSchema"

3.5.1 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. The driver always returns synonyms for the `DatabaseMetaData.getColumns()` method when accessing DB2 v8.x and higher for Linux/UNIX/Windows.

3.5.2 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()`.

3.5.3 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 Section 3.12, "Non-Default Schemas for Catalog Methods" for views for catalog tables that must exist in the specified schema.

3.5.4 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.

3.5.5 EncryptionMethod

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

3.5.6 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.

3.5.7 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.

3.5.8 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.

3.5.9 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.

3.5.10 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.

3.6 Setting the locationName on AS/400

When connecting to a DB2 database running on AS/400, you must set the `locationName` property:

1. Obtain the "Relational Database" value by executing the `WRKRDBDIRE` command on AS/400.

   You should see output similar to the following:

   ```
   ,Relational,,Remote,Option,,Database,,Location,,Text,
   ,          ,,      ,      ,,S10B757B,,*LOCAL  ,,    ,
   ```

2. In the Java client, set up the `Properties` object with "user" and "password" DB2 connection properties (see Section 3.4, "DB2 Connection Properties").

3. In `Driver.connect()`, specify the following string and the Properties object as parameters:

   ```
   jdbc:weblogic:db2://<Host>:<Port>;LocationName=RelationalDatabaseName
   ```

   In this example, *`RelationalDatabaseName`* is the value of `Database` obtained from the result of running the `WRKRDBDIRE` command.

The following is an excerpt of the Java client:

```
...
Properties props = new Properties();
props.put("user",     user);
props.put("password", password);
...
myDriver = (Driver)Class.forName("weblogic.jdbc.db2.DB2Driver").newInstance();
conn = myDriver.connect("jdbc:weblogic:db2://10.1.4.1:446;LocationName=S10B757B",
props);
stmt = conn.createStatement();
stmt.execute("select * from MYDATABASE.MYTABLE");
rs = stmt.getResultSet();
...
```

3.7 Creating a DB2 Package

A DB2 package is a control structure on the DB2 server produced during program preparation that is used to execute SQL statements. The DB2 driver automatically creates all DB2 packages required at connection time. If a package already exists, the driver uses the existing package to establish a connection.

Note: 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.

By default, DB2 packages created by the DB2 driver contain 200 dynamic sections and are created in the NULLID collection (or library). In most cases, you do not need to create DB2 packages because the DB2 driver automatically creates them at connection time. If required, you can create DB2 packages in either of the following ways:

- Manually force the DB2 driver to create a package using the WebLogic Server `dbping` utility. See Section 3.7.1, "Creating a DB2 Package Using dbping."
- Automatically create a package by setting specific connection properties in the connection URL or data source. See Section 3.7.2, "Creating a DB2 Package Using Connection Properties."

Note: Your user ID must have CREATE PACKAGE privileges on the database, or your database administrator must create packages for you.

Your user ID (the user ID listed in the JDBC data source configuration) must be the owner of the package.

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.

3.7.1 Creating a DB2 Package Using dbping

To create a package on the DB2 server with the WebLogic Type 4 JDBC DB2 driver, you can use the WebLogic Server `dbping` utility. The `dbping` utility is used to test the connection between your client machine and a DBMS via a JDBC driver. Because the WebLogic Type 4 JDBC DB2 driver automatically creates a DB2 package if one does not already exist, running this utility creates a default DB2 package on the DB2 server.

For details about using the `dbping` utility to create a DB2 package, see "Creating a DB2 Package with dbping" in *Command Reference for Oracle WebLogic Server*.

3.7.2 Creating a DB2 Package Using Connection Properties

You can create a DB2 package automatically by specifying specific connection properties in the initial connection URL. Table 3–2 lists the connection properties you should use in your initial connection URL when you create a DB2 package:

Note: This method is not recommended for use with WebLogic Server JDBC data sources because every connection in the data source uses the same URL and connection properties. When a JDBC data source with multiple connections is created, the package would be recreated when each database connection is created.

Table 3–2 Connection Properties for an Initial Connection URL When Creating DB2 Packages

Property	Database
PackageCollection=collection_name, (where *collection_name* 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

Using `CreateDefaultPackage=TRUE` creates a package with a default name. If you use `CreateDefaultPackage=TRUE`, and you do not specify a CollectionId, the NULLID CollectionId is created.

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`.

3.7.2.1 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:weblogic:db2://server1:50000;DatabaseName=SAMPLE;
CreateDefaultPackage=TRUE;ReplacePackage=TRUE;DynamicSections=400
```

3.7.2.2 Example for DB2 for z/OS and iSeries:

The following URL creates DB2 packages with 400 dynamic sections.

```
jdbc:weblogic:db2://server1:50000;LocationName=SAMPLE;
CreateDefaultPackage=TRUE;DynamicSections=400
```

3.7.3 Notes About Increasing Dynamic Sections in the DB2 Package

A dynamic section is the actual executable object that contains the logic needed to satisfy a dynamic SQL request. These sections are used for handles and prepared statements and the associated result sets.

In some cases, you may need to create DB2 packages with more than the default number of dynamic sections (200). Consider the following information if your application requires DB2 packages with a large number of dynamic sections:

- Creating DB2 packages with a large number of dynamic sections may exhaust certain server resources. In particular, you may need to increase the database parameter `PCKCACHE_SZ` to allow the larger packages to be created.
- The creation of more dynamic sections will slow down the initial creation of the DB2 package.
- Using DB2 packages with a large number of dynamic sections may impact application performance. If a small number of sections are in use at one time, there will be no impact on the application. If a large number of sections are in use at one time, the performance of the application may decrease because the database will expend resources to check all open sections for locks.
- As the number of open sections increases, so does the likelihood that a deadlock situation may occur.
- If your application is mostly executing select statements, it is best to operate in the default mode of automatically committing the database. Dynamic sections are not freed in the DB2 package until the database is committed even if the statements are closed in the application. In this mode the database will commit every time a SQL statement is executed and free all of the sections that were opened. If you need to operate in a manual commit mode, then it is advisable to commit the database as often as possible to ensure that all server resources are freed in a timely manner.
- Statements cached in the WebLogic Server prepared statement cache will keep sections in use so that the prepared statements can be reused.
- The DB2 server has a limit on dynamic sections. It is possible to try to create more sections than the server will allow you to create.

3.8 Data Types

Table 3–3 lists the data types supported by the DB2 driver and how they are mapped to JDBC data types.

Table 3–3 DB2 Data Types

DB2 Data Type	JDBC Data Type
Bigint[1]	BIGINT
Binary[2]	BINARY
Blob[3]	BLOB
Char	CHAR
Char for Bit Data	BINARY
Clob	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).
Date	DATE
DBClob[4]	CLOB
Decfloat[5]	DECIMAL
Decimal	DECIMAL

Table 3–3 (Cont.) DB2 Data Types

DB2 Data Type	JDBC Data Type
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[6]	VARBINARY
Smallint	SMALLINT
Time	TIME
Timestamp	TIMESTAMP
Varbinary	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	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 v8.1 and v 8.2 for Linux/UNIX/Windows.

[2] Supported only for DB2 v8.1 and v 8.2 for Linux/UNIX/Windows.

[3] Supported only for DB2 v8.1 and v 8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries (see Section 3.18, "Large Object (LOB) Support").

[4] Supported only for DB2 v8.1 and v 8.2 for Linux/UNIX/Windows, DB2 7.x v8.1, and v8.2 for z/OS, and DB2 V5R2 and V5R3 for iSeries (see Section 3.18, "Large Object (LOB) Support").

[5] Supported only for DB2 v8.1 and v 8.2 for Linux/UNIX/Windows, DB2 7.x v8.1, and v8.2 for z/OS, and DB2 V5R2 and V5R3 for iSeries (see Section 3.18, "Large Object (LOB) Support").

[6] Supported only for DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries.

See Section 3.18, "Large Object (LOB) Support" for more information about the Blob, Clob, and DBClob data types. See Section 3.9, "Returning and Inserting/Updating XML Data" for more information about the XML data type.See Appendix B, "GetTypeInfo" for more information about data types.

3.9 Returning and Inserting/Updating XML Data

For DB2 V9.1 for Linux/UNIX/Windows, the DB2 driver supports the XML data type. By default, the driver maps the XML data type to the JDBC CLOB data type, but you can choose to map the XML data type to the BLOB data type by setting the `XMLDescribeType` connection property to a value of blob.

3.9.1 Returning XML Data

The driver can return XML data as character or binary data. For example, given 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);
```

The driver returns the XML data from the database as character or binary data depending on the setting of the `XMLDescribeType` property. By default, the driver maps the XML data type to the JDBC CLOB data type. If the following connection URL mapped the XML data type to the BLOB data type, the driver would return the XML data as binary data instead of character data:

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

3.9.1.1 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.getClob()
CallableStatement.getString()
CallableStatement.getClob()
```

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

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

```
ResultSet.getAsciiStream()
```

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

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

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.

3.9.1.2 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.

3.9.2 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.

3.9.2.1 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.

3.9.2.2 Binary Data

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

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

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

3.10 Authentication

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See Section 2.7.1, "Authentication" 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 Section 3.10.1, "Using the AuthenticationMethod Property" for information about setting the value for this property.

3.10.1 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.

3.10.2 Configuring User ID/Password Authentication

To configure user ID/password authentication:

1. Set the `AuthenticationMethod` property to encryptedUIDPassword, encryptedPassword, or clearText (the default). See Section 3.10.1, "Using the AuthenticationMethod Property" 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.

3.10.3 Configuring Kerberos Authentication

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

3.10.3.1 Product Requirements

Verify that your environment meets the requirements listed in Table 3–4 before you configure the driver for Kerberos authentication.

Table 3–4 Kerberos Authentication Requirements for the DB2 Driver

Component	Requirements
Database server	The database server must be running one of the following database versions: ■ DB2 v8.1 or higher for Linux/UNIX/Windows ■ DB2 v7.x or higher for z/OS
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 one of the following methods: ■ Windows Active Directory on one of the following operating systems: Windows Server 2003 Windows 2000 Server Service Pack 3 or higher ■ MIT Kerberos 1.4.2 or higher
Client	J2SE 1.4.2 or higher must be installed.

3.10.3.2 Configuring the Driver

During installation of the WebLogic Server JDBC drivers, the following files required for Kerberos authentication are installed in the *WL_HOME*`\server\lib` folder, where *WL_HOME* is the directory in which you installed WebLogic Server:

- krb5.conf is a Kerberos configuration file containing values for the Kerberos realm and the KDC name for that realm. WebLogic Server 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 Section 3.10.1, "Using the AuthenticationMethod Property" 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:[OWLS][DB2 JDBC Driver]Could not establish a connection using
integrated security: No valid credentials provided
```

The `krb5.conf` file installed with the WebLogic JDBC drivers 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 Section 2.8.4, "Permissions for Kerberos Authentication" for an example.

3.10.4 Specifying User Credentials for Kerberos Authentication

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.

There may be times when you want the driver to use another set of user credentials. For example, 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:weblogic: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
```

3.10.5 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.

3.10.6 Configuring Client Authentication

Set the `AuthenticationMethod` property to client. See Section 3.10.1, "Using the AuthenticationMethod Property" for more information about setting a value for this property.

3.11 Data Encryption

The DB2 driver now supports SSL encryption for DB2 V5R3 and higher for iSeries. SSL secures the integrity of your data by encrypting information and providing authentication. The DB2 driver supports both SSL server authentication and SSL client authentication.

See Section 2.7.3, "SSL Encryption" for more information.

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

3.11.1 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 `KeyStorePassword` properties or their corresponding Java system properties (`javax.net.ssl.keyStore` and `javax.net.ssl.keyStorePassword`, respectively).

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

3.12 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 3–5 must exist in the specified schema. The views that are required depend on your DB2 database.

Table 3–5 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

Table 3–5 (Cont.) Catalog Tables for DB2

Database	Catalog Tables
DB2 for iSeries	QSYS2.SYSCST
	QSYS2.SYSKEYCST
	QSYS2.SYSPROCS
	QSYS2.SYSPARMS
	QSYS2.SYSTABLES
	QSYS2.SYSSYNONYM
	QSYS2.SYSCOLUMNS
	QSYS2.SQLTABLEPRIVILEGES
	QSYS2.SYSKEYS
	QSYS2.SYSINDEXES
	QSYS2.SYSREFCSTS

3.13 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.

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:

`CURRENT_SCHEMA`

Specifies the name of the current schema. The value must be a valid DB2 schema name.

If the `setCurrentUser()` method is called and this option is not specified or the value is set to `#USER#`, the schema is switched to the schema of the current user. If the `setCurrentUser()` method is called and this option is specified as an empty string, only the user is switched; the schema is not switched.

`CURRENT_PATH`

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 `setCurrentUser()` method is called and this option is not specified or the value is set to `#USER#`, the path is switched to the path of the current user. If the `setCurrentUser()` method is called and this option is specified as an empty string, only the user is switched; the path is not switched.

3.14 SQL Escape Sequences

See Appendix C, "SQL Escape Sequences for JDBC" for information about SQL escape sequences supported by the DB2 driver.

3.15 Isolation Levels

The DB2 driver supports the isolation levels listed in Table 3–6. JDBC isolation levels are mapped to the appropriate DB2 transaction isolation levels as shown. The default isolation level is `Read Committed`.

Table 3–6 Supported Isolation Levels

JDBC Isolation Level	DB2 Isolation Level
None	No Commit[1]
Read Committed	Cursor Stability
Read Uncommitted	Uncommitted Read
Repeatable Read	Read Stability
Serializable	Repeatable Read

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

3.16 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.

3.17 JTA Support

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

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

3.18 Large Object (LOB) Support

Retrieving and updating Blobs 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

Retrieving and updating Clobs is supported by the DB2 driver with all supported DB2 databases. The DB2 driver supports Clobs up to a maximum of 2 GB with the following DB2 databases:

- DB2 v8.x and higher for Linux/UNIX/Windows

- DB2 for z/OS
- DB2 V5R2 and higher for iSeries

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

Retrieving and updating DBClobs 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

3.19 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.

Note: When you create a data source in the Administration Console, the Administration Console sets the BatchPeformanceWorkaround connection property to true by default.

For data sources used as a JMS JDBC store that use the WebLogic Type 4 JDBC driver for DB2, the BatchPerformanceWorkaround property must be set to true.

3.20 Parameter Metadata Support

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

3.20.1 Insert and Update 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 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 <>.

3.20.2 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 = ?"
```

3.20.3 Stored Procedures

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

3.21 ResultSet Metadata Support

If your application requires table name information, the DB2 driver can return table name information in ResultSet metadata for Select statements. By setting 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.

3.22 Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

J2SE 1.4 or higher is required to use rowsets with the driver.

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

3.23 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 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.

3.24 Database Connection Property

The new Database connection property can be used as a synonym of the `DatabaseName` connection property.

If both the Database and DatabaseName connection properties are specified in a connection URL, the last of either 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:weblogic:db2://server1:50000;DatabaseName=jdbc;Database=acct;
User=test;Password=secret
```

3.25 DatabaseName Connection Property

The LocationName connection property is only supported when connecting to DB2 for z/OS or iSeries to specify the name of the DB2 location. Now, your application can use the DatabaseName connection property when you are connecting to DB2 for Linux/UNIX/Windows, z/OS, or iSeries.

When connecting to DB2 for Linux/UNIX/Windows, the DatabaseName connection property specifies the name of the database. When connecting to DB2 for z/OS or iSeries, the DatabaseName connection property specifies the name of the DB2 location.

3.26 New Data Types

The DB2 driver now supports:

- New data types for storing graphic data on all DB2 database versions
- New data types for DB2 v9.1 for z/OS, including the XML data type, which previously was supported only for DB2 V9.1 for Linux/UNIX/Windows

Table 3–7 and Table 3–8 list these data types and describe how they are mapped to JDBC data types.

Table 3–7 DB2 Graphic Data Types

DB2 Data Type	JDBC Data Type
Graphic	CHAR
Long Vargraphic	LONGVARCHAR
Vargraphic	VARCHAR

Table 3–8 New DB2 Data Types Supported for DB2 v9.1 for z/OS

DB2 Data Type	JDBC Data Type
Bigint	BIGINT
Binary	BINARY

Table 3–8 (Cont.) New DB2 Data Types Supported for DB2 v9.1 for z/OS

DB2 Data Type	JDBC Data Type
Decfloat	DECIMAL
Varbinary	VARBINARY
XML	CLOB

See Appendix B, "GetTypeInfo" for a description of the data types returned by the getTypeInfo() method.

For more information about using the XML data type, see Section 3.9, "Returning and Inserting/Updating XML Data."

For information about other data types supported by the DB2 driver, see Section 3.8, "Data Types."

3.27 SQL Procedures for z/OS

SQL Procedures now are supported for DB2 v9.1 for z/OS.

3.28 IPv6 Support

The DB2 driver now supports IPv6 for DB2 v9.1 for z/OS.

For more information about IPv6, see Section 2.6, "Using IP Addresses."

3.29 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.

4 The Informix Driver

The following sections describe how to configure and use the WebLogic Type 4 JDBC Informix driver:

- Section 4.1, "Informix Driver Classes"
- Section 4.2, "Informix URL"
- Section 4.3, "Informix Connection Properties"
- Section 4.4, "Performance Considerations"
- Section 4.5, "Data Types"
- Section 4.6, "Client Information for Connections"
- Section 4.7, "SQL Escape Sequences"
- Section 4.8, "Isolation Levels"
- Section 4.9, "Using Scrollable Cursors"
- Section 4.10, "Parameter Metadata Support"
- Section 4.11, "ResultSet MetaData Support"
- Section 4.12, "Rowset Support"
- Section 4.13, "Blob and Clob Searches"
- Section 4.14, "Auto-Generated Keys Support"
- Section 4.15, "Configuring Failover"

4.1 Informix Driver Classes

The driver classes for the WebLogic Type 4 JDBC Informix driver are:

```
XA: weblogic.jdbcx.informix.InformixDataSource
Non-XA: weblogic.jdbc.informix.InformixDriver
```

Use these driver classes when configuring a JDBC data source in your WebLogic Server domain.

4.2 Informix URL

To connect to an Informix database, use the following URL format:

```
jdbc:weblogic: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 Section 2.6, "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 Section 4.3, "Informix Connection Properties."

For example:

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

4.3 Informix Connection Properties

Table 4–1 lists the JDBC connection properties supported by the Informix driver, and describes each property. You can use these connection properties in a JDBC data source configuration in your WebLogic Server domain. To specify a property, use the following form in the JDBC data source configuration: *property=value*.

Note: 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.

Table 4–1 Informix Connection String Properties

Property	Description
AccountingInfo	Accounting information to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes Data type: string Valid Values: *string* where *string* is the accounting information. The default value is an empty string.

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description

AlternateServers	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.
	Data type: String
	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:weblogic:informix://server1:2003;InformixServer=TestServer;DatabaseName=Test;AlternateServers=(server2:2003;InformixServer=TestServer2,server3:2003;InformixServer=TestServer3)`
	Default: None

ApplicationName	The name of the application to be stored in the database. Thisvalue is stored locally and is used for database administration/value is stored locally and is used for database administration/monitoring purposes. Data Type: String Valid Values: *`string`* where *`string`* is the name of the application. The default value is empty string.
ClientHostName	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. Data Type: String Valid Values: *`string`* where *`string`* is the host name of the client machine. The default value is empty string.
ClientUser	The user ID to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes. Data Type: String Valid Values: *`string`* where *`string`* is a valid user ID. The default value is empty string.

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description
CodePageOverride	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. For example: CP950. Data Type: String Valid Values: *`string`* where *`string`* is the name of a valid code page that is supported by your JVM. The default value is None.
ConnectionRetryCount	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. 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. Data Type: Int Valid values: 0 | *x* where x is a positive integer. The default is 5. 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. The `ConnectionRetryDelay` property specifies the wait interval, in seconds, to occur between retry attempts. For 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.
ConnectionRetryDelay	The number of seconds the driver waits between connection retry attempts when `ConnectionRetryCount` is set to a positive integer. For example: If `ConnectionRetryCount` is set to 2, this property is set to 3, and alternate servers are specified using the `AlternateServers` property, the driver retries the list of servers (primary and alternate) twice after the initial retry attempt. The driver waits 3 seconds between retry attempts. Data Type: Int Valid values: 0 | *x* where x is a positive integer. The default is 1.

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description
ConvertNull	Controls how data conversions are handled for null values. Data Type: Int Valid values: 0 | 1. The default is 1. 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. 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.
Database	An alias for the `DatabaseName` property.
DatabaseName	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. Data Type: String Valid values: *`string`* where *`string`* is the name of a Informix database. 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:informix://server1:2003; InformixServer=ol_test;DatabaseName=jdbc;Database=acct; User=test;Password=secret`

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description
DBDate	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 Data Type: String Valid values are: DMY2, DMY4, MDY2, MDY4, Y4DM, Y4MD, Y2DM, and Y4MD 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 the hyphen (-), a period (.), and a forward slash (/). If a separator is not specified, a forward slash (/) is used to separate the fields. For 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. 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.

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description
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. This property is ignored if `FailoverMode=connect`. Data Type: String 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 that are restored during the failover process match the original rows. This value is applicable only when `FailoverMode=select`. Default is `nonAtomic`
FailoverMode	Specifies the type of failover method the driver uses. Data Type: String 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. Note the following: ■ 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 is `connect`.

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description
FailoverPreconnect	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`. Data Type: boolean 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. The `AlternateServers` property specifies one or multiple alternate servers for failover. Default is `false`.
FetchBufferSize	Specifies the size (in bytes) of the fetch buffer that the driver uses when retrieving data from the database. Valid values are any positive integer from 1 to 32767. 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 your application returns when executing Select statements and B is the number of row columns typically returned when executing Select statements. See Section 4.4, "Performance Considerations" for information about configuring this property for optimal performance. The default is 32767
ImportStatementPool	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. Data Type: String 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. The default is empty string.

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description
InformixServer (REQUIRED)	The name of the Informix database server to which you want to connect. Data Type: String Valid Values: `string` where `string` is the name of the Informix database server.
InitializationString	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. For example: `InitializationString=`*`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. For example: `jdbc:weblogic:informix://server1:2003; InformixServer=TestServer;DatabaseName=Test; InitializationString=(`*`command1`*`;`*`command2`*`)` 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. Data Type: String Default: None
InsensitiveResultSetBufferSize	-1, zero (0), or x. Determines the amount of memory used by the driver to cache insensitive result set data. If set to -1, the driver caches all insensitive result set data in memory. If the size of the result set exceeds available memory, an `OutOfMemoryException` is generated. Because the need to write result set data to disk is eliminated, the driver processes the data more efficiently. If set to 0, the driver caches all 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 all insensitive result set data in memory, using 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 the buffer size, 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 more efficient memory use. Data Type: int The default is 2048 (KB)

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description
JavaDoubleToString	True or false. Determines whether the driver uses its internal conversion algorithm or the JVM conversion algorithm when converting double or float values to string values. If set to true, the driver uses the JVM algorithm when converting double or float values to string values. If set to false (the default), the driver uses its internal algorithm when converting double or float values to string values. Setting the property to false improves performance; however, slight rounding differences can occur when compared to the same conversion using the JVM algorithm. These differences are within the allowable error of the double and float data types. The default is false.
JavaDoubleToString	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. Data Type: boolean Valid Values: `true` | `false` If `true`, the driver uses the JVM algorithm when converting a double or float value to a string value. If your application a double or float value to a string value. If your application sacrifice performance, set this value to true to use the JVM conversion algorithm. If `false`, the driver uses its own internal algorithm when converting a double or float value to a string value. This value improves performance, but slight rounding differences within the allowable error of the double and float data types can occur when compared to the same conversion using the JVM algorithm. The default value is `false`.
JDBCBehavior	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. Data Type: int Valid Values: 0 | 1 If 0, the driver describes the data types as JDBC 4.0 data types when using Java SE 6. If 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. The default value is 1.

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description
LoadBalancing	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. Data Type: boolean Valid Values: `true` | `false` If `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 `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). The default value is `false`.
LoginTimeout	The amount of time, in seconds, the driver waits for a connection to be established before returning control to the application and throwing a timeout exception. Data Type: int Valid Values: 0 | *x* where *x* is a positive integer. If 0, the driver does not time out a connection request. The default value is 0. If *x*, the driver waits for the specified number of seconds before returning control to the application and throwing a timeout exception.

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description
MaxPooledStatements	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. Data Type: int Valid Values: 0 | *x* where *x* is a positive integer. If set to 0, the driver's internal prepared statement pooling is not enabled. The default value is 0. If set to *x*, the driver enables the 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`, `CallableStatement`s 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.
MaxStatements	An alias for the `MaxPooledStatements` property.
Password (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. Data Type: String Valid Values: *string* where *string* is a valid password. The password is case-sensitive. The default value is null (no password).
PortNumber (REQUIRED)	The TCP port on which the database server listens for connections. The default varies depending on operating system. This property is supported only for data source connections.

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description
ProgramID	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. Data Type: String Valid Values: DDJ*VVRRM* where: ■ 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). ■ Default is empty string. Example: DDJ04100
QueryTimeout	Positive integer, -1, or zero (0). Sets the default query timeout (in seconds) for all statements created by a connection. If set to a positive integer, 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. 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), the default query timeout is infinite (the query does not time out).
ResultSetMetaDataOptions	Zero (0) or 1. The Informix driver can return table name information in the `ResultSet` metadata for `Select` statements if your application requires that information. If set to 0 (the default) 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. In this case, 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 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. See Section 4.4, "Performance Considerations" for information about configuring this property for optimal performance. The default is 0.

Table 4–1 (Cont.) Informix Connection String Properties

Property	Description
ServerName (REQUIRED)	Specifies either the IP address in IPv4 or IPv6, or the server name (if your network supports named servers) of the primary database server. For example, 122.23.15.12 or InformixServer. This property is supported only for data source connections. Data Type: String
SpyAttributes	Enables Spy to log detailed information about calls issued by the driver on behalf of the application. Data Type: String Default is not enabled. See "Tracking JDBC Calls with WebLogic JDBC Spy" for more details.
UseDelimitedIdentifier	Controls how the Informix server interprets double quote (") characters in SQL statements. Data Type: boolean 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. **Note:** If the `DELIMIDENT` environment variable is set on the server, the driver cannot change the setting. In this case, the `UseDelimitedIdentifier` connection option is ignored. The default is `true`.
User (REQUIRED)	The case-insensitive default user name used to connect to the Informix database. A user name is required only if security is enabled on your database. If so, contact your system administrator to obtain your user name. Data Type: String Default value is None.

4.3.1 Informix Limitation for Prepared Statements

If anything causes a change to a database table or procedure, such as adding an index, or recompiling the procedure, all existing JDBC PreparedStatements that access it must be re-prepared before they can be used again. This is a limitation of the Informix database management system. WebLogic Server caches, retains, and reuses application PreparedStatements along with pooled connections, so if your application uses prepared statements that access tables or procedures that are dropped and recreated or for which the definition is changed, re-execution of a cached prepared statement will fail once. WebLogic Server will then remove the defunct prepared statement from the cache and replace it when the application asks for the statement again.

To avoid any PreparedStatement failure due to table or procedure changes in the DBMS while WebLogic Server is running, set the Statement Cache Size to 0. WebLogic

will make a new PreparedStatement for each request. However, with the statement cache disabled, you will lose the performance benefit of statement caching.

For information about setting the Statement Cache Size, see "Increasing Performance with the Statement Cache" in *Configuring and Managing JDBC for Oracle WebLogic Server*.

4.4 Performance Considerations

Setting the following connection properties for the Informix driver as described in the following list can improve performance for your applications:

- Section 4.4.1, "FetchBufferSize"
- Section 4.4.2, "InsensitiveResultSetBufferSize"
- "MaxPooledStatements"
- Section 4.4.4, "ResultSetMetaDataOptions"

4.4.1 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. To determine the optimal value, use the formula `X = A * B * 50`, where A is the number of rows your application returns when executing `Select` statements and B is the number of row columns typically returned when executing `Select` statements.

4.4.2 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.

4.4.3 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.

4.4.4 ResultSetMetaDataOptions

By default, the Informix driver skips the additional processing required to return the correct table name for each column in the result set when the `ResultSetMetaData.getTableName()` method is called. Because of this, the `getTableName()` method may return an empty string for each column in the result

set. If you know that your application does not require table name information, this setting provides the best performance.

4.5 Data Types

Table 4–2 lists the data types supported by the Informix driver and how they are mapped to the JDBC data types.

Table 4–2 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
LVARCHAR	VARCHAR
MONEY	DECIMAL
NCHAR	CHAR 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 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 B, "GetTypeInfo" for more information about data types.

4.6 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 In.

4.7 SQL Escape Sequences

See Appendix C, "SQL Escape Sequences for JDBC" for information about the SQL escape sequences supported by the Informix driver.

4.8 Isolation Levels

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

4.9 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.

4.10 Parameter Metadata Support

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

4.10.1 Insert and Update Statements

The Informix driver supports returning parameter metadata for `Insert` and `Update` statements.

4.10.2 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.

4.10.3 Stored Procedures

The Informix driver does not support returning parameter metadata for stored procedure arguments.

4.11 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.

4.12 Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

J2SE 1.4 or higher is required to use rowsets with the driver.

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

4.13 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.

4.14 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.

4.15 Configuring Failover

Use the following procedure to configure 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 rss

2. Choose a failover method by setting the `FailoverMode` connection property. The default method is connection failover (`FailoverMode=connect`).
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 •gSpecifying Connection Retry•h 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`).

4.15.1 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=TestServe
r; DatabaseName=TestServer;User=test;Password=secret;
AlternateServers=(server2:2003;InformixServer=TestServer2,server
3:2003)
```

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,se
rver3:2003)
```

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

Example 4–1

```
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:

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=TestServe
r; DatabaseName=TestServer;User=test;Password=secret;
AlternateServers=(server2:2003;InformixServer=TestServer2;
DatabaseName=TestServer,server3:2003)
```

4.15.2 Specify 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=TestServe
r; 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.

4.15.3 Failover Properties

The following table summarizes the connection properties that control how failover works with the Informix driver.

Table 4–3 Summary: Failover Properties for the Informix Driver

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 and InformixServer) 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 ConnectionRetryCount 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 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).
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.

5 The Sybase Driver

The following sections describe how to configure and use the WebLogic Type 4 JDBC Sybase driver:

- Section 5.1, "Driver Classes"
- Section 5.2, "Sybase URL"
- Section 5.3, "J2EE Connector Architecture Resource Adapter Class"
- Section 5.4, "Sybase Connection Properties"
- Section 5.5, "Performance Considerations"
- Section 5.6, "Data Types"
- Section 5.7, "Authentication"
- Section 5.8, "Data Encryption"
- Section 5.9, "Client Information for Connections"
- Section 5.10, "SQL Escape Sequences"
- Section 5.11, "Isolation Levels"
- Section 5.12, "Using Scrollable Cursors"
- Section 5.13, "Large Object (LOB) Support"
- Section 5.14, "Batch Inserts and Updates"
- Section 5.15, "Parameter Metadata Support"
- Section 5.16, "ResultSet MetaData Support"
- Section 5.17, "Rowset Support"
- Section 5.18, "Auto-Generated Keys Support"
- Section 5.19, "NULL Values"
- Section 5.20, "Sybase JTA Support"
- Section 5.21, "Configuring Failover"
- Section 5.22, "Bulk Load"

5.1 Driver Classes

The driver class for the WebLogic Type 4 JDBC Sybase driver is:

- XA: `weblogic.jdbcx.sybase.SybaseDataSource`

- Non-XA: `weblogic.jdbc.sybase.SybaseDriver`

Use these driver classes when configuring a JDBC data source in your WebLogic Server domain.

5.2 Sybase URL

The connection URL format for the Sybase driver is:

```
jdbc:weblogic: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 Section 2.6, "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 Section 5.4, "Sybase Connection Properties."

For example:

```
jdbc:weblogic:sybase://server2:5000;User=test;Password=secre
```

5.3 J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the Informix resource adapter is:

```
com.weblogic.resource.spi.InformixManagedConnectionFactory
```

5.4 Sybase Connection Properties

Table 5–1 lists the JDBC connection properties supported by the Sybase driver, and describes each property. You can use these connection properties in a JDBC data source configuration in your WebLogic Server domain. To specify a property, use the following form in the JDBC data source configuration: *property=value*.

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

Table 5–1 Sybase Connection Properties

Property	Description
AccountingInfo	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 for database administration/monitoring purposes Data Type: String Valid Values: *`string`* where *`string`* is the accounting information. The default value is an empty string.

AlternateServers	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.
	
	Data type: String
	
	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 `DatabaseName` property.
	
	`jdbc:weblogic:sybase://server1:50000;DatabaseName=TEST;User=test;Password=secret;AlternateServers=(server2:50000;DatabaseName=TEST2,server3:50000;DatabaseName=TEST3)`
	
	Default: None

ApplicationName	The name of the application to be stored in the database. This value sets the `clientapplname` and `program_name` values in the sysprocesses table. This value is used for database administration/monitoring purposes. Valid Values: *`string`* where *`string`* is the name of the application. Your database may impose character length restrictions on the value. If the value exceeds a restriction, the driver truncates it. Data Type: String Default is empty string.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
AuthenticationMethod	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`* Data Type: string If `kerberos`, the driver uses Kerberos authentication. The driver ignores any user ID or password specified. If you set this value, you also must set the `ServicePrincipalName` property. If `userIdPassword` (the default), the driver uses user ID/password authentication. If a user ID and password is not specified, the driver throws an exception. The `User` property provides the user ID. The `Password` property provides the password. See Section 5.7, "Authentication" for more information about using authentication with the Sybase driver. The default is *`userIdPassword`*.
BatchPerformanceWorkaround	Determines the method used to execute batch operations. Valid values: `true` | `false` Data type: boolean If `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 `false` (the default), the driver uses the JDBC 3.0-compliant batch mechanism. The default is false. See Section 5.14, "Batch Inserts and Updates."
BulkLoadBatchSize	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. Valid Values: *x* where *x* is a positive integer. The default is 2048. Data Type: long

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
ClientHostName	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. 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. Default is empty string. Data Type: String
ClientUser	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. 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. Default is empty string. Data Type: String
CodePageOverride	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 retrieved from or written to the database is converted using the specified code page. The value must be a string containing the name of a valid code page supported by your JVM, for example, `CodePageOverride=CP950`. 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.
ConnectionRetryCount	The number of times the driver retries connections to a database server until a successful connection is established. Valid values: 0 | *x* where *x* is any positive integer. Data type: int If 0, the driver does not try to reconnect after the initial unsuccessful attempt. If *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. The `ConnectionRetryDelay` property specifies the wait interval, in seconds, used between attempts. The default is 5.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
ConnectionRetryDelay	The number of seconds the driver waits before retrying connections to a database server when `ConnectionRetryCount` is set to a positive integer. Valid values: 0 | *x* where *x* is the amout of time, in seconds. Data type: int If 0, the driver does not delay between retries. If *x*, the driver waits between connection retry attempts the specified number of seconds. The default is 1.
ConvertNull	Controls how data conversions are handled for null values. Valid values: 0 | 1 Data type: int If 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. If 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. The default is 1.
Database	Alias for `DatabaseName`.
DatabaseName	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=secre t`

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
EnableBulkLoad	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 `true`, the driver uses the native bulk load protocols for batch inserts. Any value set for `BatchPerformanceWorkaround` is ignored. If `false`, the driver uses the batch mechanism for batch inserts. Data Type: boolean Default is `false`.
EnableCancelTimeout	Determines whether a cancel request 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` Data Type: boolean If `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 `false` (the default), the cancel request does not time out.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
EncryptionMethod	Determines whether SSL encryption is used to encrypt and decrypt data transmitted over the network between the driver and database server. Valid values: `noEncryption` | `SSL` Data Type: String If `noEncryption`, data is not encrypted or decrypted. **Note:** Connection hangs can occur if the driver attempts to connect to a database server that requires SSL. You may want to set a login timeout using the `LoginTimeout` property to avoid problems when connecting to a server that requires SSL. If `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` ■ `TrustStore` ■ `TrustStorePassword` ■ `ValidateServerCertificate` The default is `noEncryption`.
ErrorBehavior	Determines how the driver handles errors returned from stored procedures. Valid values: `exception` | `warning` | `raiseerrorwarning` Data type: String If `exception`, the driver throws an exception when it encounters stored procedure errors, including `RAISERROR`s. If `warning`, the driver returns stored procedure errors, including `RAISERROR`s, as SQLWarnings. If `raiseerrorwarning`, the driver returns RAISERRORs as SQLWarnings and throws exceptions for other stored procedure errors. 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 The default is `exception`.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
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. This property is ignored if `FailoverMode=connect`. Valid Values: `nonAtomic` | `atomic` | `atomicWithRepositioning` | `disableIntegrityCheck` If `nonAtomic`, the driver continues with the failover process and posts any exceptions on the statement on which they occur. If `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 `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 `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`. Data Type: String Default is `nonAtomic`.
FailoverMode	Specifies the type of failover method the driver uses. Valid Values: `connect` | `extended`| `select` If `connect`, the driver provides failover protection for new connections only. If `extended`, the driver provides failover protection for new and lost connections, but not any work in progress. If `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. Data Type: String Default is `connect`.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
FailoverPreconnect	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 `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 `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. Data Type: boolean Default is `false`.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
HostNameInCertificate	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`, 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. Data Type: String Default is empty string.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
ImportStatementPool	Specifies the path and file name of the file to be used to load the contents of the statement pool. When this property is specified, statements are imported into the statement pool from the specified file. If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver throws an exception. Valid Values: *string* where *string* is the path and file name of the file to be used to load the contents of the statement pool. Data Type: String Default is empty string.
InitializationString	Specifies one or multiple SQL commands to be executed by the driver after it has established the connection to the database and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver throws an exception indicating which SQL command or commands failed. Valid Values: *command*[[;*command*]...] where *command* is a SQL command. 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 sets the handling of null values to the Sybase default and allows delimited identifiers: `jdbc:datadirect:sybase://server1:5000; InitializationString=(set ANSINULL off; set QUOTED_IDENTIFIER on);DatabaseName=test` Data Type: String

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
InsensitiveResultSetBufferSize	Determines the amount of memory used by the driver to cache insensitive result set data. Valid Values -1 | 0 | *x* Data Type: int If -1, the driver caches all insensitive result set data in memory. If the size of the result set exceeds available memory, an `OutOfMemoryException` is generated. Because the need to write result set data to disk is eliminated, the driver processes the data more efficiently. If 0, the driver caches all 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 *x*, where *x* is a positive integer that specifies the size (in KB) of the memory buffer used to cache insensitive result set data. If the size of the result set data exceeds the buffer size, 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 more efficient memory use. The default is 2048 (KB)
JavaDoubleToString	Determines whether the driver uses its internal conversion algorithm or the JVM conversion algorithm when converting double or float values to string values. Valid Values: `true` | `false` If `true`, the driver uses the JVM algorithm when converting double or float values to string values. If `false`, the driver uses its internal algorithm when converting double or float values to string values. Using this value improves performance; however, slight rounding differences can occur when compared to the same conversion using the JVM algorithm. These differences are within the allowable error of the double and float data types. The default is `false`.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
JDBCBehavior	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 Data Type: int If 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 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 is 1.
LoadBalancing	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. Data Type: boolean Valid Values: `true` | `false` If `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 `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). The default value is `false`.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
LoginTimeout	The amount of time, in seconds, the driver waits for a connection to be established before returning control to the application and throwing a timeout exception. Valid Values: 0 | *x* where *x* is a positive integer. Data Type: int If 0, the driver does not time out a connection request. The default value is 0. If *x*, the driver waits for the specified number of seconds before returning control to the application and throwing a timeout exception.
LongDataCacheSize	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 -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 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 *x*, where *x* is a positive integer, 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. See Section 5.5, "Performance Considerations" for information about configuring this property for optimal performance. The default is 2048.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
MaxPooledStatements	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. Data Type: int Valid Values: 0 | *x* where *x* is a positive integer. If set to 0, the driver's internal prepared statement pooling is not enabled. The default value is 0. If set to *x*, the driver enables the 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`, `CallableStatement`s 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.
MaxStatements	An alias for the `MaxPooledStatements` property.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
PacketSize	Determines the number of bytes for each database protocol packet 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 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. Valid values: -1 | 0 | *x* where *x* is an integer from 1 to 127. Data Type: int If -1, the driver uses the maximum packet size that is used by the database server. If 0, the driver uses the default maximum packet size used by the database server. If *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. See Section 5.5, "Performance Considerations" for information about configuring this property for optimal performance. The default is 0.
Password	The password used to connect to your Sybase database. A password is required only if security is enabled on your database. If so, contact your system administrator to get your password. Valid Values: string where string is a valid password. The password is case-sensitive. Data type: String
PortNumber (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. The default varies depending on operating system. Data type: int

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
PrepareMethod	Determines whether stored procedures are created on the server for prepared statements. Valid Values: `StoredProc` | `StoredProclfParam` | `Direct`. Data type: String If `StoredProc`, a stored procedure is created when the statement is prepared and is executed when the prepared statement is executed. If `StoredProcIfParam`, 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 `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. The default is `StoredProclfParam`. See Section 5.5, "Performance Considerations" for information about configuring this property for optimal performance.
ProgramID	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. Data Type: String Valid Values: DDJ*VVRRM* where: ■ 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). ■ Default is 0000016a. Example: DDJ04100

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
QueryTimeout	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. Data type: int If -1, the query timeout functionality is disabled. The driver silently ignores calls to the `Statement.setQueryTimeout()` method. If 0, the default query timeout is infinite (the query does not time out). If *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. The default is 0.
ResultSetMetaDataOptions	Determines whether the driver returns table name information in the `ResultSet` metadata for `Select` statements. Valid Values: 0 | 1 Data Type: int If 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. In this case, the `getTableName()` method may return an empty string for each column in the result set. If 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 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. Default is 0.

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
SelectMethod	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` Data Type: String If `direct` (the default), 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. 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 `cursor`, a server-side database cursor is requested. When returning forward-only result sets, the rows are retrieved from the server in blocks. The `setFetchSize()` method can be used to control the number of rows that are returned for each request. 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. See Section 5.5, "Performance Considerations" for information about configuring this property for optimal performance. The default is `direct`.
ServerName	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. Data type: String

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
ServicePrincipalName	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. Data type: String The value of this property can include the Kerberos realm name, but it is optional. If you do not specify the Kerberos realm name, the default Kerberos 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` or `server/sybase125ase1` See Section 5.7, "Authentication" for more information about using authentication with the Sybase driver.
SpyAttributes	Enables Spy to log detailed information about calls issued by the driver on behalf of the application. Valid Values: `(spy_attribute[;spy_attribute]...)` where spy_attribute is any valid DataDirect Spy attribute. See "Tracking JDBC Calls with WebLogic JDBC Spy" for more details. Data Type: String Default: None. 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)`

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
TransactionMode	Controls how the driver delimits the start of a local transaction. Valid Values: `implicit` | `explicit` Data Type: String If `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 `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 is `implicit`.
TrustStore	Specifies the directory of the truststore file to be used when SSL 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 specified by the javax.net.ssl.trustStore Java system property. If this property is not specified, the truststore directory is specified by the `javax.net.ssl.trustStore` Java system property. This property is ignored if `ValidateServerCertificate=false`. Valid Values: *`string`* where *`string`* is the directory of the truststore file. Data Type: String Default: None
TrustStorePassword	Specifies the password of the truststore file to be used when SSL 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 specified by the `javax.net.ssl.trustStorePassword` Java system property. If this property is not specified, the truststore password is specified by the javax.net.ssl.trustStorePassword Java system property. This property is ignored if `ValidateServerCertificate=false`. Valid Values: *`string`* where *`string`* is a valid password for the truststore file. Data Type: String Default: None

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
UseAlternateProductInfo	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` Data Type: boolean If `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 `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 is `false`.
User	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. Data Type: String Default: None

Table 5–1 (Cont.) Sybase Connection Properties

Property	Description
ValidateServerCertificate	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` Data Type: boolean If `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 `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. Default is `true`.

5.5 Performance Considerations

Setting the following connection properties for the Sybase driver as described in the following list can improve performance for your applications:

- Section 5.5.1, "BatchPerformanceWorkaround"
- Section 5.5.2, "EnableBulkLoad"
- Section 5.5.3, "EncryptionMethod"
- Section 5.5.4, "InsensitiveResultSetBufferSize"
- Section 5.5.5, "LongDataCacheSize"
- Section 5.5.6, "MaxPooledStatements"
- Section 5.5.7, "PacketSize"
- Section 5.5.8, "PrepareMethod"
- Section 5.5.9, "ResultSetMetaDataOptions"
- Section 5.5.10, "SelectMethod"

5.5.1 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.

5.5.2 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.

5.5.3 EncryptionMethod

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

5.5.4 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.

5.5.5 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 on the client. In this case, performance can be improved by increasing the amount of memory used by the driver before writing data to disk.

5.5.6 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 rss for more information about using prepared statement pooling to optimize performance.

5.5.7 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.

5.5.8 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.

5.5.9 ResultSetMetaDataOptions

By default, the Sybase driver skips the additional processing required to return the correct table name for each column in the result set when the `ResultSetMetaData.getTableName()` method is called. Because of this, the `getTableName()` method may return an empty string for each column in the result set. If you know that your application does not require table name information, this setting provides the best performance. See Section 5.16, "ResultSet MetaData Support" for more information about returning ResultSet metadata.

5.5.10 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.

5.6 Data Types

Table 5–2 lists the data types supported by the Sybase driver and how they are mapped to JDBC data types.

Table 5–2 Sybase Data Types

Sybase Data Type	JDBC Data Type
BIGINT[1]	BIGINT
BINARY	BINARY
BIT	BIT
CHAR	CHAR

Table 5–2 (Cont.) Sybase Data Types

Sybase Data Type	JDBC Data Type
DATE[2]	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[3]	TIME
TIMESTAMP	VARBINARY
TINYINT	TINYINT
UNICHAR[4]	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[5]	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[6]	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[7]	DECIMAL
UNSIGNED INT[8]	BIGINT
UNSIGNED SMALLINT[9]	INTEGER
VARBINARY	VARBINARY
VARCHAR	VARCHAR

[1] Supported only for Sybase 15.
[2] Supported only for Sybase 12.5 and higher.

[3] Supported only for Sybase 12.5 and higher
[4] Supported only for Sybase 12.5 and higher
[5] Supported only for Sybase 15.
[6] Supported only for Sybase 12.5 and higher
[7] Supported only for Sybase 15.
[8] Supported only for Sybase 15.
[9] Supported only for Sybase 15.

Note: FOR USERS OF SYBASE ADAPTIVE SERVER 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 B, "GetTypeInfo" for more information about data types.

5.7 Authentication

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See Section 2.7.1, "Authentication" 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 Section 5.7.1, "Using the AuthenticationMethod Property" for information about setting the value for this property.

5.7.1 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.

5.7.2 Configuring User ID/Password Authentication

Perform the following steps to configure the user ID and password:

1. Set the `AuthenticationMethod` property to userIdPassword. See Section 5.7.1, "Using the AuthenticationMethod Property" 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.

5.7.3 Configuring Kerberos Authentication

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

5.7.3.1 Product Requirements

Verify that your environment meets the requirements listed in Table 5–3 before you configure the driver for Kerberos authentication.

Table 5–3 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
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 one of the following methods: ▪ Windows Active Directory on one of the following operating systems: Windows Server 2003, Windows 2000 Server Service Pack 3 or higher ▪ MIT Kerberos 1.4.2 or higher
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.

5.7.3.2 Configuring the Driver

During installation of the WebLogic Server JDBC drivers, the following files required for Kerberos authentication are installed in the *`WL_HOME`*`/server/lib` folder, where *WL_HOME* is the directory in which you installed WebLogic Server

- krb5.conf is a Kerberos configuration file containing values for the Kerberos realm and the KDC name for that realm. WebLogic Server 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 Section 5.7.1, "Using the AuthenticationMethod Property" 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:[OWLS][Sybase JDBC Driver]Could not establish a connection using
   integrated security: No valid credentials provided
   ```

   The `krb5.conf` file installed with the WebLogic Type 4 JDBC drivers 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 Section 2.8.4, "Permissions for Kerberos Authentication" for an example.

5.7.4 Specifying User Credentials for Kerberos Authentication

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.

There may be times when you want the driver to use a set of user credentials other than the operating system user name and password. For example, 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:weblogic: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
```

5.7.5 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.

5.8 Data Encryption

The Sybase driver supports SSL for data encryption. SSL secures the integrity of your data by encrypting information and providing authentication. See Section 2.7.2, "Data Encryption Across the Network" 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.

5.9 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 rss.

5.10 SQL Escape Sequences

See Appendix C, "SQL Escape Sequences for JDBC" for information about the SQL escape sequences supported by the Sybase driver.

5.11 Isolation Levels

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

5.12 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.

5.13 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 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 retrieving 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.

5.14 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` connection property to `true`.

5.15 Parameter Metadata Support

The Sybase driver supports returning parameter metadata for all types of SQL statements and stored procedure arguments.

5.16 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.

5.17 Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

J2SE 1.4 or higher is required to use rowsets with the driver.

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

5.18 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.

5.19 NULL Values

When the Sybase driver establishes a connection, the driver sets the Sybase database option `ansinull` to `on`. Setting ansinull to on ensures that the driver is compliant with the ANSI SQL standard and is consistent with the behavior of other WebLogic Type 4 JDBC drivers, which simplifies developing cross-database applications.

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:weblogic:sybase://server1:5000;
InitializationString=set ANSINULL off;DatabaseName=test
```

- Explicitly execute the following statement after the connection is established:

```
SET ANSINULL OFF
```

5.20 Sybase JTA Support

Before you can use the Sybase XA driver in a global transaction, you must first set up your Sybase server to support global transactions. See "Configuring a Sybase Server for XA Support" in *Programming JTA for Oracle WebLogic Server*.

5.21 Configuring Failover

Use the following procedure to configure 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 rss
2. Choose a failover method by setting the `FailoverMode` connection property. The default method is connection failover (`FailoverMode=connect`).
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 rss.
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`).

5.21.1 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:sybase://server1:4100;DatabaseName=TEST;User=tes
t;
Password=secret;AlternateServers=(server2:4100;DatabaseName=TEST
2, 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 primary and alternate servers specified using a JDBC data source would look like this:

Example 5–1 Example JDBC Data Source Configuration

```
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 either of the following connection properties: `DatabaseName` or `InformixServer`. These connection properties are optional for each alternate server entry. For example:

  ```
  jdbc:datadirect:sybase://server1:4100;DatabaseName=TEST;User=
  test;
  Password=secret;AlternateServers=(server2:4100;DatabaseName=T
  EST2, 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=tes
t; Password=secret;AlternateServers=(server2:4100,server3:4100)
```

5.21.2 Specify 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:sybase://server1:4100;DatabaseName=TEST;User=tes
t;
Password=secret;AlternateServers=(server2:4100;DatabaseName=TEST
2, 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.

5.21.3 Failover Properties

The following table summarizes the connection properties that control how failover works with the Informix driver.

Table 5–4 Summary: Failover Properties for the Informix Driver

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).
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).

Table 5–4 (Cont.) Summary: Failover Properties for the Informix Driver

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.

5.22 Bulk Load

The driver supports WebLogic 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 rss.

6

The MS SQL Server Driver

The following sections describe how to configure and use the WebLogic Type 4 JDBC SQL Server driver:

- Section 6.1, "Driver Class"
- Section 6.2, "Microsoft SQL Server URL"
- Section 6.3, "Connecting to Named Instances"
- Section 6.4, "SQL Server Connection Properties"
- Section 6.5, "Performance Considerations"
- Section 6.6, "Data Types"
- Section 6.7, "Returning and Inserting/Updating XML Data"
- Section 6.8, "Authentication"
- Section 6.9, "Data Encryption"
- Section 6.10, "DML with Results (Microsoft SQL Server 2005 and Higher)"
- Section 6.11, "Reauthentication"
- Section 6.12, "Client Information for Connections"
- Section 6.13, "SQL Escape Sequences"
- Section 6.14, "Isolation Levels"
- Section 6.15, "Using the Snapshot Isolation Level (Microsoft SQL Server 2005 and Higher)"
- Section 6.16, "Using Scrollable Cursors"
- Section 6.17, "Server-Side Updatable Cursors"
- Section 6.18, "Installing Stored Procedures for JTA"
- Section 6.19, "Distributed Transaction Cleanup"
- Section 6.20, "Large Object (LOB) Support"
- Section 6.21, "Batch Inserts and Updates"
- Section 6.22, "Parameter Metadata Support"
- Section 6.23, "ResultSet MetaData Support"
- Section 6.24, "Rowset Support"
- Section 6.25, "Auto-Generated Keys Support"
- Section 6.26, "Null Values"

- Section 6.27, "Configuring Failover"
- Section 6.28, "Specifying Primary and Alternate Servers"
- Section 6.29, "Specifying Connection Retry"
- Section 6.30, "Failover Properties"
- Section 6.31, "Bulk Load"

Note: The WebLogic Type 4 JDBC MS SQL Server driver (the subject of this chapter) replaces the WebLogic jDriver for Microsoft SQL Server, which is deprecated. The new driver offers JDBC 3.0 compliance, support for some JDBC 2.0 extensions, and better performance. Oracle recommends that you use the new WebLogic Type 4 JDBC MS SQL Server driver in place of the WebLogic jDriver for Microsoft SQL Server.

6.1 Driver Class

The driver classes for the WebLogic Type 4 JDBC MS SQL Server driver are:

```
XA: weblogic.jdbcx.sqlserver.SQLServerDataSource
Non-XA: weblogic.jdbc.sqlserver.SQLServerDriver
```

6.2 Microsoft SQL Server URL

To connect to a Microsoft SQL Server database, use the following URL format:

```
jdbc:weblogic: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 Section 2.6, "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 Section 6.4, "SQL Server Connection Properties."

For example:

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

See Section 6.3, "Connecting to Named Instances" for instructions on connecting to named instances.

6.3 Connecting to Named Instances

Microsoft SQL Server and Microsoft SQL Server 2005 support multiple instances of a 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:weblogic:sqlserver://server_name\\instance_name
```

> **Note:** The first back slash 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:weblogic:sqlserver://server1\\instance1;User=test;Pasword=secret
```

6.4 SQL Server Connection Properties

Table 6–1 lists the JDBC connection properties supported by the SQL Server driver, and describes each property. You can use these connection properties in a JDBC data source configuration in your WebLogic Server domain. To specify a property, use the following form in the JDBC data source configuration: *property=value*.

> **Note:** All connection string property names are case-insensitive. For example, Password is the same as password.

Table 6–1 SQL Server Connection Properties

Property	Description
AccountingInfo	Accounting information to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes Data type: string Valid Values: *string* where *string* is the accounting information. The default value is an empty string.

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description

AlternateServers	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.
	Data type: String
	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 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, `DatabaseName`.
	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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
AlwaysReportTriggerResults	Determines how the driver reports results 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, this includes triggers fired by Data Definition Language (DDL) events. Valid Values: `true` or `false` If `true`, the driver returns all results, including results generated by triggers. Multiple trigger results are returned one at a time. Use the `Statement.getMoreResults()` method to retrieve individual trigger results. Warnings and errors are reported in the results as they are encountered. If `false` (the default): ■ For Microsoft SQL Server 2005, 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. In this case, the only result that is returned is the update count generated by the statement that was executed (if errors do not occur). Although trigger results are ignored, any errors generated by the trigger are reported. Any warnings generated by the trigger are enqueued. If errors are reported, the update count is not reported. The default is `false`.
ApplicationName	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. Default: empty string Data Type: String Alias: `ProgramName property`

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
AuthenticationMethod	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: `auto` | `kerberos` | `ntlm` | `userIdPassword` If `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 Section 6.8.1, "Using the AuthenticationMethod Property" for more information about using the default value. If `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. If `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. If `userIdPassword`, the driver uses SQL Server authentication when establishing a connection. If a user ID is not specified, the driver throws an exception. The `User` property provides the user ID. The `Password` property provides the password. **Note:** The values type4, type2, and none are deprecated, but are recognized for backward compatibility. We recommend that you use the kerberos, ntlm, and userIdPassword value, respectively, instead. See Section 6.8, "Authentication" for more information about using authentication with the SQL Server driver. Default: `userIdPassword` Data Type: String

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
BulkLoadBatchSize	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 •gDDBulkLoad Interface•h on page.603 for a description of the method. Valid Values: *x* where *x* is a positive integer. Default: 2048
BulkLoadOptions	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: ■ 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 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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
ClientHostName	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: *`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. Default: empty string Data Typ:e String Alias: `WSID` property
CodePageOverride	Specifies the code page the driver uses when converting character data. The specified code page overrides the default database code page. All character data retrieved from or written to the database is converted using the specified code page. The value must be a string containing the name of a valid code page supported by your JVM, for example, `CodePageOverride=CP950`. 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. If a value is set for the `CodePageOverride` property and the `SendStringParametersAsUnicode` property is set to true, the driver ignores the property and generates a warning. The driver always sends parameters using the code page specified by `CodePageOverride` if this property is specified.

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
ConnectionRetryCount	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 0, the driver does not try to reconnect after the initial unsuccessful attempt. If *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
ConnectionRetryDelay	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 0, the driver does not delay between retries. If *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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
ConvertNull	Controls how data conversions are handled for null values. Valid Values: 0 | 1 If 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. If 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. Default: 1 Data Type: int
Database	An alias for the DatabaseName property.
DatabaseName	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 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:weblogic:sqlserver://server1:1433;DatabaseName=jdbc;Database=acct;User=test;Password=secret`

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
DescribeParameters	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` | `describeIfString` The `SendStringParametersAsUnicode` property controls whether the driver sends String parameter values to the server as Unicode (for example, nvarchar) or non-Unicode (for example, varchar). 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. If `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 `SendStringParametersAsUnicode` property. If `describeIfString`, 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 setString(), setCharacterStream(), and setAsciiStream()). 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, nvarchar). 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 `SendStringParametersAsUnicode` property. Default: `noDescribe` Data Type: String

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
EnableBulkLoad	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 `true`, the driver uses the native bulk load protocols for batch inserts. If `false`, the driver uses the batch mechanism for batch inserts. Default: `false` Data Type: boolean
EnableCancelTimeout	Determines whether a cancel request 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 `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 sets `Statement.setQueryTimeout(5)` 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, for example, because the server is down, the driver throws an exception indicating that the cancel request was timed out and the connection is no longer valid. If `false`, the cancel request does not time out. Default: `false` Data Type: boolean

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description

EncryptionMethod	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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
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. 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
FailoverMode	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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
FailoverPreconnect	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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
HostNameInCertificate	Specifies a host name for certificate validation when SSL encryption is enabled (`EncryptionMethod=SSL`) and validation is enabled (`ValidateServerCertificate=true`). This property is optional and provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested. Valid Values: *`host_name`* | *`#SERVERNAME#`* where *`host_name`* is a valid host name. If a *`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, 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, the driver compares the host name to the CN parts of the certificate's `Subject` name. If the values do not match, the connection fails and the driver throws an exception. **Note:** 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. If unspecified, the driver does not validate the host name in the certificate. If SSL encryption or certificate validation is not enabled, any value specified for this property is ignored. See Section 6.9, "Data Encryption" for information about configuring for authentication. The default is an empty string.
HostProcess	An alias for the `ProgramID` property.
ImportStatementPool	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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description

InitializationString	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 The following connection URL sets the handling of null values to the Microsoft SQL Server default and allows delimited identifiers:
	
	`jdbc:datadirect:sqlserver://server1:1433;`
	
	`InitializationString=(set ANSI_NULLS off;`
	
	`set QUOTED_IDENTIFIER on);DatabaseName=test`
	
	Default: None
	
	Data Type: String

InsensitiveResultSetBufferSize	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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
JavaDoubleToString	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	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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
LoadBalancing	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
LoadLibraryPath	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 `install_dir/lib` subdirectory, where `install_dir` 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. Default: None Data Type: String
LoginTimeout	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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
LongDataCacheSize	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*, where *x* is a positive integer, 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. See Section 6.5, "Performance Considerations" for information about configuring this property for optimal performance. Default: 2048 Data Type: int
MaxPooledStatements	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 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: `MaxStatements property`
MaxStatements	An alias for the `MaxPooledStatements` property.

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
NetAddress	The Media Access Control (MAC) address of the network interface card of the application connecting to Microsoft SQL Server. This value is a string up to a maximum of 12 characters. The value of this property may be useful for database administration purposes. This value is stored in the net_address column of the: ■ sys.sysprocesses table (Microsoft SQL Server 2005) ■ master.dbo.sysprocesses table (Microsoft SQL Server 2000) The default is 000000000000.
PacketSize	Determines the number of bytes for each database protocol packet 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 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. Valid Values: `-1` | `0` | *x* where *x* is an integer from 1 to 128. If set to `-1`, the driver uses the default maximum packet size used by the database server. If set to `0` (the default), the driver uses a packet size of 64 KB. If set to *x*, an integer from 1 to 128, 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). See Section 6.5, "Performance Considerations" for information about configuring this property for optimal performance. The default is `0`.
Password	A case-insensitive password used to connect to your Microsoft SQL Server database. A password is required only if SQL Server authentication is enabled on your database. If so, 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 See Section 6.8, "Authentication" for more information about configuring authentication.

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
PortNumber	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	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: *DDJVVRRM* where: ■ *DDJ* is an identifier for a 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: `HostProcess` property
ProgramName	An alias for the `ApplicationName` property.
QueryTimeout	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 *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. 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), the default query timeout is infinite (the query does not time out). Default: 0 Data Type: int

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
ReceiveStringParameterType	Specifies how the driver describes String stored procedure output parameters to the database. Valid Values: `NVARCHAR` | `VARCHAR` | `DESCRIBE` If set to `NVARCHAR` (the default), the driver describes String stored procedure output parameters as nvarchar (4000). Use this value if all output parameters 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 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 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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
ResultSetMetaDataOptions	The SQL Server driver can return table name information in the ResultSet metadata for Select statements if your application requires that information. Valid Values: 0 | 1 If set to 0 (the default) 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. In this case, 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 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. See Section 6.23, "ResultSet MetaData Support" for more information about returning ResultSet metadata. Default: 0 Data Type: int

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
SelectMethod	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 default), 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 retrieved from the server in blocks. The `setFetchSize()` method can be used to control the number of rows that are retrieved 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. Oracle recommends 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. See Section 6.5, "Performance Considerations" for information about configuring this property for optimal performance. Default: `direct` Data Type: String

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
SendStringParametersAsUnicode	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` (the default), string parameters are sent to Microsoft SQL Server in Unicode. 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. 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. See Section 6.5, "Performance Considerations" for information about configuring this property for optimal performance. Default: `true` Data Type: boolean
ServerName (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 or named instance. For example, 122.23.15.12 or SQLServerServer. 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. This property is supported only for data source connections. See Section 6.3, "Connecting to Named Instances" for more information about connecting to named instances. Defaul:t None Data Type: String

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
SnapshotSerializable	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
SpyAttributes	Enables Spy to log detailed information about calls issued by the driver on behalf of the application. Spy is not enabled by default. Valid Values: (*`spy_attribute`*[;*`spy_attribute`*]...) where *`spy_attribute`* is any valid Spy attribute. See Appendix D, "Tracking JDBC Calls with WebLogic JDBC Spy." 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)`D efault: None Data Typ:e String

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
TransactionMode	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	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	Specifies the directory of the truststore file to be used when SSL 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 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

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
TrustStorePassword	Specifies the password of the truststore file to be used when SSL 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 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	The case-insensitive user name used to connect to your Microsoft SQL Server database. A user name is required only if SQL Server authentication is enabled on your database. If so, 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 Typ: e String
UseServerSideUpdatableCursors	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 default updatable result set functionality is used. See Section 6.17, "Server-Side Updatable Cursors" for more information about using server-side updatable cursors. See Section 6.5, "Performance Considerations" for information about configuring this property for optimal performance. Default: `false` Data Type: boolean

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
ValidateServerCertificate	Determines whether the driver validates the certificate sent by the database server when SSL encryption is enabled (`EncryptionMethod=SSL`). When using SSL server authentication, any certificate sent by the server must be issued by a trusted Certificate Authority (CA). Allowing the driver to trust any certificate 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 `false` (the default), the driver does not validate the certificate sent by the database server. The driver ignores any truststore information specified by the TrustStore and TrustStorePassword properties or Java system properties. If set to `true`, the driver validates the certificate 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. Truststore information is specified using the TrustStore and TrustStorePassword properties or by using Java system properties. See Section 6.9, "Data Encryption" for information about configuring for authentication. Default: `true` Data Type: boolean
WSID	An alias for the `ClientHostName` property.

Table 6–1 (Cont.) SQL Server Connection Properties

Property	Description
XATransactionGroup	The transaction group ID that identifies any transactions initiated by the connection. This ID can be used for distributed transaction cleanup purposes. Valid Values: `string` where `string` is a valid transaction group ID. 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. For example, if you specify `XATransactionGroup=ACCT200` and call `XAResource.recover` on the same connection, any transactions left in an unprepared state identified by the transaction group ID of ACCT200 are rolled back. See Section 6.19, "Distributed Transaction Cleanup" for more information about distributed transaction cleanup. Default: None Data Type: String
XMLDescribeType	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. See Section 6.7, "Returning and Inserting/Updating XML Data" for more information. Default: None Data Type: String

6.5 Performance Considerations

Setting the following connection properties for the SQL Server driver as described in the following list can improve performance for your applications.

- Section 6.5.1, "EnableBulkLoad"
- Section 6.5.2, "EncryptionMethod"
- Section 6.5.3, "InsensitiveResultSetBufferSize"
- Section 6.5.4, "LongDataCacheSize"
- Section 6.5.6, "PacketSize"
- Section 6.5.7, "ResultSetMetaDataOptions"
- Section 6.5.8, "SelectMethod"
- Section 6.5.9, "SendStringParametersAsUnicode"
- Section 6.5.10, "SnapshotSerializable"

- Section 6.5.11, "UseServerSideUpdatableCursors"

6.5.1 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.

6.5.2 EncryptionMethod

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

6.5.3 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.

6.5.4 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 on the client. In this case, performance can be improved by increasing the amount of memory used by the driver before writing data to disk.

6.5.5 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.

6.5.6 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.

6.5.7 ResultSetMetaDataOptions

By default, the SQL Server driver skips the additional processing required to return the correct table name for each column in the result set when the `ResultSetMetaData.getTableName()` method is called. Because of this, the `getTableName()` method may return an empty string for each column in the result set. If you know that your application does not require table name information, this setting provides the best performance.

See Section 6.23, "ResultSet MetaData Support" for more information about returning ResultSet metadata.

6.5.8 SelectMethod

In most cases, using server-side database cursors impacts performance negatively. However, if the following variables 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.

6.5.9 SendStringParametersAsUnicode

If all the data accessed by your application is stored in the database using the default database character encoding, setting `SendStringParametersAsUnicode` to false can improve performance.

6.5.10 SnapshotSerializable

You must have your Microsoft SQL Server 2005 or higher database configured for Snapshot Isolation for this connection property to work. See Section 6.15, "Using the Snapshot Isolation Level (Microsoft SQL Server 2005 and Higher)" 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.

6.5.11 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 Section 6.17, "Server-Side Updatable Cursors" for more information about using server-side updatable cursors.

6.6 Data Types

Table 6–2 lists the data types supported by the SQL Server driver and and how they are mapped to the JDBC data types.

Table 6–2 Microsoft SQL Server Data Types

Microsoft SQL Server Data Type	JDBC Data Type
bigint[1]	BIGINT
bigint identity [2]	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).
ntext	LONGVARCHAR NOTE: If `JDBCBehavior=0`, the data type depends on the JVM used by the application: `LONGNVARCHAR` (if using Java SE 6) or `LONGVARCHA`R (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).

Table 6–2 (Cont.) Microsoft SQL Server Data Types

Microsoft SQL Server Data Type	JDBC Data Type
nvarchar(max)[3]	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 [4]	VARCHAR
sysname	VARCHAR
text	LONGVARCHAR
time	TIMESTAMP
timestamp	BINARY
tinyint	TINYINT
tinyint identity	TINYINT
uniqueidentifier	CHAR
varbinary	VARBINARY
varbinary(max) [5]	LONGVARBINARY
varchar	VARCHAR
varchar(max) [6]	LONGVARCHAR
xml [7]	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).

[1] Supported only for Microsoft SQL Server 2000 and higher.
[2] Supported only for Microsoft SQL Server 2000 and higher.
[3] Supported only for Microsoft SQL Server 2005
[4] Supported only for Microsoft SQL Server 2000 and higher.
[5] Supported only for Microsoft SQL Server 2005.
[6] Supported only for Microsoft SQL Server 2005
[7] Supported only for Microsoft SQL Server 2005

See Appendix B, "GetTypeInfo" for more information about data types.

6.7 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.

6.7.1 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:weblogic:sqlserver://server1:1433;DatabaseName=jdbc;User=test;
Password=secret;XMLDescribeType=longvarbinary
```

6.7.1.1 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.

6.7.1.2 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.

6.7.2 Inserting/Updating XML Data

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

6.7.2.1 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.

6.7.2.2 Binary Data

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

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

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

6.8 Authentication

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See Section 2.7.1, "Authentication" 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 Section 6.8.1, "Using

the AuthenticationMethod Property" for information about setting the value for this property.

6.8.1 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.

6.8.2 Configuring SQL Server Authentication

1. Set the `AuthenticationMethod` property to auto or userIdPassword (the default). See Section 6.8.1, "Using the AuthenticationMethod Property" 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.

6.8.3 Configuring Kerberos Authentication

This section provides requirements and instructions for configuring Kerberos authentication for the Microsoft SQL Server driver.

6.8.3.1 Product Requirements

Verify that your environment meets the requirements listed in Table 6–3 before you configure the driver for Kerberos authentication.

Table 6–3 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: ■ Microsoft SQL Server 2008 ■ Microsoft SQL Server 2008 ■ Microsoft SQL Server 2000 ■ Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher
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: ■ Windows Server 2003 ■ Windows 2000 Server Service Pack 3 or higher
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.

6.8.3.2 Configuring the Driver

During installation of the WebLogic Server JDBC drivers, the following files required for Kerberos authentication are installed in the *WL_HOME*`/server/lib` folder, where *WL_HOME* is the directory in which you installed WebLogic Server:

- krb5.conf is a Kerberos configuration file containing values for the Kerberos realm and the KDC name for that realm. WebLogic Server 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 Section 6.8.1, "Using the AuthenticationMethod Property" 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:[OWLS][SQLServer JDBC Driver]Could not establish a connection using
integrated security: No valid credentials provided
```

The krb5.conf file installed with the WebLogic JDBC drivers 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 Section 2.8.4, "Permissions for Kerberos Authentication" 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

6.8.4 Specifying User Credentials for Kerberos Authentication

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.

There may be times when you want the driver to use a set of user credentials other than the operating system user name and password. For example, 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:weblogic: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
```

6.8.5 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.

6.8.6 Configuring NTLM Authentication

This section provides requirements and instructions for configuring NTLM authentication for the Microsoft SQL Server driver.

6.8.6.1 Product Requirements

Verify that your environment meets the requirements listed in Table 6–4 before you configure your environment for NTLM authentication.

Table 6–4 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: ■ Microsoft SQL Server 2008 ■ Microsoft SQL Server 2005 ■ Microsoft SQL Server 2000 Service Pack 3 or higher ■ Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher
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: ■ Windows Server 2003 ■ Windows 2000 Server Service Pack 3 or higher
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: ■ Windows Vista ■ Windows Server 2003 ■ Windows XP Service Pack 2 or higher ■ Windows 2000 Service Pack 4 or higher ■ Windows NT 4.0 In addition, J2SE 1.4 or higher must be installed.

6.8.6.2 Configuring the Driver

WebLogic Type 4 JDBC drivers provide the following NTLM authentication DLLs:

- DDJDBCAuth*xx*.dll (32-bit)

- DDJDBC64Auth*xx*.dll (Itanium 64-bit)
- DDJDBCx64Auth*xx*.dll (AMD64 and Intel EM64T 64-bit)

where *xx* is a two-digit number.

The DLLs are located in the *WL_HOME*`/server/lib` directory (where *WL_HOME* is the directory in which you installed WebLogic Server). 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 Section 6.8.1, "Using the AuthenticationMethod Property" 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 *WL_HOME*`/server/lib` directory to the Windows system path, where *WL_HOME* is the directory in which you installed WebLogic Server.
   - Copy the NTLM authentication DLLs from *WL_HOME*`/server/lib` to a directory that is on the Windows system path, where *WL_HOME* is the directory in which you installed WebLogic Server.
   - 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:weblogic:sqlserver://server3:1521;
     DatabaseName=test;LoadLibraryPath=C:\DataDirect\lib;User=test;Password=secr
     et
     ```

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 Section 2.8.1, "Permissions for Establishing Connections" for an example.

6.9 Data Encryption

The SQL Server driver supports SSL for data encryption. SSL secures the integrity of your data by encrypting information and providing authentication. See Section 2.7.2, "Data Encryption Across the Network" 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 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.

6.9.1 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 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 6–5 shows how the different `EncryptionMethod` property values behave with different Microsoft SQL Server configurations.

Table 6–5 EncryptionMethod Property and Microsoft SQL Server Configurations

Value	No SSL Certificate	SSL Certificate (SSL Optional)	SSL Certificate (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: 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.

6.9.2 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.

6.10 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();
}
```

6.11 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`.

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 reauthentication options supported for the SQL Server driver are:

- `CURRENT_DATABASE`: Specifies the name of the current database. The value must be a valid Microsoft SQL Server database name.If the `setCurrentUser()` 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.

6.12 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.

6.13 SQL Escape Sequences

See Appendix C, "SQL Escape Sequences for JDBC" for information about the SQL escape sequences supported by the SQL Server driver.

6.14 Isolation Levels

The SQL Server driver supports the following isolation levels for Microsoft SQL Server:

- Read Committed with Locks (supported for Microsoft SQL Server 2005 only) or Read Committed
- Read Committed with Snapshots (supported for Microsoft SQL Server 2005 only)
- Read Uncommitted
- Repeatable Read
- Serializable
- Snapshot (supported for Microsoft SQL Server 2005 only)

The default is Read Committed with Locks (Microsoft SQL Server 2005) or Read Committed.

6.15 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 description of this property in Table 6–1 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);
```

6.16 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.

6.17 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 were 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 would not be visible.

6.18 Installing Stored Procedures for JTA

To use JDBC distributed transactions through JTA, your system administrator should use the following procedure to install Microsoft SQL Server JDBC XA procedures. This procedure must be repeated for each MS SQL Server installation that will be involved in a distributed transaction.

To install stored procedures for JTA:

1. Copy the appropriate `sqljdbc.dll` and `instjdbc.sql` files from the *`WL_HOME`*`\server\lib` directory to the *`SQL_Server_Root`*`/bin` directory of the MS SQL Server database server, where *WL_HOME* is the directory in which WebLogic server is installed, typically `c:\Oracle\Middleware\wlserver_10.x`.

   Note: If you are installing stored procedures on a database server with multiple Microsoft SQL Server instances, each running SQL Server instance must be able to locate the `sqljdbc.dll` file. Therefore the `sqljdbc.dll` file needs to be anywhere on the global PATH or on the application-specific path. For the application-specific path, place the `sqljdbc.dll` file into the `<drive>:\Program Files\Microsoft SQL Server\MSSQL$<Instance 1 Name>\Binn` directory for each instance.

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, use the following syntax to run `instjdbc.sql`:

   ```
   ISQL -Usa -Psa_password -Sserver_name -ilocation\instjdbc.sql
   ```

   where:

   sa_password is the password of the system administrator.

   server_name is the name of the server on which SQL Server resides.

   location is the full path to `instjdbc.sql`. (You copied this script to the *`SQL_Server_Root`*`/bin` directory in step 1.)

   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.

6.19 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.

6.19.1 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.

6.19.2 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.

6.20 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.

6.21 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.

6.22 Parameter Metadata Support

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

6.22.1 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 <>.

6.22.2 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 = ?"
```

6.22.3 Stored Procedures

The SQL Server driver does not support returning parameter metadata for stored procedure arguments.

6.23 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.

6.24 Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

J2SE 1.4 or higher is required to use rowsets with the driver.

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

6.25 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 those values depends on whether you are using an Insert statement that contains parameters:

- When using an Insert statement that contains no parameters, the MS 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 MS SQL Server driver supports the following form of the `Connection.prepareStatement()` method to inform 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.

6.26 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:weblogic:sqlserver://server1:1433;
  InitializationString=set ANSI_NULLS off;
  DatabaseName=test
  ```

- Explicitly execute the following statement after the connection is established:

  ```
  SET ANSI_NULLS OFF
  ```

6.27 Configuring Failover

Use the following steps to configure 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.

   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`).
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.
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`).

6.28 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:weblogic:sqlserver://server1:1433;DatabaseName=TEST;User=te
st;
Password=secret;AlternateServers=(server2:1433;DatabaseName=TEST
2, 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:weblogic: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.

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:

  ```
  Password=secret;AlternateServers=(server2:1433;DatabaseName=T
  EST2, server3:1433;DatabaseName=TEST3)
  ```

  or, if connecting to named instances:

  ```
  jdbc:weblogic:sqlserver://server1\\instance1:1433;DatabaseNam
  e=TEST;
  User=test;Password=secret;AlternateServers=(server2\\instance
  2:1433;
  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;Us
  er=test;
  Password=secret;AlternateServers=(server2:1433,server3:1433)
  ```

6.29 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.

6.30 Failover Properties

The following section summarizes the connection properties that control how failover works with the SQL Server driver:

- `AlternateServers`: One or multiple alternate database servers. An IP address or server name identifying each server is required. Port number and the connection property `DatabaseName` 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 `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).
- `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.

6.31 Bulk Load

The driver supports `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.

A

JDBC Support

This appendix provides information about JDBC compatibility and developing JDBC applications using WebLogic Type 4 JDBC drivers.

- Section A.1, "JDBC Compatibility"
- Section A.2, "Supported Functionality"
  - Section A.2.1, "Array Object"
  - Section A.2.2, "Blob Object"
  - Section A.2.3, "CallableStatement Object"
  - Section A.2.4, "Clob Object"
  - Section A.2.5, "Connection Object"
  - Section A.2.6, "ConnectionEventListener Object"
  - Section A.2.7, "ConnectionPoolDataSource Object"
  - Section A.2.8, "DatabaseMetaData Object"
  - Section A.2.9, "Data Source Object"
  - Section A.2.10, "Driver Object"
  - Section A.2.11, "ParameterMetaData Object"
  - Section A.2.12, "PooledConnection Object"
  - Section A.2.13, "PreparedStatement Object"
  - Section A.2.14, "Ref Object"
  - Section A.2.15, "ResultSet Object"
  - Section A.2.16, "ResultSetMetaData Object"
  - Section A.2.17, "RowSet Object"
  - Section A.2.18, "SavePoint Object"
  - Section A.2.19, "Statement Object"
  - Section A.2.20, "StatementEventListener Object"
  - Section A.2.21, "Struct Object"
  - Section A.2.22, "XAConnection Object"
  - Section A.2.23, "XADataSource Object"
  - Section A.2.24, "XAResource Object"

A.1 JDBC Compatibility

Table A–1 shows compatibility among the JDBC specification versions, JVMs, and the WebLogic Type 4 JDBC drivers.

Table A–1 JDBC Compatibility

JDBC Version	Java 2 SDK	Drivers Compatible?
3.0	5.0	Yes
4.0	6.0	Yes

A.2 Supported Functionality

The following tables list functionality supported for each JDBC object.

A.2.1 Array Object

The following table lists functionality supported for each Array object.

Table A–2 Array Object

Array Object Methods	Version Introduced	Supported	Comments
(all)	2.0 Core	No	Array objects are not exposed or used as input.

A.2.2 Blob Object

The following table lists functionality supported for each Blob object.

Table A–3 Blob Object

Blob Object Methods	Version Introduced	Supported	Comments
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 JDBCLONGVARBINARY data type.
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 LONGVARBINARY data type.

Table A–3 (Cont.) Blob Object

Blob Object Methods	Version Introduced	Supported	Comments
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 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 LONGVARBINARY data type.
long position (byte[], long)	2.0 Core	Yes	The DB2 driver only supports with DB2v8.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 byte array 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 LONGVARBINARY data type.
OutputStream setBinaryStream (long)	3.0	Yes	The DB2 driver only supports with DB2v8.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 LONGVARBINARY data type.

Table A–3 (Cont.) Blob Object

Blob Object Methods	Version Introduced	Supported	Comments
int setBytes (long, byte[])	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 LONGVARBINARY data type.
int setBytes (long, byte[], int, int)	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 LONGVARBINARY data type.
void truncate (long)	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 LONGVARBINARY data type.

A.2.3 CallableStatement Object

The following table lists functionality supported for each CallableStatement object.

Table A–4 CallableStatement Object

CallableStatement Object Methods	Version Introduced	Supported	Comments
Array getArray (int)	2.0 Core	No	Throws "unsupported method" exception.
Array getArray (String)	3.0	No	Throws "unsupported method" exception.
Reader getCharacterStream (int)	4.0	Yes	
Reader getCharacterStream (String)	4.0	Yes	
BigDecimal getBigDecimal (int)	2.0 Core	Yes	N/A
BigDecimal getBigDecimal (int, int)	1.0	Yes	N/A

Table A–4 (Cont.) CallableStatement Object

CallableStatement Object Methods	Version Introduced	Supported	Comments
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	N/A
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	N/A
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	N/A
byte [] getBytes (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.
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.

Table A–4 (Cont.) CallableStatement Object

CallableStatement Object Methods	Version Introduced	Supported	Comments
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	N/A
Date getDate (int, Calendar)	2.0 Core	Yes	N/A
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	N/A
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	N/A
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	N/A
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	N/A
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	

Table A–4 (Cont.) CallableStatement Object

CallableStatement Object Methods	Version Introduced	Supported	Comments
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	N/A
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.
Ref getRef (String)	3.0	No	Throws "unsupported method" exception.
short getShort (int)	1.0	Yes	N/A
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	N/A
Time getTime (int, Calendar)	2.0 Core	Yes	N/A

Table A–4 (Cont.) CallableStatement Object

CallableStatement Object Methods	Version Introduced	Supported	Comments
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 the SQL Server driver only. All other drivers throw "unsupported method" exception.
Timestamp getTimestamp (int)	1.0	Yes	N/A
Timestamp getTimestamp (int, Calendar)	2.0 Core	Yes	N/A
Timestamp getTimestamp (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
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	N/A
void registerOutParameter (int, int, int)	1.0	Yes	N/A
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.

Table A–4 (Cont.) CallableStatement Object

CallableStatement Object Methods	Version Introduced	Supported	Comments
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.
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.
void setBlob (String, InputStream)	4.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.

Table A–4 (Cont.) CallableStatement Object

CallableStatement Object Methods	Version Introduced	Supported	Comments
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.
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.

Table A–4 (Cont.) CallableStatement Object

CallableStatement Object Methods	Version Introduced	Supported	Comments
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	
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.

Table A–4 (Cont.) CallableStatement Object

CallableStatement Object Methods	Version Introduced	Supported	Comments
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	
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.
void setURL (String, URL)	3.0	No	Throws "unsupported method" exception.
boolean wasNull ()	1.0	Yes	N/A

A.2.4 Clob Object

The following table lists functionality supported for each Clob object.

Table A–5 Clob Object

Clob Object Methods	Version Introduced	Supported	Comments
void free ()	4.0	Yes	

Table A–5 (Cont.) Clob Object

Clob Object Methods	Version Introduced	Supported	Comments
InputStream getAsciiStream ()	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the LONGVARCHAR data type.
Reader getCharacterStream ()	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the 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 LONGVARCHAR data type.
long length ()	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the 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 LONGVARCHAR data type.
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 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 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 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 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 LONGVARCHAR data type.

Table A–5 (Cont.) Clob Object

Clob Object Methods	Version Introduced	Supported	Comments
void truncate (long)	3.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the LONGVARCHAR data type.

A.2.5 Connection Object

The following table lists functionality supported for each Connection object.

Table A–6 Connection Object

Connection Object Methods	Version Introduced	Supported	Comments
void clearWarnings ()	1.0	Yes	N/A
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	N/A
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	N/A
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	N/A
String getCatalog ()	1.0	Yes	Supported for all drivers.
String getClientInfo ()	4.0	Yes	N/A
String getClientInfo (String)	4.0	Yes	N/A
int getHoldability ()	3.0	Yes	N/A
DatabaseMetaData getMetaData ()	1.0	Yes	N/A
int getTransactionIsolation ()	1.0	Yes	N/A
Map getTypeMap ()	2.0 Core	Yes	Always returns empty java.util.HashMap.
SQLWarning getWarnings ()	1.0	Yes	N/A

Table A–6 (Cont.) Connection Object

Connection Object Methods	Version Introduced	Supported	Comments
boolean isClosed ()	1.0	Yes	N/A
boolean isReadOnly ()	1.0	Yes	N/A
boolean isValid ()	4.0	Yes	N/A
boolean isWrapperFor (Class<?> iface)	4.0	Yes	N/A
String nativeSQL (String)	1.0	Yes	Always returns same String as passed in.
CallableStatement prepareCall (String)	1.0	Yes	N/A
CallableStatement prepareCall (String, int, int)	2.0 Core	Yes	ResultSet.TYPE_SCROLL_ SENSITIVE downgraded to TYPE_SCROLL_INSENSITIVE for the DB2 driver.
CallableStatement prepareCall (String, int, int, int)	3.0	No	Throws "unsupported method" exception.
PreparedStatement prepareStatement (String)	1.0	Yes	N/A
PreparedStatement prepareStatement (String, int)	3.0	Yes	N/A
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 SQL Server driver. For all other drivers, throws "unsupported method" exception.
PreparedStatement prepareStatement (String, String [])	3.0	Yes	Supported for the SQL Server driver. 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	N/A
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.

Table A–6 (Cont.) Connection Object

Connection Object Methods	Version Introduced	Supported	Comments
void setAutoCommit (boolean)	1.0	Yes	N/A
void setCatalog (String)	1.0	Yes	Supported for all drivers
String setClientInfo (Properties)	4.0	Yes	N/A
String setClientInfo (String, String)	4.0	Yes	N/A
void setHoldability (int)	3.0	Yes	Holdability parameter value is ignored.
void setReadOnly (boolean)	1.0	Yes	N/A
Savepoint setSavepoint ()	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 8.2 for Linux/UNIX/Windows.
Savepoint setSavepoint (String)	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 for Linux/UNIX/Windows.
void setTransactionIsolation (int)	1.0	Yes	N/A
void setTypeMap (Map)	2.0 Core	Yes	Ignored.
<T> T unwrap(Class<T> iface)	4.0	Yes	

A.2.6 ConnectionEventListener Object

The following table lists functionality supported for each ConnectionEventListener object.

Table A–7 ConnectionEventListener Object

ConnectionEventListener Object Methods	Version Introduced	Supported	Comments
void connectionClosed (event))	3.0	Yes	
void connectionErrorOccurred (event)	3.0	Yes	

A.2.7 ConnectionPoolDataSource Object

The following table lists functionality supported for each ConnectionPoolDataSource object.

Table A–8 ConnectionPoolDataSource Object

ConnectionPoolDataSource Object Methods	Version Introduced	Supported	Comments
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	

A.2.8 DatabaseMetaData Object

The following table lists functionality supported for each DatabaseMetaData object.

Table A–9 DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
boolean autoCommitFailureClosesAllResultSets ()	4.0	Yew	
boolean allProceduresAreCallable ()	1.0	Yes	N/A
boolean allTablesAreSelectable ()	1.0	Yes	N/A
boolean dataDefinitionCausesTransaction Commit ()	1.0	Yes	N/A
boolean dataDefinitionIgnoredInTransactions ()	1.0	Yes	N/A
boolean deletesAreDetected (int)	2.0 Core	Yes	N/A
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.

Table A–9 (Cont.) DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
ResultSet getBestRowIdentifier (String, String, String, int, boolean)	1.0	Yes	N/A
ResultSet getCatalogs ()	1.0	Yes	N/A
String getCatalogSeparator ()	1.0	Yes	N/A
String getCatalogTerm ()	1.0	Yes	N/A
String getClientInfoProperties ()	4.0	Yes	N/A
ResultSet getColumnPrivileges (String, String, String, String)	1.0	Yes	N/A
ResultSet getColumns (String, String, String, String)	1.0	Yes	N/A
Connection getConnection ()	2.0 Core	Yes	N/A
ResultSet getCrossReference (String, String, String, String, String, String)	1.0	Yes	N/A
ResultSet getFunctions ()	4.0	Yes	
ResultSet getFunctionColumns ()	4.0	Yes	
int getDatabaseMajorVersion ()	3.0	Yes	N/A
int getDatabaseMinorVersion ()	3.0	Yes	N/A
String getDatabaseProductName ()	1.0	Yes	For Sybase, returns "SQL Server," which is the string returned internally by the Sybase database server. This value may not be the same return as seen with other JDBC drivers, including the Sybase JConnect JDBC drivers.
String getDatabaseProductVersion ()	1.0	Yes	N/A
int getDefaultTransactionIsolation ()	1.0	Yes	N/A

Table A–9 (Cont.) DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
int getDriverMajorVersion ()	1.0	Yes	N/A
int getDriverMinorVersion ()	1.0	Yes	N/A
String getDriverName ()	1.0	Yes	N/A
String getDriverVersion ()	1.0	Yes	N/A
ResultSet getExportedKeys (String, String, String)	1.0	Yes	N/A
String getExtraNameCharacters ()	1.0	Yes	N/A
String getIdentifierQuoteString ()	1.0	Yes	N/A
ResultSet getImportedKeys (String, String, String)	1.0	Yes	N/A
ResultSet getIndexInfo (String, String, String, boolean, boolean)	1.0	Yes	N/A
int getJDBCMajorVersion ()	3.0	Yes	N/A
int getJDBCMinorVersion ()	3.0	Yes	N/A
int getMaxBinaryLiteralLength ()	1.0	Yes	N/A
int getMaxCatalogNameLength ()	1.0	Yes	N/A
int getMaxCharLiteralLength ()	1.0	Yes	N/A
int getMaxColumnNameLength ()	1.0	Yes	N/A
int getMaxColumnsInGroupBy ()	1.0	Yes	N/A
int getMaxColumnsInIndex ()	1.0	Yes	N/A
int getMaxColumnsInOrderBy ()	1.0	Yes	N/A

Table A–9 (Cont.) DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
int getMaxColumnsInSelect ()	1.0	Yes	N/A
int getMaxColumnsInTable ()	1.0	Yes	N/A
int getMaxConnections ()	1.0	Yes	N/A
int getMaxCursorNameLength ()	1.0	Yes	N/A
int getMaxIndexLength ()	1.0	Yes	N/A
int getMaxProcedureNameLength ()	1.0	Yes	N/A
int getMaxRowSize ()	1.0	Yes	N/A
int getMaxSchemaNameLength ()	1.0	Yes	N/A
int getMaxStatementLength ()	1.0	Yes	N/A
int getMaxStatements ()	1.0	Yes	N/A
int getMaxTableNameLength ()	1.0	Yes	N/A
int getMaxTablesInSelect ()	1.0	Yes	N/A
int getMaxUserNameLength ()	1.0	Yes	N/A
String getNumericFunctions ()	1.0	Yes	N/A
ResultSet getPrimaryKeys (String, String, String)	1.0	Yes	N/A
ResultSet getProcedureColumns (String, String, String, String)	1.0	Yes	N/A
ResultSet getProcedures (String, String, String)	1.0	Yes	N/A
String getProcedureTerm ()	1.0	Yes	N/A
int getResultSetHoldability ()	3.0	Yes	N/A
ResultSet getSchemas ()	1.0	Yes	N/A

Table A–9 (Cont.) DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
ResultSet getSchemas (catalog, pattern)	4.0	Yes	
String getSchemaTerm ()	1.0	Yes	N/A
String getSearchStringEscape ()	1.0	Yes	N/A
String getSQLKeywords ()	1.0	Yes	N/A
int getSQLStateType ()	3.0	Yes	N/A
String getStringFunctions ()	1.0	Yes	N/A
ResultSet getSuperTables (String, String, String)	3.0	Yes	Empty result set is returned.
ResultSet getSuperTypes (String, String, String)	3.0	Yes	Empty result set is returned.
String getSystemFunctions ()	1.0	Yes	N/A
ResultSet getTablePrivileges (String, String, String)	1.0	Yes	N/A
ResultSet getTables (String, String, String, String [])	1.0	Yes	N/A
ResultSet getTableTypes ()	1.0	Yes	N/A
String getTimeDateFunctions ()	1.0	Yes	N/A
ResultSet getTypeInfo ()	1.0	Yes	N/A
ResultSet getUDTs (String, String, String, int [])	2.0 Core	No	Always returns empty ResultSet.
String getURL ()	1.0	Yes	N/A
String getUserName ()	1.0	Yes	N/A
ResultSet getVersionColumns (String, String, String)	1.0	Yes	N/A
boolean insertsAreDetected (int)	2.0 Core	Yes	N/A
boolean isCatalogAtStart ()	1.0	Yes	N/A
boolean isReadOnly ()	1.0	Yes	N/A
boolean locatorsUpdateCopy ()	3.0	Yes	N/A
boolean nullPlusNonNullIsNull ()	1.0	Yes	N/A

Table A–9 (Cont.) DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
boolean nullsAreSortedAtEnd ()	1.0	Yes	N/A
boolean nullsAreSortedAtStart ()	1.0	Yes	N/A
boolean nullsAreSortedHigh ()	1.0	Yes	N/A
boolean nullsAreSortedLow ()	1.0	Yes	N/A
boolean othersDeletesAreVisible (int)	2.0 Core	Yes	N/A
boolean othersInsertsAreVisible (int)	2.0 Core	Yes	N/A
boolean othersUpdatesAreVisible (int)	2.0 Core	Yes	N/A
boolean ownDeletesAreVisible (int)	2.0 Core	Yes	N/A
boolean ownInsertsAreVisible (int)	2.0 Core	Yes	N/A
boolean ownUpdatesAreVisible (int)	2.0 Core	Yes	N/A
boolean storesLowerCaseIdentifiers ()	1.0	Yes	N/A
boolean storesLowerCaseQuoted Identifiers ()	1.0	Yes	N/A
boolean storesMixedCaseIdentifiers ()	1.0	Yes	N/A
boolean storesMixedCaseQuoted Identifiers ()	1.0	Yes	N/A
boolean storesUpperCaseIdentifiers ()	1.0	Yes	N/A
boolean storesUpperCaseQuoted Identifiers ()	1.0	Yes	N/A
boolean supportsAlterTableWith AddColumn ()	1.0	Yes	N/A

Table A–9 (Cont.) DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
boolean supportsAlterTableWith DropColumn ()	1.0	Yes	N/A
boolean supportsANSI92EntryLevelSQL ()	1.0	Yes	N/A
boolean supportsANSI92FullSQL ()	1.0	Yes	N/A
boolean supportsANSI92Intermediate SQL ()	1.0	Yes	N/A
boolean supportsBatchUpdates ()	2.0 Core	Yes	N/A
boolean supportsCatalogsInData Manipulation ()	1.0	Yes	N/A
boolean supportsCatalogsInIndex Definitions ()	1.0	Yes	N/A
boolean supportsCatalogsInPrivilege Definitions ()	1.0	Yes	N/A
boolean supportsCatalogsInProcedure Calls ()	1.0	Yes	N/A
boolean supportsCatalogsInTable Definitions ()	1.0	Yes	N/A
boolean supportsColumnAliasing ()	1.0	Yes	N/A
boolean supportsConvert ()	1.0	Yes	N/A
boolean supportsConvert (int, int)	1.0	Yes	N/A
boolean supportsCoreSQLGrammar ()	1.0	Yes	N/A
boolean supportsCorrelatedSubqueries ()	1.0	Yes	N/A

Table A–9 (Cont.) DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
boolean supportsDataDefinitionAndData ManipulationTransactions ()	1.0	Yes	N/A
boolean supportsDataManipulation TransactionsOnly ()	1.0	Yes	N/A
boolean supportsDifferentTableCorrelation Names ()	1.0	Yes	N/A
boolean supportsExpressionsIn OrderBy ()	1.0	Yes	N/A
boolean supportsExtendedSQLGrammar ()	1.0	Yes	N/A
boolean supportsFullOuterJoins ()	1.0	Yes	N/A
boolean supportsGetGeneratedKeys ()	3.0	Yes	N/A
boolean supportsGroupBy ()	1.0	Yes	N/A
boolean supportsGroupByBeyondSelect ()	1.0	Yes	N/A
boolean supportsGroupByUnrelated ()	1.0	Yes	N/A
boolean supportsIntegrityEnhancement Facility ()	1.0	Yes	N/A
boolean supportsLikeEscapeClause ()	1.0	Yes	N/A
boolean supportsLimitedOuterJoins ()	1.0	Yes	N/A
boolean supportsMinimumSQLGrammar ()	1.0	Yes	N/A
boolean supportsMixedCaseIdentifiers ()	1.0	Yes	N/A

Table A–9 (Cont.) DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
boolean supportsMixedCaseQuoted Identifiers ()	1.0	Yes	N/A
boolean supportsMultipleOpenResults ()	3.0	Yes	N/A
boolean supportsMultipleResultSets ()	1.0	Yes	N/A
boolean supportsMultipleTransactions ()	1.0	Yes	N/A
boolean supportsNamedParameters ()	3.0	Yes	N/A
boolean supportsNonNullableColumns ()	1.0	Yes	N/A
boolean supportsOpenCursorsAcross Commit ()	1.0	Yes	N/A
boolean supportsOpenCursorsAcross Rollback ()	1.0	Yes	N/A
boolean supportsOpenStatements Across Commit ()	1.0	Yes	N/A
boolean supportsOpenStatements Across Rollback ()	1.0	Yes	N/A
boolean supportsOrderByUnrelated ()	1.0	Yes	N/A
boolean supportsOuterJoins ()	1.0	Yes	N/A
boolean supportsPositionedDelete ()	1.0	Yes	N/A
boolean supportsPositionedUpdate ()	1.0	Yes	N/A
boolean supportsResultSetConcurrency (int, int)	2.0 Core	Yes	N/A

Table A–9 (Cont.) DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
boolean supportsResultSetHoldability (int)	3.0	Yes	N/A
boolean supportsResultSetType (int)	2.0 Core	Yes	N/A
boolean supportsSavePoints ()	3.0	Yes	N/A
boolean supportsSchemasInData Manipulation ()	1.0	Yes	N/A
boolean supportsSchemasInIndex Definitions ()	1.0	Yes	N/A
boolean supportsSchemasIn PrivilegeDefinitions ()	1.0	Yes	N/A
boolean supportsSchemasInProcedure Calls ()	1.0	Yes	N/A
boolean supportsSchemasInTable Definitions ()	1.0	Yes	N/A
boolean supportsSelectForUpdate ()	1.0	Yes	N/A
boolean supportsStoredFunctions UsingCallSyntax ()	4.0	Yes	
boolean supportsStoredProcedures ()	1.0	Yes	N/A
boolean supportsSubqueriesIn Comparisons ()	1.0	Yes	N/A
boolean supportsSubqueriesInExists ()	1.0	Yes	N/A
boolean supportsSubqueriesInIns ()	1.0	Yes	N/A
boolean supportsSubqueriesIn Quantifieds ()	1.0	Yes	N/A
boolean supportsTableCorrelation Names ()	1.0	Yes	N/A

Table A–9 (Cont.) DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
boolean supportsTransactionIsolationLevel (int)	1.0	Yes	N/A
boolean supportsTransactions ()	1.0	Yes	N/A
boolean supportsUnion ()	1.0	Yes	N/A
boolean supportsUnionAll ()	1.0	Yes	N/A
boolean updatesAreDetected (int)	2.0 Core	Yes	N/A
boolean usesLocalFilePerTable ()	1.0	Yes	N/A
boolean usesLocalFiles ()	1.0	Yes	N/A

A.2.9 Data Source Object

The following table lists functionality supported for each data source object.

Note: The DataSource object implements the `javax.naming.Referenceable` and `java.io.Serializable` interfaces.

Table A–10 Data Source Object

Data Source Object Methods	Version Introduced	Supported	Comments
Connection getConnection ()	2.0 Optional	Yes	N/A
Connection getConnection (String, String)	1.0	Yes	N/A
int getLoginTimeout ()	1.0	Yes	N/A
PrintWriter getLogWriter ()	1.0	Yes	N/A
boolean isWrapperFor (Class<?> iface)	1.0	Yes	N/A
void setLoginTimeout (int)			
void setLogWriter (PrintWriter)			
<T> T unwrap(Class<T> iface)			

A.2.10 Driver Object

The following table lists functionality supported for each Driver object.

Table A–11 Driver Object

Driver Object Methods	Version Introduced	Supported	Comments
boolean acceptsURL (String)	1.0	Yes	N/A
Connection connect (String, Properties)	1.0	Yes	N/A
int getMajorVersion ()	1.0	Yes	N/A
int getMinorVersion ()	1.0	Yes	N/A
DriverPropertyInfo [] getPropertyInfo (String, Properties)	1.0	Yes	N/A

A.2.11 ParameterMetaData Object

The following table lists functionality supported for each ParameterMetaData object.

Table A–12 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 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
int getParameterCount ()	3.0	Yes	N/A
int getParameterMode (int)	3.0	Yes	The DB2 driver supports parameter metadata for stored procedures for DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
int getParameterType (int)	3.0	Yes	The DB2 driver supports parameter metadata for stored procedures for DB2 v8.1 and v8.2 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 v8.1 and v8.2 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 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.

Table A–12 (Cont.) ParameterMetaData Object

ParameterMetaData Object Methods	Version Introduced	Supported	Comments
int getScale (int)	3.0	Yes	The DB2 driver supports parameter metadata for stored procedures for DB2 v8.1 and v8.2 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 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
boolean isSigned (int)	3.0	Yes	The DB2 driver supports parameter metadata for stored procedures for DB2 v8.1 and v8.2 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	N/A
<T> T unwrap(Class<T> iface)	4.0	Yes	

A.2.12 PooledConnection Object

The following table lists functionality supported for each PooledConnection object.

Table A–13 PooledConnection Object

PooledConnection Object Methods	Version Introduced	Supported	Comments
void addConnectionEventListener (ConnectionEventListener)	2.0 Optional	Yes	N/A
void addStatementEventListener (listener)	4.0	Yes	N/A
void close()	2.0 Optional	Yes	N/A

Table A–13 (Cont.) PooledConnection Object

PooledConnection Object Methods	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 removeConnectionEventListener (ConnectionEventListener)	2.0 Optional	Yes	N/A
void removeStatementEventListener (listener)	4.0	Yes	

A.2.13 PreparedStatement Object

The following table lists functionality supported for each PreparedStatement object.

Table A–14 PreparedStatement Object

PreparedStatement Object Methods	Version Introduced	Supported	Comments
void addBatch ()	2.0 Core	Yes	N/A
void clearParameters ()	1.0	Yes	N/A
boolean execute ()	1.0	Yes	N/A
ResultSet executeQuery ()	1.0	Yes	N/A
int executeUpdate ()	1.0	Yes	N/A
ResultSetMetaData getMetaData ()	2.0 Core	Yes	N/A
ParameterMetaData getParameterMetaData ()	3.0	Yes	N/A
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	

Table A–14 (Cont.) PreparedStatement Object

PreparedStatement Object Methods	Version Introduced	Supported	Comments
void setAsciiStream (int, InputStream, int)	1.0	Yes	N/A
void setAsciiStream (int, InputStream, long)	4.0	Yes	
void setBigDecimal (int, BigDecimal)	1.0	Yes	N/A
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.
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.

Table A–14 (Cont.) PreparedStatement Object

PreparedStatement Object Methods	Version Introduced	Supported	Comments
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.
void setBoolean (int, boolean)	1.0	Yes	N/A
void setByte (int, byte)	1.0	Yes	N/A
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	N/A
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 setDate (int, Date)	1.0	Yes	N/A
void setDate (int, Date, Calendar)	2.0 Core	Yes	N/A
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 setDouble (int, double)	1.0	Yes	N/A
void setDate (int, Date)	4.0	Yes	
void setFloat (int, float)	1.0	Yes	N/A
void setInt (int, int)	1.0	Yes	N/A

Table A–14 (Cont.) PreparedStatement Object

PreparedStatement Object Methods	Version Introduced	Supported	Comments
void setDate (int, Date, Calendar)	2.0 Core	Yes	
void setLong (int, long)	1.0	Yes	N/A
void setNull (int, int)	1.0	Yes	N/A
void setNull (int, int, String)	2.0 Core	Yes	N/A
void setObject (int, Object)	1.0	Yes	N/A
void setObject (int, Object, int)	1.0	Yes	N/A
void setObject (int, Object, int, int)	1.0	Yes	N/A
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 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	N/A
)	1.0	Yes	N/A
void setSCLXML			

Table A–14 (Cont.) PreparedStatement Object

PreparedStatement Object Methods	Version Introduced	Supported	Comments
void setTime (int, Time)	1.0	Yes	N/A
void setTime (int, Time, Calendar)	2.0 Core	Yes	N/A
void setTimestamp (int, Timestamp)	1.0	Yes	N/A
void setTimestamp (int, Timestamp, Calendar)	2.0 Core	Yes	N/A
void setUnicodeStream (int, InputStream, int)	1.0	No	Throws "unsupported method" exception. This method was deprecated in JDBC 2.0.
void setURL (int, URL)	3.0	No	Throws "unsupported method" exception.

A.2.14 Ref Object

The following table lists functionality supported for each Ref object.

Table A–15 Ref Object

Ref Object Methods	Version Introduced	Supported	Comments
(all)	2.0 Core	No	N/A

A.2.15 ResultSet Object

The following table lists functionality supported for each ResultSet object.

Table A–16 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
boolean absolute (int)	2.0 Core	Yes	N/A
void afterLast ()	2.0 Core	Yes	N/A
void beforeFirst ()	2.0 Core	Yes	N/A
void cancelRowUpdates ()	2.0 Core	Yes	N/A
void clearWarnings ()	1.0	Yes	N/A
void close ()	1.0	Yes	N/A
void deleteRow ()	2.0 Core	Yes	N/A
int findColumn (String)	1.0	Yes	N/A
boolean first ()	2.0 Core	Yes	N/A
Array getArray (int)	2.0 Core	No	Throws "unsupported method" exception.
Array getArray (String)	2.0 Core	No	Throws "unsupported method" exception.

Table A–16 (Cont.) ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
InputStream getAsciiStream (int)	1.0	Yes	N/A
InputStream getAsciiStream (String)	1.0	Yes	N/A
BigDecimal getBigDecimal (int)	2.0 Core	Yes	N/A
BigDecimal getBigDecimal (int, int)	1.0	Yes	N/A
BigDecimal getBigDecimal (String)	2.0 Core	Yes	N/A
BigDecimal getBigDecimal (String, int)	1.0	Yes	N/A
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.

Table A–16 (Cont.) ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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	N/A
boolean getBoolean (String)	1.0	Yes	N/A
byte getByte (int)	1.0	Yes	N/A
byte getByte (String)	1.0	Yes	N/A
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.
Reader getCharacterStream (int)	2.0 Core	Yes	N/A
Reader getCharacterStream (String)	2.0 Core	Yes	N/A
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.

Table A–16 (Cont.) ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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	N/A
String getCursorName ()	1.0	No	Throws "unsupported method" exception.
Date getDate (int)	1.0	Yes	N/A
Date getDate (int, Calendar)	2.0 Core	Yes	N/A
Date getDate (String)	1.0	Yes	N/A
Date getDate (String, Calendar)	2.0 Core	Yes	N/A
double getDouble (int)	1.0	Yes	N/A
double getDouble (String)	1.0	Yes	N/A
int getFetchDirection ()	2.0 Core	Yes	N/A
int getFetchSize ()	2.0 Core	Yes	N/A
float getFloat (int)	1.0	Yes	N/A
float getFloat (String)	1.0	Yes	N/A
int getHoldability ()	1	Yes	
int getInt (int)	1.0	Yes	N/A
int getInt (String)	1.0	Yes	N/A
long getLong (int)	1.0	Yes	N/A
long getLong (String)	1.0	Yes	N/A
ResultSetMetaData getMetaData ()	1.0	Yes	N/A
Reader getNCharacterStream (int)	4.0	Yes	
Reader getNCharacterStream (String)	4.0	Yes	
NClob getNClob (int)4	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	N/A

Table A–16 (Cont.) ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
Object getObject (String)	1.0	Yes	N/A
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	N/A
short getShort (int)	1.0	Yes	N/A
short getShort (String)	1.0	Yes	N/A
SQLXML getSQLXML (int)	4.0	Yes	
SQLXML getSQLXML (String)	4.0	Yes	
Statement getStatement ()	2.0 Core	Yes	N/A
String getString (int)	1.0	Yes	N/A
String getString (String)	1.0	Yes	N/A
Time getTime (int)	1.0	Yes	N/A
Time getTime (int, Calendar)	2.0 Core	Yes	N/A
Time getTime (String)	1.0	Yes	N/A
Time getTime (String, Calendar)	2.0 Core	Yes	N/A
Timestamp getTimestamp (int)	1.0	Yes	N/A
Timestamp getTimestamp (int, Calendar)	2.0 Core	Yes	N/A
Timestamp getTimestamp (String)	1.0	Yes	N/A
Timestamp getTimestamp (String, Calendar)	2.0 Core	Yes	N/A
int getType ()	2.0 Core	Yes	N/A
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.

Table A–16 (Cont.) ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
SQLWarning getWarnings ()	1.0	Yes	N/A
void insertRow ()	2.0 Core	Yes	N/A
boolean isAfterLast ()	2.0 Core	Yes	N/A
boolean isBeforeFirst ()	2.0 Core	Yes	N/A
boolean isClosed ()	4.0	Yes	
boolean isFirst ()	2.0 Core	Yes	N/A
boolean isLast ()	2.0 Core	Yes	N/A
boolean isWrapperFor (Class<?> iface)	4.0	Yes	
boolean last ()	2.0 Core	Yes	N/A
void moveToCurrentRow ()	2.0 Core	Yes	N/A
void moveToInsertRow ()	2.0 Core	Yes	N/A
boolean next ()	1.0	Yes	N/A
boolean previous ()	2.0 Core	Yes	N/A
void refreshRow ()	2.0 Core	Yes	N/A
boolean relative (int)	2.0 Core	Yes	N/A
boolean rowDeleted ()	2.0 Core	Yes	N/A
boolean rowInserted ()	2.0 Core	Yes	N/A
boolean rowUpdated ()	2.0 Core	Yes	N/A
void setFetchDirection (int)	2.0 Core	Yes	N/A
void setFetchSize (int)	2.0 Core	Yes	N/A
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	N/A
void updateAsciiStream (int, InputStream, long)	4.0	Yes	
void updateAsciiStream (String, InputStream)	4.0	Yes	
void updateAsciiStream (String, InputStream, int)	2.0 Core	Yes	N/A
void updateAsciiStream (String, InputStream, long)	4.0	Yes	
void updateBigDecimal (int, BigDecimal)	2.0 Core	Yes	N/A

Table A–16 (Cont.) ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
void updateBigDecimal (String, BigDecimal)	2.0 Core	Yes	N/A
void updateBinaryStream (int, InputStream)	4.0	Yes	
void updateBinaryStream (int, InputStream, int)	2.0 Core	Yes	N/A
void updateBinaryStream (int, InputStream, long)	4.0	Yes	
void updateBinaryStream (String, InputStream)	4.0	Yes	
void updateBinaryStream (String, InputStream, int)	2.0 Core	Yes	N/A
void updateBinaryStream (String, InputStream, long)	4.0	Yes	
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)	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.

Table A–16 (Cont.) ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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	N/A
void updateBoolean (String, boolean)	2.0 Core	Yes	N/A
void updateByte (int, byte)	2.0 Core	Yes	N/A
void updateByte (String, byte)	2.0 Core	Yes	N/A
void updateBytes (int, byte [])	2.0 Core	Yes	N/A
void updateBytes (String, byte [])	2.0 Core	Yes	N/A
void updateCharacterStream (int, Reader)	4.0	Yes	

Table A–16 (Cont.) ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
void updateCharacterStream (int, Reader, int)	2.0 Core	Yes	N/A
void updateCharacterStream (int, Reader, long)	4.0	Yes	
void updateCharacterStream (String, Reader)	4.0	Yes	
void updateCharacterStream (String, Reader, int)	2.0 Core	Yes	N/A
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.
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	N/A

Table A–16 (Cont.) ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
void updateDate (String, Date)	2.0 Core	Yes	N/A
void updateDouble (int, double)	2.0 Core	Yes	N/A
void updateDouble (String, double)	2.0 Core	Yes	N/A
void updateFloat (int, float)	2.0 Core	Yes	N/A
void updateFloat (String, float)	2.0 Core	Yes	N/A
void updateInt (int, int)	2.0 Core	Yes	N/A
void updateInt (String, int)	2.0 Core	Yes	N/A
void updateLong (int, long)	2.0 Core	Yes	N/A
void updateLong (String, long)	2.0 Core	Yes	N/A
void updateNCharacterStream (int, Reader)	4.0	Yes	
void updateNCharacterStream (int, Reader, long)	4.0	Yes	
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	N/A

Table A–16 (Cont.) ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
void updateNull (String)	2.0 Core	Yes	N/A
void updateObject (int, Object)	2.0 Core	Yes	N/A
void updateObject (int, Object, int)	2.0 Core	Yes	N/A
void updateObject (String, Object)	2.0 Core	Yes	N/A
void updateObject (String, Object, int)	2.0 Core	Yes	N/A
void updateRef (int, Ref)	3.0	No	Throws "unsupported method" exception.
void updateRef (String, Ref)	3.0	No	Throws "unsupported method" exception.
void updateRow ()	2.0 Core	Yes	N/A
void updateShort (int, short)	2.0 Core	Yes	N/A
void updateShort (String, short)	2.0 Core	Yes	N/A
void updateSQLXML (int, SQLXML)	4.0	Yes	
void updateSQLXML (String, SQLXML)	4.0	Yes	
void updateString (int, String)	2.0 Core	Yes	N/A
void updateString (String, String)	2.0 Core	Yes	N/A
void updateTime (int, Time)	2.0 Core	Yes	N/A
void updateTime (String, Time)	2.0 Core	Yes	N/A
void updateTimestamp (int, Timestamp)	2.0 Core	Yes	N/A
void updateTimestamp (String, Timestamp)	2.0 Core	Yes	N/A
boolean wasNull ()	1.0	Yes	N/A

A.2.16 ResultSetMetaData Object

The following table lists functionality supported for each ResultSetMetaData object.

Table A–17 ResultSetMetaData Object

ResultSetMetaData Object Methods	Version Introduced	Supported	Comments
String getCatalogName (int)	1.0	Yes	N/A

Table A–17 (Cont.) ResultSetMetaData Object

ResultSetMetaData Object Methods	Version Introduced	Supported	Comments
String getColumnClassName (int)	2.0 Core	Yes	N/A
int getColumnCount ()	1.0	Yes	N/A
int getColumnDisplaySize (int)	1.0	Yes	N/A
String getColumnLabel (int)	1.0	Yes	N/A
String getColumnName (int)	1.0	Yes	N/A
int getColumnType (int)	1.0	Yes	N/A
String getColumnTypeName (int)	1.0	Yes	N/A
int getPrecision (int)	1.0	Yes	N/A
int getScale (int)	1.0	Yes	N/A
String getSchemaName (int)	1.0	Yes	N/A
String getTableName (int)	1.0	Yes	For versions 3.4 and higher: ■ By default, `getTableName` returns an empty string for Informix, and SQL Server Type 4 drivers. ■ To return a table name for the Informix, and SQL Server Type 4 drivers, add the following property to the connection pool Properties field: `ResultsetMetaDataOptions=1` See "JDBC Data Source: Configuration: Connection Pool" in the *Administration Console Online Help*.
boolean isAutoIncrement (int)	1.0	Yes	N/A
boolean isCaseSensitive (int)	1.0	Yes	N/A
boolean isCurrency (int)	1.0	Yes	N/A
boolean isDefinitelyWritable (int)	1.0	Yes	N/A
int isNullable (int)	1.0	Yes	N/A
boolean isReadOnly (int)	1.0	Yes	N/A
boolean isSearchable (int)	1.0	Yes	N/A
boolean isSigned (int)	1.0	Yes	N/A

Table A–17 (Cont.) ResultSetMetaData Object

ResultSetMetaData Object Methods	Version Introduced	Supported	Comments
boolean isWrapperFor (Class<?> iface)	4.0	Yes	
boolean isWritable (int)	1.0	Yes	N/A
<T> T unwrap(Class<T> iface)	4.0	Yes	

A.2.17 RowSet Object

The following table lists functionality supported for each RowSet object.

Table A–18 RowSet Object

RowSet Object Methods	Version Introduced	Supported	Comments
(all)	2.0 Optional	No	

A.2.18 SavePoint Object

The following table lists functionality supported for each SavePoint object.

Table A–19 SavePoint Object

SavePoint Object Methods	Version 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.

A.2.19 Statement Object

The following table lists functionality supported for each Statement object.

Table A–20 Statement Object

Statement Object Methods	Version Introduced	Supported	Comments
void addBatch (String)	2.0 Core	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.

Table A–20 (Cont.) Statement Object

Statement Object Methods	Version Introduced	Supported	Comments
void cancel ()	1.0	Yes	The DB2 driver cancels the execution of the statement with DB2 v8.1 and v8.2 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 throws an "unsupported method" exception. The 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	N/A
void clearWarnings ()	1.0	Yes	N/A
void close ()	1.0	Yes	N/A
boolean execute (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.
boolean execute (String, int)	3.0	Yes	N/A
boolean execute (String, int [])	3.0	Yes	Supported for the SQL Server drivers. For all other drivers, throws "unsupported method" exception.
boolean execute (String, String [])	3.0	Yes	Supported for the SQL Server drivers. For all other drivers, throws "unsupported method" exception.
int [] executeBatch ()	2.0 Core	Yes	N/A
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	N/A
int executeUpdate (String, int [])	3.0	Yes	Supported for the SQL Server drivers. For all other drivers, throws "unsupported method" exception.

Table A–20 (Cont.) Statement Object

Statement Object Methods	Version Introduced	Supported	Comments
int executeUpdate (String, String [])	3.0	Yes	Supported for the SQL Server drivers. For all other drivers, throws "unsupported method" exception.
Connection getConnection ()	2.0 Core	Yes	N/A
int getFetchDirection ()	2.0 Core	Yes	N/A
int getFetchSize ()	2.0 Core	Yes	N/A
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.
int getMaxFieldSize ()	1.0	Yes	N/A
int getMaxRows ()	1.0	Yes	N/A
boolean getMoreResults ()	1.0	Yes	N/A
boolean getMoreResults (int)	3.0	Yes	N/A
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 returns 0. The SQL Server and Sybase drivers return the timeout value, in seconds, set for the statement.
ResultSet getResultSet ()	1.0	Yes	N/A
int getResultSetConcurrency ()	2.0 Core	Yes	N/A
int getResultSetHoldability ()	3.0	Yes	N/A
int getResultSetType ()	2.0 Core	Yes	N/A
int getUpdateCount ()	1.0	Yes	N/A
SQLWarning getWarnings ()	1.0	Yes	N/A
boolean isClosed ()	4.0	Yes	

Table A–20 (Cont.) Statement Object

Statement Object Methods	Version Introduced	Supported	Comments
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	N/A
void setFetchSize (int)	2.0 Core	Yes	N/A
void setMaxFieldSize (int)	1.0	Yes	N/A
void setMaxRows (int)	1.0	Yes	N/A
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 SQL Server and Sybase driver supports 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 setPoolable (boolean)4	4.0	Yes	

Table A–20 (Cont.) Statement Object

Statement Object Methods	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 throw an "unsupported method" exception. The 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
<T> T unwrap(Class<T> iface)	4.0	Yes	

A.2.20 StatementEventListener Object

The following table lists functionality supported for each StatementEventListener object.

Table A–21 StatementEventListener Object Object

StatementEventListener Object Methods	Version Introduced	Supported	Comments
void statementClosed (event)	4.0	Yes	N/A
void statementErrorOccurred (event)	4.0	Yes	N/A

A.2.21 Struct Object

The following table lists functionality supported for each Struct object.

Table A–22 Struct Object

Statement Object Methods	Version Introduced	Supported	Comments
(all)	2.0	No	N/A

A.2.22 XAConnection Object

The following table lists functionality supported for each XAConnection object.

Table A–23 XAConnection Object

XAConnection Object Methods	Version 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.

A.2.23 XADataSource Object

The following table lists functionality supported for each XADataSource object.

Table A–24 XADataSource Object

XADataSource Object Methods	Version 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 v8.1for z/OS.

A.2.24 XAResource Object

The following table lists functionality supported for each XAResource object.

Table A–25 XAResource Object

XAResource Object Methods	Version 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

GetTypeInfo

The following tables provide results returned from the `DataBaseMetaData.getTypeInfo` method for all of the WebLogic Type 4 JDBC drivers. The `getTypeInfo()` method retrieves 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.

- Section B.1, "DB2 Driver"
- Section B.2, "Informix Driver"
- Section B.3, "SQL Server Driver"
- Section B.4, "Sybase Driver"

B.1 DB2 Driver

Table B–1 provides getTypeInfo results for all DB2 databases supported by the DB2 driver (see Chapter 3, "The DB2 Driver").

Table B–1 getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = bigint [1]	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -5 (BIGINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = bigint
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 19
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = binary[2]	AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = *length* DATA_TYPE = -2 (BINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = BINARY(X' LITERAL_SUFFIX = ') LOCAL_TYPE_NAME = binary MAXIMUM_SCALE = NULL MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 255 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = blob [3]	AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = *length* DATA_TYPE = 2004 (BLOB) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = BLOB MAXIMUM_SCALE = NULL MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = char	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *length*
	DATA_TYPE = 1 (CHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = char
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 254 (DB2 for Linux/UNIX/Windows), 255 (DB2 for z/OS), 32765 (DB2 for iSeries)
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = char for bit data	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = *length*
	DATA_TYPE = -2 (BINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = X'
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = char for bit data
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 254 (DB2 for Linux/UNIX/Windows), 255 (DB2 for z/OS), 32765 (DB2 for iSeries)
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = clob	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *length*
	DATA_TYPE = 2005 (CLOB)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = clob
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 1
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = date	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 91 (DATE)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = {d'
	LITERAL_SUFFIX = '}
	LOCAL_TYPE_NAME = date
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 10
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = dbclob [4]	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = (length) (DB2 for Linux/UNIX/Windows and DB2 for z/OS (length) CCSID 13488 (DB2 V5R2, V5R3 for iSeries)
	DATA_TYPE = 2005 (DBCLOB)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = dbclob
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 1
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = decfloat [5]	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = *precision*
	DATA_TYPE = -3 (DECIMAL)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = NULL
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 34
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = decimal	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = (*precision,scale*)
	DATA_TYPE = 3 (DECIMAL)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = decimal
	MAXIMUM_SCALE = 31
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 31
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = double	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 8 (DOUBLE)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = double
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 15
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = float	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 6 (FLOAT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = float
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 15
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = graphic	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = length
	DATA_TYPE = 1 (CHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = G'
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = char
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 127 (DB2 for Linux/UNIX/Windows), 127 (DB2 for z/OS), 16352 (DB2 for iSeries)
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = integer	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 4 (INTEGER)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = integer
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 10
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = long varchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = NULL
	DATA_TYPE = -1 (LONGVARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = long varchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 32700 (DB2 for Linux/UNIX/Windows,[6] 32704 (DB2 for z/OS) [7] 32700 (DB2 for iSeries) [8]
	SEARCHABLE = 1
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = long varchar for bit data	AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -4 (LONGVARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = long varchar for bit data MAXIMUM_SCALE = NULL MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32700 (DB2 for Linux/UNIX/Windows), 32698 (DB2 for z/OS), 32739 (DB2 for iSeries) SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = long vargraphic	AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = *length* DATA_TYPE = -1 (LONGVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = G' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = longvarchar MAXIMUM_SCALE = NULL MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 16352 SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = numeric	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = (*precision, scale*)
	DATA_TYPE = 2 (NUMERIC)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = numeric
	MAXIMUM_SCALE = 31
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX =10
	PRECISION = 31
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = real	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 7 (REAL)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = float(4)
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 7
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = rowid [9]	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = not null generated always
	DATA_TYPE = -2 (Binary)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = rowid
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = NULL
	NULLABLE = 0
	NUM_PREC_RADIX = NULL
	PRECISION = 40
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = true
TYPE_NAME = smallint	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 5 (SMALLINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = smallint
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 5
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = time	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 92 (TIME)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = {t'
	LITERAL_SUFFIX = '}
	LOCAL_TYPE_NAME = time
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 8
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = timestamp	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 93 (TIMESTAMP)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = {ts'
	LITERAL_SUFFIX = '}
	LOCAL_TYPE_NAME = timestamp
	MAXIMUM_SCALE = 6
	MINIMUM_SCALE = 6
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 26
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = varbinary[10]	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = length
	DATA_TYPE = -3 (VARVINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = VARBINARY(X'
	LITERAL_SUFFIX = ')
	LOCAL_TYPE_NAME = varbinary
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 32703
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *max length*
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = varchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 32704 (DB2 v7.*x* for Linux/UNIX/Windows), 32762 (DB2 v8.x and higher for Linux/UNIX/Windows), 32698 (DB2 for z/OS), 32739 (DB2 for iSeries)
	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

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = varchar for bit data	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = *max length*
	DATA_TYPE = -3 (VARBINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = X'
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = varchar() for bit data
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 32704 (DB2 v7.*x* for Linux/UNIX/Windows), 32762 (DB2 v8.x and higher for Linux/UNIX/Windows), 32698 (DB2 for z/OS), 32739 (DB2 for iSeries)
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = vargraphic	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = length
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = G'
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = varchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 16352
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–1 (Cont.) getTypeInfo for DB2

Type Name	Type Info/Value
TYPE_NAME = xml[11]	AUTO_INCREMENT = false
	CASE_SENSITIVE = true
	CREATE_PARAMS = NULL
	DATA_TYPE = 2005 (CLOB)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX =NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = xml
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 1
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

[1] Supported only for DB2 for Linux/UNIX/Windows, DB2 for iSeries, and DB2 v9.1 for z/O

[2] Supported only for DB2 v9.1 for z/OS

[3] Supported only for DB2 v8.1 and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.

[4] Supported only for DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.

[5] Supported only for DB2 v9.1 for z/OS

[6] Precision depends on several factors, such as the number of columns in the table and whether the columns allow NULL values. Refer to your IBM documentation for more information.

[7] Precision depends on several factors, such as the number of columns in the table and whether the columns allow NULL values. Refer to your IBM documentation for more information.

[8] Precision depends on several factors, such as the number of columns in the table and whether the columns allow NULL values. Refer to your IBM documentation for more information.

[9] Supported only for DB2 for z/OS and DB2 V5R2 and higher for iSeries.

[10] Supported only for DB2 v9.1 for z/OS

[11] Supported only for DB2 V9.1 for Linux/UNIX/Windows.and DB2 v9.1 for z/OS.

B.2 Informix Driver

Table B–2 provides getTypeInfo results for all Informix databases supported by the Informix driver (see Chapter 4, "The Informix Driver").

Table B–2 getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = blob	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 2004 (BLOB)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = blob
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 0
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = boolean	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -7 (BIT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = boolean
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 1
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–2 (Cont.) getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = byte	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -4 (LONGVARBINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = byte
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 0
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = char	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *length*
	DATA_TYPE = 1 (CHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = char
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 32766
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–2 (Cont.) getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = clob	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = NULL
	DATA_TYPE = 2005 (CLOB)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = clob
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 0
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = date	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 91 (DATE)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = {d'
	LITERAL_SUFFIX = '}
	LOCAL_TYPE_NAME = date
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 10
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–2 (Cont.) getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = datetime hour to second	AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 92 (TIME) FIXED_PREC_SCALE = false LITERAL_PREFIX = {t' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = datetime hour to second MAXIMUM_SCALE = 0 MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 8 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = datetime year to day	AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 91 (DATE) FIXED_PREC_SCALE = false LITERAL_PREFIX = {d' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = datetime year to day MAXIMUM_SCALE = NULL MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Table B–2 (Cont.) getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = datetime year to fraction(5)	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 93 (TIMESTAMP)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = {ts'
	LITERAL_SUFFIX = '}
	LOCAL_TYPE_NAME = datetime hour to fraction(5)
	MAXIMUM_SCALE = 5
	MINIMUM_SCALE = 5
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 25
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = datetime year to second	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 93 (TIMESTAMP)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = {ts'
	LITERAL_SUFFIX = '}
	LOCAL_TYPE_NAME = datetime hour to second
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 19
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–2 (Cont.) getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = decimal	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = *precision, scale*
	DATA_TYPE = 3 (DECIMAL)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = decimal
	MAXIMUM_SCALE = 32
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 32
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = float	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 6 (FLOAT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = float
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 15
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–2 (Cont.) getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = int8	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -5 (BIGINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = int8
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 19
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = integer	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 4 (INTEGER)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = integer
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 10
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–2 (Cont.) getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = lvarchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = NULL (Informix 9.2, 9.3), *max length* (Informix 9.4, 10)
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = lvarchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2048 (Informix 9.2, 9.3), 32739 (Informix 9.4, 10)
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = money	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = *precision,scale*
	DATA_TYPE = 3 (DECIMAL)
	FIXED_PREC_SCALE = true
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = money
	MAXIMUM_SCALE = 32
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 32
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–2 (Cont.) getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = nchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *length*
	DATA_TYPE = 1 (CHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = nchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 32766
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = nvarchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *max length*
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = nvarchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 254
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–2 (Cont.) getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = serial	AUTO_INCREMENT = true
	CASE_SENSITIVE = false
	CREATE_PARAMS = start
	DATA_TYPE = 4 (INTEGER)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = serial
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 10
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = serial8	AUTO_INCREMENT = true
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -5 (BIGINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = serial8
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 19
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–2 (Cont.) getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = smallfloat	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 7 (REAL)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = smallfloat
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 7
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = smallint	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 5 (SMALLINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = smallint
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 5
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–2 (Cont.) getTypeInfo for Informix

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = text	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = NULL
	DATA_TYPE = -1 (LONGVARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = text
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 0
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *max length*
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = varchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 254
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

B.3 SQL Server Driver

Table B–3 provides getTypeInfo results for for all Microsoft SQL Server databases supported by the SQL Server driver. See Chapter 6, "The MS SQL Server Driver."

Table B–3 getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = bigint [1]	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -5 (BIGINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = bigint
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 19
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = bigint identity [2]	AUTO_INCREMENT = true
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -5 (BIGINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = bigint identity
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 0
	NUM_PREC_RADIX = 10
	PRECISION = 19
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = binary	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = *length*
	DATA_TYPE = -2 (BINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = 0x
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = binary
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 8000
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = bit	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -7 (BIT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = bit
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 1
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = char	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = *length*
	DATA_TYPE = 1 (CHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = char
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 8000
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = datetime	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 93 (TIMESTAMP)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = datetime
	MAXIMUM_SCALE = 3
	MINIMUM_SCALE = 3
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 23
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = datetime2	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 93 (TIMESTAMP)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = datetime2
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = datetimeoffset	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = datetimeoffset
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 34
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = decimal	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = *precision,scale*
	DATA_TYPE = 3 (DECIMAL)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = decimal
	MAXIMUM_SCALE = 28 (SQL Server 7), [3] 38 (SQL Server 2000 and SQL Server 2005) [4]
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 28 (SQL Server 7) [5], 38 (SQL Server 2000 and SQL Server 2005) [6]
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = decimal() identity	AUTO_INCREMENT = true
	CASE_SENSITIVE = false
	CREATE_PARAMS = *precision*
	DATA_TYPE = 3 (DECIMAL)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = decimal() identity
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 0
	NUM_PREC_RADIX = 10
	PRECISION = 28 (SQL Server 7), 38 (SQL Server 2000 and SQL Server 2005)
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = float	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 6 (FLOAT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = float
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = 2
	PRECISION = 53
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = image	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -4 (LONGVARBINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = 0x
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = image
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 0
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = int	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 4 (INTEGER)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = int
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 10
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = int identity	AUTO_INCREMENT = true
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 4 (INTEGER)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = int identity
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 0
	NUM_PREC_RADIX = 10
	PRECISION = 10
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = money	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 3 (DECIMAL)
	FIXED_PREC_SCALE = true
	LITERAL_PREFIX = $
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = money
	MAXIMUM_SCALE = 4
	MINIMUM_SCALE = 4
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 19
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = nchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = *length*
	DATA_TYPE = 1 (CHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = N'
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = nchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 4000
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = ntext	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -1 (LONGVARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = N'
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = ntext
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 1073741823
	SEARCHABLE = 1
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = numeric	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = *precision,scale*
	DATA_TYPE = 2 (NUMERIC)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = numeric
	MAXIMUM_SCALE = 28 (SQL Server 7),[7] 38 (SQL Server 2000 and SQL Server 2005) [8]
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 28 (SQL Server 7),[9] 38 (SQL Server 2000 and SQL Server 2005)[10]
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = numeric() identity	AUTO_INCREMENT = true
	CASE_SENSITIVE = false
	CREATE_PARAMS = *precision*
	DATA_TYPE = 2 (NUMERIC)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = numeric() identity
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 0
	NUM_PREC_RADIX = 10
	PRECISION = 28 (SQL Server 7.0), 38 (SQL Server 2000 and SQL Server 2005)
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = nvarchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = *max length*
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = N'
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = nvarchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 4000
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = nvarchar(max) [11]	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -1 (LONGVARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = N'
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = nvarchar(max)
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 1073741823
	SEARCHABLE = 1
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = real	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 7 (REAL)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = real
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = 2
	PRECISION = 24
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = smalldatetime	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 93 (TIMESTAMP)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = smalldatetime
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 16
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = smallint	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 5 (SMALLINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = smallint
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 5
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = smallint identity	AUTO_INCREMENT = true
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 5 (SMALLINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = smallint identity
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 0
	NUM_PREC_RADIX = 10
	PRECISION = 5
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = smallmoney	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 3 (DECIMAL)
	FIXED_PREC_SCALE = true
	LITERAL_PREFIX = $
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = smallmoney
	MAXIMUM_SCALE = 4
	MINIMUM_SCALE = 4
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 10
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = sql_variant [12]	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = sql_variant
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 8000
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = sysname	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = N'
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = sysname
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 0
	NUM_PREC_RADIX = NULL
	PRECISION = 128
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = text	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -1 (LONGVARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = text
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 1
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = time	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 93 (TIMESTAMP)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = ’
	LITERAL_SUFFIX = ’
	LOCAL_TYPE_NAME = time
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 16
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = timestamp	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -2 (BINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = 0x
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = timestamp
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 0
	NUM_PREC_RADIX = NULL
	PRECISION = 8
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = tinyint	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -6 (TINYINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = tinyint
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 3
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = true

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = tinyint identity	AUTO_INCREMENT = true
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -6 (TINYINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = tinyint identity
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 0
	NUM_PREC_RADIX = 10
	PRECISION = 3
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = true
TYPE_NAME = uniqueidentifier	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 1(CHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = uniqueidentifier
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 36
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = varbinary	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = *max length*
	DATA_TYPE = -3 (VARBINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = 0x
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = varbinary
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 8000
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varbinary(max) [13]	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -4 (LONGVARBINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = 0x
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = varbinary(max)
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 0
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = varchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = max length
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = varchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 8000
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varchar(max) [14]	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -1 (LONGVARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = varchar(max)
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 1
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–3 (Cont.) getTypeInfo for SQL Server

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = xml [15]	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = NULL
	DATA_TYPE = -1 (LONGVARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = N'
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = xml
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 1073741823
	SEARCHABLE = 0
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

[1] Supported only for Microsoft SQL Server 2000 and higher.
[2] Supported only for Microsoft SQL Server 2000 and higher.
[3] Configurable server option for Microsoft SQL Server 2000 and higher.
[4] Configurable server option for Microsoft SQL Server 2000 and higher.
[5] Configurable server option for Microsoft SQL Server 2000 and higher.
[6] Configurable server option for Microsoft SQL Server 2000 and higher.
[7] Configurable server option for Microsoft SQL Server 2000 and higher.
[8] Configurable server option for Microsoft SQL Server 2000 and higher.
[9] Configurable server option for Microsoft SQL Server 2000 and higher.
[10] Configurable server option for Microsoft SQL Server 2000 and higher.
[11] Supported only for Microsoft SQL Server 2005.
[12] Supported only for Microsoft SQL Server 2000 and higher.
[13] Supported only for Microsoft SQL Server 2005.
[14] Supported only for Microsoft SQL Server 2005.
[15] Supported only for Microsoft SQL Server 2005.

B.4 Sybase Driver

Table B–4 provides getTypeInfo results for all Sybase databases supported by the Sybase driver (see Chapter 5, "The Sybase Driver").

Table B–4 getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = bigint [1]	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -5 (BIGINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = bigint
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 19
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBU
TYPE_NAME = binary	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = *length*
	DATA_TYPE = -2 (BINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = 0x
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = binary
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 255 (Sybase 11.*x*, 12.0)[2], 2048 (Sybase 12.5 and higher) [3]
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = bit	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -7 (BIT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = bit
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 0
	NUM_PREC_RADIX = NULL
	PRECISION = 1
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = char	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *length*
	DATA_TYPE = 1 (CHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = char
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 255 (Sybase 11.*x*, 12.0)[4], 2048 (Sybase 12.5 and higher) [5]
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = date [6]	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 91 (DATE)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = date
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 10
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = datetime	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 93 (TIMESTAMP)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = datetime
	MAXIMUM_SCALE = 3
	MINIMUM_SCALE = 3
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 23
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = decimal	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = *precision,scale*
	DATA_TYPE = 3 (DECIMAL)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = decimal
	MAXIMUM_SCALE = 38
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 38
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = float	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 6 (FLOAT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = float
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 15
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = image	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -4 (LONGVARBINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = 0x
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = image
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 1
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = int	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 4 (INTEGER)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = int
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 10
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = money	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 3 (DECIMAL)
	FIXED_PREC_SCALE = true
	LITERAL_PREFIX = $
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = money
	MAXIMUM_SCALE = 4
	MINIMUM_SCALE = 4
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 19
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = nchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = NULL
	DATA_TYPE = 1 (CHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = nchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 255 (Sybase 11.*x*, 12.0[7]), 2048 (Sybase 12.5 and higher) [8]
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = numeric	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = *precision,scale*
	DATA_TYPE = 2 (NUMERIC)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = numeric
	MAXIMUM_SCALE = 38
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 38
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = nvarchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *max length*
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = nvarchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 255 (Sybase 11.x, 12.0)[9], 2048 (Sybase 12.5 and higher) [10]
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = real	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 7 (REAL)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = real
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 7
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = smalldatetime	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 93 (TIMESTAMP)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = smalldatetime
	MAXIMUM_SCALE = 3
	MINIMUM_SCALE = 3
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 16
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = smallint	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 5 (SMALLINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = smallint
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 5
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = smallmoney	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 3 (DECIMAL)
	FIXED_PREC_SCALE = true
	LITERAL_PREFIX = $
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = smallmoney
	MAXIMUM_SCALE = 4
	MINIMUM_SCALE = 4
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 10
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = sysname	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *max length*
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = sysname
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 30
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = text	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = NULL
	DATA_TYPE = -1 (LONGVARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = text
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 1
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = time [11]	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 92 (TIME)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = time
	MAXIMUM_SCALE = 3
	MINIMUM_SCALE = 3
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 12
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = timestamp	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -3 (VARBINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX =0x
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = timestamp
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 8
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = tinyint	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -6 (TINTYINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = tinyint
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 3
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = true
TYPE_NAME = unsigned bigint [12]	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 3 (DECIMAL)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = unsigned bigint
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 20
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = true

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = unsigned int [13]	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = -5 (BIGINT)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = unsigned int
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 10
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = true
TYPE_NAME = unsigned smallint [14]	AUTO_INCREMENT = false
	CASE_SENSITIVE = false
	CREATE_PARAMS = NULL
	DATA_TYPE = 4 (INTEGER)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = NULL
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = unsigned smallint
	MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 5
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = true

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = unichar [15]	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *length*
	DATA_TYPE = 1 (CHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = unichar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION =2048
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = unitext	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = NULL
	DATA_TYPE = -1 (LONGVARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = unitext
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2147483647
	SEARCHABLE = 1
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = univarchar [16]	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *max length*
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = univarchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2048
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varbinary	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = false
	CREATE_PARAMS = *max length*
	DATA_TYPE = -3 (VARBINARY)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = 0x
	LITERAL_SUFFIX = NULL
	LOCAL_TYPE_NAME = varbinary
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 255 (Sybase 11.*x*, 12.0)[17], 2048 (Sybase 12.5 and higher) [18]
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B–4 (Cont.) getTypeInfo for Sybase

Type Info/Value	Type Info/Value (cont)
TYPE_NAME = varchar	AUTO_INCREMENT = NULL
	CASE_SENSITIVE = true
	CREATE_PARAMS = *max length*
	DATA_TYPE = 12 (VARCHAR)
	FIXED_PREC_SCALE = false
	LITERAL_PREFIX = '
	LITERAL_SUFFIX = '
	LOCAL_TYPE_NAME = varchar
	MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 255 (Sybase 11.*x*, 12.0)[19], 2048 (Sybase 12.5 and higher)[20]
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

[1] Supported only for Sybase 15.
[2] For Sybase 12.5.1 and higher, precision is determined by the server page size.
[3] For Sybase 12.5.1 and higher, precision is determined by the server page size.
[4] For Sybase 12.5.1 and higher, precision is determined by the server page size.
[5] For Sybase 12.5.1 and higher, precision is determined by the server page size.
[6] Supported only for Sybase 12.5.1 and higher.
[7] For Sybase 12.5.1 and higher, precision is determined by the server page size.
[8] For Sybase 12.5.1 and higher, precision is determined by the server page size.
[9] For Sybase 12.5.1 and higher, precision is determined by the server page size.
[10] For Sybase 12.5.1 and higher, precision is determined by the server page size.
[11] Supported only for Sybase 12.5.1 and higher.
[12] Supported only for Sybase 15.
[13] Supported only for Sybase 15.
[14] Supported only for Sybase 15.
[15] Supported only for Sybase 15.
[16] Supported only for Sybase 12.5 and higher.
[17] For Sybase 12.5.1 and higher, precision is determined by the server page size.
[18] For Sybase 12.5.1 and higher, precision is determined by the server page size.
[19] For Sybase 12.5.1 and higher, precision is determined by the server page size.
[20] For Sybase 12.5.1 and higher, precision is determined by the server page size.

C

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 the WebLogic Type 4 JDBC drivers, which replace the escape sequences with data store-specific grammar.

C.1 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:

Table C–1 Literal Types for Date, Time, and Timestamp Escape Sequences

literal-type	Description	Value Format
d	Date	yyyy-mm-dd
t	Time	hh:mm:ss [1]
ts	Timestamp	yyyy-mm-dd hh:mm:ss[.f...]

For example:

```
UPDATE Orders SET OpenDate={d '1995-01-15'}
WHERE OrderID=1023
```

C.2 Scalar Functions

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

```
{fn scalar-function}
```

where *scalar-function* is a scalar function supported by the WebLogic Type 4 JDBC drivers, as listed in Table C–2.

F or example:

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

Table C–2 Scalar Functions Supported

Data Store	String Functions	Numeric Functions	Timedate Functions	System Functions
DB2	ASCII	ABS or	CURDATE	COALESCE
	BLOB	ABSVAL	CURTIME	DEREF
	CHAR	ACOS	DATE	DLCOMMENT
	CHR	ASIN	DAY	DLLINKTYPE
	CLOB	ATAN	DAYNAME	DLURLCOMPLETE
	CONCAT	ATANH	DAYOFWEEK	DLURLPATH
	DBCLOB	ATAN2	DAYOFYEAR	DLURLPATHONLY
	DIFFERENCE	BIGINT	DAYS	DLURLSCHEME
	GRAPHIC	CEILING	HOUR	DLURLSERVER
	HEX	or CEIL	JULIAN_DAY	DLVALUE
	INSERT	COS	MICROSECOND	EVENT_MON_STATE
	LCASE or LOWER	COSH	MIDNIGHT_SECONDS	GENERATE_UNIQUE
	LCASE (SYSFUN schema)	COT	MINUTE	NODENUMBER
	LEFT	DECIMAL	MONTH	NULLIF
	LENGTH	DEGREES	MONTHNAME	PARTITION
	LOCATE	DIGITS	NOW	RAISE_ERROR
	LONG_VARCHAR	DOUBLE	QUARTER	TABLE_NAME
	LONG_VARGRAPHIC	EXP	SECOND	TABLE_SCHEMA
	LTRIM	FLOAT	TIME	TRANSLATE
	LTRIM (SYSFUN schema)	FLOOR	TIMESTAMP	TYPE_ID
	POSSTR	INTEGER	TIMESTAMP_ISO	TYPE_NAME
	REPEAT	LN	TIMESTAMPDIFF	TYPE_SCHEMA
	REPLACE	LOG	WEEK	VALUE
	RIGHT	LOG10	YEAR	
	RTRIM	MOD		
	RTRIM (SYSFUN schema)	POWER		
		RADIANS		
		RAND		
		REAL		

Table C–2 (Cont.) Scalar Functions Supported

Data Store	String Functions	Numeric Functions	Timedate Functions	System Functions
DB2	SOUNDEX	ROUND	N/A	N/A
(continued)	SPACE	SIGN		
	SUBSTR	SIN		
	TRUNCATE or TRUNC	SINH		
	UCASE or UPPER	SMALLINT		
	VARCHAR	SQRT		
	VARGRAPHIC	TAN		
		TANH		
		TRUNCATE		
Informix	CONCAT	ABS	CURDATE	DATABASE
	LEFT	ACOS	CURTIME	USER
	LENGTH	ASIN	DAYOFMONTH	
	LTRIM	ATAN	DAYOFWEEK	
	REPLACE	ATAN2	MONTH	
	RTRIM	COS	NOW	
	SUBSTRING	COT	TIMESTAMPADD	
		EXP	TIMESTAMPDIFF	
		FLOOR	YEAR	
		LOG		
		LOG10		
		MOD		
		PI		
		POWER		
		ROUND		
		SIN		
		SQRT		
		TAN		
		TRUNCATE		

Table C–2 (Cont.) Scalar Functions Supported

Data Store	String Functions	Numeric Functions	Timedate Functions	System Functions
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 C–2 (Cont.) Scalar Functions Supported

Data Store	String Functions	Numeric Functions	Timedate Functions	System Functions
Sybase	ASCII	ABS	DAYNAME	DATABASE
	CHAR	ACOS	DAYOFMONTH	IFNULL
	CONCAT	ASIN	DAYOFWEEK	USER
	DIFFERENCE	ATAN	DAYOFYEAR	
	INSERT	ATAN2	HOUR	
	LCASE	CEILING	MINUTE	
	LEFT	COS	MONTH	
	LENGTH	COT	MONTHNAME	
	LOCATE	DEGREES	NOW	
	LTRIM	EXP	QUARTER	
	REPEAT	FLOOR	SECOND	
	RIGHT	LOG	TIMESTAMPADD	
	RTRIM	LOG10	TIMESTAMPDIFF	
	SOUNDEX	MOD	WEEK	
	SPACE	PI	YEAR	
	SUBSTRING	POWER		
	UCASE	RADIANS		
		RAND		
		ROUND		
		SIGN		
		SIN		
		SQRT		
		TAN		

C.3 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.

For 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 C–3 lists the outer join escape sequences supported by WebLogic Type 4 JDBC drivers for each data store.

Table C–3 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
SQL Server	Left outer joins
	Right outer joins
	Full outer joins
	Nested outer joins
Sybase	Left outer joins
	Right outer joins
	Nested outer joins

C.4 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 '*'}
```

C.5 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.

D

Tracking JDBC Calls with WebLogic JDBC Spy

WebLogic JDBC Spy is a wrapper that wraps a WebLogic Type 4 JDBC driver. It logs detailed information about JDBC calls issued by an application and then passes the calls to the wrapped WebLogic Type 4 JDBC driver. You can use the information in the logs to help troubleshoot problems in your application. WebLogic JDBC Spy provides the following advantages:

- Logging is JDBC 4.0-compliant.
- Logging works with all WebLogic Type 4 JDBC drivers.
- Logging is consistent, regardless of which WebLogic Type 4 JDBC driver is used.
- All parameters and function results for JDBC calls can be logged.
- Logging can be enabled without changing the application, but instead by changing the JDBC data source in your WebLogic Server configuration.

> **Note:** The WebLogic JDBC Spy implements standard JDBC APIs only. It does not implement JDBC extensions implemented in other WebLogic Type 4 JDBC drivers. If your application uses JDBC extensions, you may see errors when using the WebLogic JDBC Spy.

D.1 Configuring WebLogic JDBC Data Sources for WebLogic JDBC Spy

To use WebLogic JDBC Spy with WebLogic Server, you add JDBC Spy attributes to the end of the URL in the JDBC data source configuration.

> **Note:** The `wlspy.jar` has been deprecated. This functionality is now built into the driver jar files and can be enabled by setting the SpyAttributes connection parameter.

Follow these instructions for modifying your data source configuration:

1. In the WebLogic Server Administration Console or in the configuration file for your WebLogic domain, append the WebLogic JDBC Spy options to the data source URL. Enclose all JDBC Spy options in one set of parentheses; separate multiple options with a semi-colon.

   In the Administration Console on the Domain Configurations > Data Sources, select the particular Data Source that you want to be spy enabled. Open the

Connection Pool tab and add the `spyAttributes` to the end of the existing URL. For example:

```
jdbc:weblogic:DB2://db2host:50000;spyAttributes=(log=(file)d:\spy.log;timestamp
=yes)
```

Alternatively, in the *datasource_name*`-jdbc.xml` file, update the URL in the JDBC data source entry. For example:

```
<jdbc-driver-params>
<url>jdbc:weblogic:db2://bangpcdb2:50000;spyAttributes=(log=(file)db2-spy.out;l
oad=weblogic.jdbc.db2.DB2Driver;timestamp=yes)
</url>
    <driver-name>weblogic.jdbc.db2.DB2Driver</driver-name>
    <properties>
      <property>
        <name>user</name>
        <value>john</value>
      </property>
      <property>
        <name>portNumber</name>
        <value>50000</value>
      </property>
      <property>
        <name>databaseName</name>
        <value>wls</value>
      </property>
      <property>
        <name>serverName</name>
        <value>db2host</value>
      </property>
      <property>
        <name>batchPerformanceWorkaround</name>
        <value>true</value>
      </property>
    </properties>
    <password-encrypted>{3DES}hqKps8ozo98=</password-encrypted>
  </jdbc-driver-params>
```

2. Stop and restart WebLogic Server.

D.2 WebLogic JDBC Spy URL Attributes

Table D–1 lists the options available for configuring WebLogic JDBC Spy. Use these options as attributes for the spyAttributes property for an XA driver or in the URL for a non-XA driver.

Table D–1 WebLogic JDBC Spy URL Attributes

Key-Value Pair	Description
log=System.out	Redirects logging to the Java output standard.
log=(file)*filename*	Redirects logging to the file specified by *filename*. By default, WebLogic JDBC Spy uses the stream specified in `DriverManager.setLogStream()`.
load=*classname*	Loads the driver specified by *classname*. For example, `weblogic.jdbc.db2.DB2Driver`.

Table D–1 (Cont.) WebLogic JDBC Spy URL Attributes

Key-Value Pair	Description
linelimit=*numberofchars*	The maximum number of characters, specified by *numberofchars*, that WebLogic JDBC Spy will log on one line. The default is 0 (no maximum limit).
logIS={`yes` or `no` or `nosingleread`}	Specifies whether WebLogic JDBC 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. This avoids the generation of large log files containing single-byte / single character read messages. The default is `no`.
logTName={`yes` or `no`}	Specifies whether WebLogic JDBC Spy logs the name of the current thread. The default is `no`.
timestamp={`yes` or `no`}	Specifies whether a timestamp should be included on each line of the WebLogic JDBC Spy log.

D.3 WebLogic JDBC Spy Log Example

See the notes following the example for the referenced text.

Example D–1 WebLogic JDBC Spy Log Example

```
All rights reserved.[1]
registerDriver:driver[className=weblogic.jdbcspy.SpyDriver,
context=null,weblogic.jdbcspy.SpyDriver@1ec49f][2]
*Driver.connect(jdbc:spy:{jdbc:weblogic:sqlserver://QANT:4003;
databaseName=Test;})
trying driver[className=weblogic.jdbcspy.SpyDriver,
context=null,weblogic.jdbcspy.SpyDriver@1ec49f][3]
spy>> Driver.connect(String url, Properties info)
spy>> url = jdbc:spy:{jdbc:weblogic:sqlserver://QANT:4003;databaseName=Test;
OSUser=qauser;OSPassword=null12}
spy>> info = {password=tiger, user=scott}
spy>> OK (Connection[1])[4]
getConnection returning driver[className=weblogic.jdbcspy.SpyDriver,
context=null,weblogic.jdbcspy.SpyDriver@1ec49f][5]
spy>> Connection[1].getWarnings()
spy>> OK[6]
spy>> Connection[1].createStatement
```

[1] The WebLogic JDBC Spy driver is registered. The spy>> prefix indicates that this line has been logged by WebLogic JDBC Spy.

[2] The JDBC Driver Manager logs a message each time a JDBC driver is registered.

[3] This is the logging of the JDBC Driver Manager. It logs a message each time a JDBC application makes a connection.

[4] The application connects with the specified URL. The User Name and Password are specified using properties.

[5] This is the logging of the JDBC Driver Manager. It logs a message each time a successful connection is made.

[6] The application checks to see if there are any warnings. In this example, no warnings are present.

```
spy>> OK (Statement[1])[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].getColumnLabel(int column)
spy>> column = 1
spy>> OK (EMPNO)[5]
spy>> ResultSetMetaData[1].getColumnLabel(int column)
spy>> column = 2
spy>> OK (ENAME)[6]
spy>> ResultSetMetaData[1].getColumnLabel(int column)
spy>> column = 3
spy>> OK (JOB)[7]
spy>> ResultSet[1].next()
spy>> OK (true)[8]
spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 1
spy>> OK (7369)[9]
spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 2
spy>> OK (SMITH)[10]
spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 3
spy>> OK (CLERK)[11]
spy>> ResultSet[1].next()
spy>> OK (false)[12]
spy>> ResultSet[1].close()
spy>> OK[13]
spy>> Connection[1].close()
spy>> OK[14]
```

[1] The statement "select empno,ename,job from emp where empno=7369" is created.

[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 fetched and its data retrieved.

[9] The first row is fetched and its data retrieved.

[10] The first row is fetched and its data retrieved.

[11] The first row is fetched and its data retrieved.

[12] The application attempts to fetch the second row, but the database returned only one row for this query.

[13] After fetching all data, the result set is closed.

[14] The application finishes and disconnects.