Oracle® WebLogic Server

Type 4 JDBC Drivers

10*g* Release 3 (10.3)

July 2008

ORACLE®

Oracle WebLogic Server Type 4 JDBC Drivers, 10*g* Release 3 (10.3)

Contents

6. The Oracle Driver (Deprecated)

7. The Sybase Driver

Introduction and Roadmap

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

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.

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 5, “The MS SQL Server Driver,” provides detailed information about the Microsoft SQL Server driver.
- Chapter 6, “The Oracle Driver (Deprecated),” provides detailed information about the Oracle driver.
- Chapter 7, “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.

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 WebLogic JDBC* is a guide to designing and using JDBC connections in your applications.
- *Configuring and Managing WebLogic JDBC* is a guide to JDBC configuration and management for WebLogic Server.

- *Developing Applications with WebLogic Server* is a guide to developing WebLogic Server applications.
- *Deploying Applications to WebLogic Server* is the primary source of information about deploying WebLogic Server applications.
- *WebLogic Server Performance and Tuning* contains information on monitoring and improving the performance of WebLogic Server applications.

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.

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.

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.

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 WebLogic Server" in *Release Notes*.

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:

- "JDBC Specification Compliance" on page 2-2
- "Installation" on page 2-2
- "Supported Databases" on page 2-3
- "Connecting Through WebLogic JDBC Data Sources" on page 2-4
- "Specifying Connection Properties" on page 2-4
- "Using IP Addresses" on page 2-5

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."

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 (e.g., `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:

- `wlbase.jar`
- `wlutil.jar`
- The DBMS-specific JAR file:

- For DB2: `wldb2.jar`
- For Informix: `wlinformix.jar`
- For MS SQL Server: `wlsqlserver.jar`
- For Oracle: `wloracle.jar` (Deprecated)
- For Sybase: `wlsybase.jar`

Supported Databases

Table 2-1 shows the databases supported by each of the WebLogic Type 4 JDBC drivers.

Table 2-1 Supported Databases

Driver	Supported Databases
DB2	• DB2 V9.1 for z/OS • DB2 V9.1 and V9.5 on Linux, UNIX, and Windows via DRDA • DB2 Universal Database (UDB) v7.*x*, v8.x on Linux, UNIX, and Windows via DRDA • DB2 UDB v7.x and v8.1 for z/OS via DRDA • DB2 UDB V5R1, V5R2, V5R3, and V5R4 for iSeries
Informix	Informix 9.2, 9.3, 9.4, 10, 11
Oracle (Deprecated. For more information, see Chapter 6, “The Oracle Driver (Deprecated).”	• Oracle 11g • Oracle 10*g* (R1 and R2) • Oracle 9i (R1 and R2)

Table 2-1 Supported Databases (Continued)

Driver	Supported Databases
SQL Server	• Microsoft SQL Server 2005 • Microsoft SQL Server 2000 • Microsoft SQL Server 2000 Desktop Engine (MSDE 2000) • SQL Server 2000 Enterprise Edition (64-bit) • Microsoft SQL Server 7.0
Sybase	• Sybase Adaptive Server Enterprise 15.0 • Sybase Adaptive Server Enterprise 12.0, 12.5, 12.5.1, 12.5.2, 12.5.3, and 12.5.4 • Sybase Adaptive Server 11.5 and 11.9

Note: Table 2-1 specifies the databases that are supported by the WebLogic Type 4 JDBC drivers, not the databases supported by WebLogic Server. For a list of databases supported by WebLogic Server, see *Supported Configurations*.

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 WebLogic JDBC*.
- For information about requesting a connection from a data source, see "Obtaining a Client Connection Using a DataSource" in *Programming WebLogic JDBC*.

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 "DB2 Connection Properties" on page 3-3.
- For the Informix driver, see "Informix Connection Properties" on page 4-3.
- For the MS SQL Server driver, see "SQL Server Connection Properties" on page 5-4.
- For the Oracle driver, see "Oracle Connection Properties" on page 6-3.
- For the Sybase driver, see "Sybase Connection Properties" on page 7-3.

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:

- "DB2 Connection Properties" on page 3-3
- "Informix Connection Properties" on page 4-3
- "SQL Server Connection Properties" on page 5-4
- "Oracle Connection Properties" on page 6-3
- "Sybase JTA Support" on page 7-34

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 2-2). 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 2-2 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
Oracle	All supported versions	Not supported.
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:bea: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:bea: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:bea: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:bea:db2://[0000:0000:0000:0000:0000:FFFF:123.456.78.90]:50000;
DatabaseName=jdbc;User=test;Password=secret
```

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

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

Using Security

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

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, Oracle (deprecated), 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 2-3 shows the authentication methods supported by the WebLogic Type 4 JDBC drivers.

Table 2-3 Authentication Methods Supported by the WebLogic Type 4 JDBC Drivers

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

1. Supported for Microsoft SQL Server 2000 and higher.

Kerberos Authentication Requirements

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

Table 2-4 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 **Oracle:** • Oracle 11g • Oracle 10g (R1 and R2) • Oracle 9i (R2) **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
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, Oracle, 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.

NTLM Authentication Requirements

Verify that your environment meets the requirements listed in Table 2-5 before you configure the driver for NTLM authentication.

Table 2-5 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: **Oracle:** • Oracle 11g • Oracle 10g (R1 and R2) • Oracle 9i (R1 and R2) **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

Table 2-5 NTLM Authentication Requirements for the Drivers (Continued)

Component	Requirements
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 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.

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“Data Encryption” on page 3-37 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 2-6 shows the data encryption methods supported by the WebLogic Type 4 JDBC drivers.

Table 2-6 Data Encryption Methods Supported by the WebLogic Type 4 JDBC Drivers

Driver	Database-Specific	SSL
DB2 for Linux/UNIX/Windows	X	
DB2 for z/OS	X	
DB2 for iSeries		X[1]
Informix		
Oracle		X
Microsoft SQL Server		X[2]
Sybase		X

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

SSL Encryption

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

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

  Note: SSL client authentication is supported with Oracle and DB2 only.

See the individual driver chapters for details about configuring SSL.

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.

SSL Client Authentication (Oracle and 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.

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 except for access to temporary files and access to `tnsnames.ora`. 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 WebLogic Security*.

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 `wlbase.jar` and `wlutil.jar` files as well as 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:

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

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

And one or more of the following:

```
//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 Oracle:
grant codeBase "file:WL_HOME${/}server${/}lib${/}wloracle.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";
};
```

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 "Permissions for Establishing Connections" on page 2-16.

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 "Permissions for Establishing Connections" on page 2-16.

Granting Access to Oracle tnsnames.ora Files

If you are using an Oracle tnsnames.ora file to connect with the WebLogic Type 4 Oracle driver, read access to the tnsnames.ora file must be granted to the driver in the security policy file of the Java 2 Platform.

```
   grant codeBase "file:WL_HOME${/}server${/}lib${/}-" {
      permission java.io.FilePermission "C:\\oracle\\ora92\\network\\admin
\\
         tnsnames.ora", "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 "Permissions for Establishing Connections" on page 2-16.

See "Performance Considerations" on page 6-22 for more information about using tnsnames.ora files to connect to Oracle databases.

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.

DB2

```
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.

Oracle (Deprecated)

```
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 Kerberos Key Distribution Center (KDC) that identifies the database service.
- *db_hostname* is the host name of the machine running the database.

Microsoft SQL Server

```
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.

Sybase

```
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.

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 "JTA Support" on page 3-44.
- For Microsoft SQL Server, see "Installing Stored Procedures for JTA" on page 5-48.
- For Oracle, see "Batch Inserts and Updates" on page 6-50.

- For Sybase, see "Sybase JTA Support" on page 7-34

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.

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

Driver Errors

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

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

For example:

```
[BEA][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.

Database Errors

An error generated by the database has the following format:

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

For example:

```
[BEA][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.

The DB2 Driver

The following sections describe how to configure and use the WebLogic Type 4 JDBC driver for DB2:

Database Version Support

The WebLogic Type 4 JDBC driver for DB2 (the "DB2 driver") supports:

- DB2 V9.1 and V9.5 for Linux, UNIX, and Windows via DRDA
- DB2 Universal Database (UDB) v7.x and 8.x on Linux, UNIX, and Windows via DRDA
- DB2 UDB v7.x and v8.1 for z/OS via DRDA
- DB2 UDB V5R1, V5R2, V5R3, and V5R4 for iSeries via DRDA

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

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

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.

DB2 URL

The connection URL format for the DB2 driver is:

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

where:

- *`hostname`* is the IP address or TCP/IP host name of the server to which you are connecting. See "Using IP Addresses" on page 2-5 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 "DB2 Connection Properties" on page 3-3.

For example:

DB2 UDB for Linux, UNIX, and Windows

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

DB2 UDB for z/OS and iSeries

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

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
AddToCreateTable OPTIONAL	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.
AllowImplicitResultSetCloseForXA OPTIONAL	{true | 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 OPTIONAL	Sets the default DB2 schema used by unqualified SQL identifiers to the specified value. The value must be a valid DB2 schema.

Table 3-1 DB2 Connection Properties (Continued)

Property	Description
AuthenticationMethod	{kerberos | encryptedUIDPassword | encryptedPassword | clearText | client}. Determines which authentication method the driver uses when establishing a connection. If set to kerberos, the driver uses Kerberos authentication. The driver ignores any user ID or password specified. If set to encryptedUIDPassword, the driver uses user ID/password authentication. The driver sends an encrypted user ID and password to the DB2 server for authentication. If a user ID and password are not specified, the driver throws an exception. If this value is set, the driver can also use data encryption (see the `EncryptionMethod` property for details). If set to encryptedPassword, the driver uses user ID/password authentication. The driver sends a user ID in clear text and an encrypted password to the DB2 server for authentication. If a user ID and password are not specified, the driver throws an exception. If this value is set, the driver can also use data encryption (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 set to client, the driver uses client authentication. The DB2 server relies on the client to authenticate the user and does not provide additional authentication. The driver ignores any user ID or password specified. 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"Authentication" on page 3-30 for more information about using authentication with the DB2 driver.

Table 3-1 DB2 Connection Properties (Continued)

Property	Description
BatchPerformanceWorkaround OPTIONAL	`{true | false}`.The DB2 driver uses the native DB2 batch mechanism. This property determines whether certain restrictions are enforced to facilitate data conversions. • 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`, this restriction is removed; however, parameter sets may not be executed in the order they were specified. The default is `false`. See "Batch Inserts and Updates" on page 3-45 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.
CatalogIncludesSynonyms OPTIONAL	`{true | false}`. Determines whether synonyms are included in the result sets returned from the DatabaseMetaData.getColumns() method. 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. This property is ignored for DB2 v8.x and higher for Linux/UNIX/Windows. The driver always returns synonyms for the DatabaseMetaData.getColumns() method when connected to DB2 v8.x and higher for Linux/UNIX/Windows. See "Performance Considerations" on page 3-19 for information about configuring this property for optimal performance. The default is `true`.

Table 3-1 DB2 Connection Properties (Continued)

Property	Description
CatalogSchema OPTIONAL	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. The default is SYSCAT (DB2 for Linux/UNIX/Windows), SYSIBM (DB2 for z/OS), or QSYS2 (DB2 for iSeries) To improve performance, views of system catalog tables can be created in a schema other than the default catalog schema. Setting this property to a schema that contains views of the catalog tables allows the driver to use those views. To ensure that catalog methods function correctly, views for specific catalog tables must exist in the specified schema. The views that are required depend on your DB2 database. See "Non-Default Schemas for Catalog Methods" on page 3-42 for the required views of catalog tables. See "Performance Considerations" on page 3-19 for information about configuring this property for optimal performance.
CharsetFor65535 OPTIONAL	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.
CodePageOverride OPTIONAL	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. 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.

Table 3-1 DB2 Connection Properties (Continued)

Property	Description
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. See "Creating a DB2 Package" on page 3-22 for more information about creating DB2 packages.
ConnectionRetryCount OPTIONAL	The number of times the driver retries connection attempts until a successful connection is established. Valid values are 0 and any positive integer. If set to 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 an application sets a login timeout value (for example, using DataSource.loginTimeout or DriverManager.loginTimeout), the login timeout takes precedence over this property. For example, if the login timeout expires, any connection attempts stop. The `ConnectionRetryDelay` property specifies the wait interval, in seconds, used between retry attempts. The default is 5.
ConnectionRetryDelay OPTIONAL	The number of seconds the driver waits between connection retry attempts when ConnectionRetryCount is set to a positive integer. The default is 1.
ConvertNull	{1 | 0}. Controls how data conversions are handled for null values. If set to 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. 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. The default is 1.

Table 3-1 DB2 Connection Properties (Continued)

Property	Description
CreateDefaultPackage OPTIONAL	{true | false}. Determines whether the driver automatically creates required DB2 packages. If set to true, the DB2 driver automatically creates required DB2 packages, even if they already exist. Existing DB2 packages are replaced by the new packages. If set to false (the 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 "Creating a DB2 Package" on page 3-22. The default is false.
DatabaseName	The name of the database to which you want to connect (used with UDB). **Note:** This property is supported only for DB2 for Linux/UNIX/Windows. See also "Database Connection Property" on page 3-50.
DynamicSections OPTIONAL	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 DB2 Connection Properties (Continued)

Property	Description
EnableCancelTimeout	{true | false}. 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. If set to true, the cancel request times out using the same timeout value, in seconds, that is set for the statement it cancels. For example, if your application calls `Statement.setQueryTimeout(5)` on a statement and that statement is cancelled because its timeout value was exceeded, the driver sends a cancel request that also will time out if its execution exceeds 5 seconds. If the cancel request times out, because the server is down, for example, the driver throws an exception indicating that the cancel request was timed out and the connection is no longer valid. If set to false (the default), the cancel request does not time out.

Table 3-1 DB2 Connection Properties (Continued)

Property	Description
EncryptionMethod	{noEncryption | DBEncryption | requestDBEncryption}. **Note:** The DB2 driver now supports SSL encryption for DB2 V5R3 and higher for iSeries. See "Data Encryption" on page 3-37 for more information about using encryption with the DB2 driver. Determines whether a DB2-specific encryption algorithm is used to encrypt and decrypt data transmitted over the network between the driver and database server. To use encryption, you also must set the `AuthenticationMethod` property to a value of clearText, encryptedPassword, or encryptedUIDPassword. If set to noEncryption (the default), data is not encrypted or decrypted. If set to DBEncryption, data is encrypted using DES encryption if the database server supports it. If the database server does not support DES encryption, the connection fails and the driver throws an exception. If set to requestDBEncryption, data is encrypted using DES encryption if the database server supports it. If the database server does not support DES encryption, the driver attempts to establish an unencrypted connection. See "Performance Considerations" on page 3-19 for information about configuring this property for optimal performance. The default is noEncryption.
Grantee OPTIONAL	Specifies the name of the schema to which you want to grant EXECUTE privileges for DB2 packages. The value must be a valid DB2 schema.This property is ignored if the `GrantExecute` property is set to false. 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. See "Creating a DB2 Package" on page 3-22 for more information about creating DB2 packages. The default is PUBLIC.

Table 3-1 DB2 Connection Properties (Continued)

Property	Description
GrantExecute OPTIONAL	{true | false}. Determines which DB2 schema is granted EXECUTE privileges for DB2 packages. If set to true (the default), EXECUTE privileges are granted to the schema specified by the `Grantee` property. If set to false, EXECUTE privileges are granted to the schema that created the DB2 packages. See "Creating a DB2 Package" on page 3-22 for more information about creating DB2 packages. The default is true.

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, suppose USER1 needs to invoke stored procedures owned by USER2 without specifying the qualified name for those procedures. You can use this property to add USER2 to the CURRENT PATH special register, which sets the default schema, or schemas, to use when executing a user-defined function or stored procedure.
	`jdbc:bea:db2://server1:50000;`
	`InitializationString=SET CURRENT`
	`PATH=current_path, USER2`
	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. The following connection URL adds USER2 to the CURRENT PATH special register and sets the CURRENT PRECISION special register to DEC31.
	`jdbc:bea: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.
	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.

Table 3-1 DB2 Connection Properties (Continued)

Property	Description
InsensitiveResultSetBufferSize	{-1 | 0 | *x*}. Determines the amount of memory used by the driver to cache insensitive result set data. It must have one of the following values: 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 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	{true | 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. 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 DB2 Connection Properties (Continued)

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.
	`WRKRDBDIRE`
	This property is supported only for DB2 for z/OS and DB2 for iSeries.
	See also "DatabaseName Connection Property" on page 3-50.

LoginTimeout OPTIONAL	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. If set to 0 (the default), the driver does not time out a connection request.
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 "Creating a DB2 Package" on page 3-22 for more information about creating DB2 packages. The default is NULLID.
PackageOwner OPTIONAL	The owner to be used for any DB2 packages that are created. See "Creating a DB2 Package" on page 3-22 for more information about creating DB2 packages. The default is NULL.

Table 3-1 DB2 Connection Properties (Continued)

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.
PortNumber OPTIONAL	The TCP port on which the database server listens for connections. The default is 50000.
QueryTimeout	{*positive integer* | -1 | 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.
ReplacePackage OPTIONAL	{true | false}.Determines whether the current bind process will replace the existing DB2 packages used by the driver. If set to true, the current bind process will replace the existing DB2 packages used by the driver. If set to false (the default), the current bind process will not replace the existing DB2 packages. For DB2 for Linux/UNIX/Windows, this property must be used in conjunction with the CreateDefaultPackage property. For more information about creating DB2 packages, see "Creating a DB2 Package" on page 3-22. The default is false.

Table 3-1 DB2 Connection Properties (Continued)

Property	Description
ResultSetMetaDataOptions	{0 | 1}. The DB2 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 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 "Performance Considerations" on page 3-19. The default is 0.
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 DB2 Connection Properties (Continued)

Property	Description
SendStreamAsBlob OPTIONAL	{true | false}. 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. If set to 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 set to 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 "Performance Considerations" on page 3-19 for information about configuring this property for optimal performance. The default is false.
ServerName	Specifies either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the database server. For example, 122.23.15.12 or DB2Server. This property is supported only for data source connections.

Table 3-1 DB2 Connection Properties (Continued)

Property	Description
StripNewlines OPTIONAL	{true | false}. 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. Is set to true, the DB2 driver removes all new-line characters from SQL statements. If set to false (the default), the driver does not remove any newline characters from SQL statements. See "Performance Considerations" on page 3-19 for information about configuring this property for optimal performance. The default is false.
UseCurrentSchema OPTIONAL	{true | false}. 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. If set to true, results that are returned from the `getTables()` and `getColumns()` methods are restricted to tables and views in the current schema. If set to false (the default), results of the `getTables()` and `getColumns()` methods are not restricted. See "Performance Considerations" on page 3-19 for information about configuring this property for optimal performance. The default is false.
User	The case-sensitive user name used to connect to the DB2 database.

Table 3-1 DB2 Connection Properties (Continued)

Property	Description
WithHoldCursors OPTIONAL	{true | false}. 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. If set to true (the default), the cursor behavior is Preserve. If set to false, the cursor behavior is Delete. The default is true.
XMLDescribeType	{clob | blob}. Determines whether the driver maps XML data to the CLOB or BLOB data type. If set to clob (the default), the driver maps XML data to the CLOB data type. If set to blob, the driver maps XML data to the BLOB data type. See "Returning and Inserting/Updating XML Data" on page 3-27 for more information. The default is clob.

Performance Considerations

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

- "CatalogIncludesSynonyms" on page 3-20
- "CatalogSchema" on page 3-20
- "EncryptionMethod" on page 3-20
- "InsensitiveResultSetBufferSize" on page 3-20
- "ResultSetMetaDataOptions" on page 3-20
- "SendStreamAsBlob" on page 3-21
- "StripNewLines" on page 3-21
- "UseCurrentSchema" on page 3-21

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.

CatalogSchema

To improve performance, views of system catalog tables can be created in a catalog schema other than the default. The DB2 driver can access the views of catalog tables if this property is set to the name of the schema containing the views. The default catalog schema is SYSCAT for DB2 for Linux/UNIX/Windows, SYSIBM for DB2 for z/OS, and QSYS2 for DB2 for iSeries.

To ensure that catalog methods function correctly, views for specific catalog tables must exist in the specified schema. The views that are required depend on your DB2 database. See "Non-Default Schemas for Catalog Methods" on page 3-42 for views for catalog tables that must exist in the specified schema.

EncryptionMethod

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

InsensitiveResultSetBufferSize

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

ResultSetMetaDataOptions

By default, the DB2 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 "ResultSet Metadata Support" on page 3-48 for more information about returning ResultSet metadata.

SendStreamAsBlob

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

StripNewLines

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

UseCurrentSchema

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

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 "DB2 Connection Properties" on page 3-3).

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

   ```
   jdbc:bea: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").newInst
ance();
   conn = myDriver.connect("jdbc:bea:db2://10.1.4.1:446;LocationName=S10B7
57B", props);
   stmt = conn.createStatement();
   stmt.execute("select * from MYDATABASE.MYTABLE");
   rs = stmt.getResultSet();
   ...
```

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.

Notes: The initial connection may take a few minutes because of the number and size of the packages that must be created for the connection. Subsequent connections do not incur this delay.

When the driver has completed creating packages, it writes the following message to the standard output: DB2 packages created.

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 “Creating a DB2 Package Using dbping” on page 3-23.

- Automatically create a package by setting specific connection properties in the connection URL or data source. See "Creating a DB2 Package Using Connection Properties" on page 3-23.

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.

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.

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`.

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

Example for DB2 for z/OS and iSeries:

The following URL creates DB2 packages with 400 dynamic sections.

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

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.

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
Blob[2]	BLOB
Char	CHAR
Char for Bit Data	BINARY
Clob	CLOB
Date	DATE
DBClob[3]	CLOB
Decimal	DECIMAL
Double	DOUBLE
Float	FLOAT
Integer	INTEGER
Long Varchar	LONGVARCHAR
Long Varchar for Bit Data	LONGVARBINARY
Numeric	NUMERIC
Real	REAL
Rowid[4]	VARBINARY
Smallint	SMALLINT
Time	TIME
Timestamp	TIMESTAMP

Table 3-3 DB2 Data Types (Continued)

DB2 Data Type	JDBC Data Type
Varchar	VARCHAR
Varchar for Bit Data	VARBINARY

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, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries (see "Large Object (LOB) Support" on page 3-44).
3. 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 "Large Object (LOB) Support" on page 3-44).
4. Supported only for DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries.

See "Large Object (LOB) Support" on page 3-44 for more information about the Blob, Clob, and DBClob data types. See "Returning and Inserting/Updating XML Data" on page 3-27 for more information about the XML data type.See "GetTypeInfo" on page B-1 for more information about data types.

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.

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

Character Data

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

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

```
ResultSet.getString()
ResultSet.getCharacterStream()
ResultSet.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.

Binary Data

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

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

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

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

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

Inserting/Updating XML Data

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

Character Data

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

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

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

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

```
PreparedStatement.setAsciiStream()
ResultSet.updateAsciiStream()
```

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

Binary Data

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

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

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

Authentication

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See “Authentication” on page 2-7 for an overview.

The DB2 driver supports the following methods of authentication:

- User ID/password authentication authenticates the user to the database using a database user name and password. Depending on the method you specify, the driver passes one of the following sets of credentials to the DB2 database server for authentication:
  - Encrypted user ID and password
  - User ID in clear text and an encrypted password
  - Both user ID and password in clear text
- Kerberos authentication uses Kerberos, a trusted third-party authentication service, to verify user identities. Kerberos authentication can take advantage of the user name and

password maintained by the operating system to authenticate users to the database or use another set of user credentials specified by the application.

This method requires knowledge of how to configure your Kerberos environment and supports Windows Active Directory Kerberos and MIT Kerberos.

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

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

The driver's `AuthenticationMethod` connection property controls which authentication mechanism the driver uses when establishing connections. See"Using the AuthenticationMethod Property" on page 3-31 for information about setting the value for this property.

Using the AuthenticationMethod Property

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

When `AuthenticationMethod=kerberos`, the driver uses Kerberos authentication when establishing a connection. The driver ignores any values specified by the User property and Password properties.

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

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

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

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

Configuring User ID/Password Authentication

J2SE 1.4 or higher is required to use encrypted user ID and password authentication.

To configure user ID/password authentication:

1. Set the `AuthenticationMethod` property to encryptedUIDPassword, encryptedPassword, or clearText (the default). See "Using the AuthenticationMethod Property" on page 3-31 for more information about setting a value for this property.
2. Set the `User` property to provide the user ID.
3. Set the `Password` property to provide the password.

Configuring Kerberos Authentication

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

Product Requirements

Verify that your environment meets the requirements listed in Table 3-4 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.

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 "Using the AuthenticationMethod Property" on page 3-31 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:[BEA][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 "Permissions for Kerberos Authentication" on page 2-19 for an example.

Specifying User Credentials for Kerberos Authentication (Delegation of Credentials)

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

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:bea:db2://myServer:50000;
            DatabaseName=jdbc";
         con = DriverManager.getConnection(url);
        }
     catch (Exception except) {

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

        return con;
    }
});

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

... // do something with the results
```

Obtaining a Kerberos Ticket Granting Ticket

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

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

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

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

where *user* is the application user.

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

Configuring Client Authentication

Set the `AuthenticationMethod` property to client. See "Using the AuthenticationMethod Property" on page 3-31 for more information about setting a value for this property.

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 "SSL Encryption" on page 2-13 for more information.

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

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.

SSL Connection Properties

The EncryptionMethod connection property supports a new value, SSL, that enables SSL encryption. New connection properties that control how the driver implements SSL encryption are:

- HostNameInCertificate
- KeyPassword
- KeyStore
- KeyStorePassword
- TrustStore
- TrustStorePassword
- ValidateServerCertificate

Table 3-5 describes these connection properties.

Table 3-5 SSL Connection Properties (DB2 Driver)

Property	Description

`EncryptionMethod`
Default: noEncryption
Data type: String
	`{noEncryption | DBEncryption | requestDBEncryption | SSL}`. Determines whether data is encrypted and decrypted when transmitted over the network between the driver and database server. The DB2 driver supports the following encryption methods:
	• DB2-specific encryption (DB2 for Linux/UNIX/Windows and DB2 for z/OS)
	• SSL (DB2 V5R3 and higher for iSeries)
	If set to `noEncryption` (the default), data is not encrypted or decrypted.
	If set to `DBEncryption`, data is encrypted using DES encryption if the database server supports it. If the database server does not support DES encryption, the connection fails and the driver throws an exception. This value is supported for DB2 for Linux/UNIX/Windows and DB2 for z/OS.
	If set to `requestDBEncryption`, data is encrypted using DES encryption if the database server supports it. If the database server does not support DES encryption, the driver attempts to establish an unencrypted connection. This value is supported for DB2 for Linux/UNIX/Windows and DB2 for z/OS.
	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. This value is supported for DB2 V5R3 and higher for iSeries.
	When SSL is enabled, the following properties also apply:
	`HostNameInCertificate`
	`KeyStore` (for SSL client authentication)
	`KeyStorePassword` (for SSL client authentication)
	`KeyPassword` (for SSL client authentication)
	`TrustStore`
	`TrustStorePassword`
	`ValidateServerCertificate`
	Note: Connection hangs can occur when the driver is configured for SSL and the database server does not support SSL. You may want to set a login timeout using the LoginTimeout property to avoid problems when connecting to a server that does not support SSL.
	See "Configuring SSL Encryption" on page 3-37 for more information about configuring data encryption.

Table 3-5 SSL Connection Properties (DB2 Driver) (Continued)

Property	Description
`HostNameInCertificate` Default: Empty String Data type: String	{*`host_name`* | `#SERVERNAME#`}. 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. 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 in the `SubjectAlternativeName` or if the certificate does not have a `SubjectAlternativeName`, the driver compares the host name with the Common Name (CN) part of the certificate's Subject name. If the values do not match, the connection fails and the driver throws an exception. If `#SERVERNAME#` is specified, the driver compares the server name specified in the connection URL or data source of the connection to the DNSName value of the `SubjectAlternativeName` in the certificate. If a DNSName value does not exist in the `SubjectAlternativeName` or if the certificate does not have a SubjectAlternativeName, the driver compares the host name to the CN 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 uses the server name specified in the connection URL or data source of the connection to validate the certificate. If SSL encryption or certificate validation is not enabled, this property is ignored.
`KeyPassword` Default: No default Data type: String	Specifies the password used to access the individual keys in the keystore file when SSL is enabled using the EncryptionMethod property and SSL client authentication is enabled on the database server. This property is useful if any of the keys in the keystore file have a different password than the keystore file.
`KeyStore` Default: No default Data type: String	Specifies the directory of the keystore file to be used when SSL is enabled using the `EncryptionMethod` property 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 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.

Table 3-5 SSL Connection Properties (DB2 Driver) (Continued)

Property	Description
`KeyStorePassword` Default: No default Data type: String	Specifies the password used to access the keystore file when SSL is enabled using the `EncryptionMethod` property 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 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.
`TrustStore` Default: No default Data type: String	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 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`.
`TrustStorePassword` Default: No default Data type: String	Specifies the password 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 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`.

Table 3-5 SSL Connection Properties (DB2 Driver) (Continued)

Property	Description
`ValidateServerCertificate` Default: `true` Data type: boolean	{`true` | `false`}. 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. If set to `true` (the default), 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. If set to false, 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. Truststore information is specified using the `TrustStore` and `TrustStorePassword` properties or by using Java system properties.

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 3-6 must exist in the specified schema. The views that are required depend on your DB2 database.

Table 3-6 Catalog Tables for DB2

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

SQL Escape Sequences

See "SQL Escape Sequences for JDBC" on page C-1 for information about SQL escape sequences supported by the DB2 driver.

Isolation Levels

The DB2 driver supports the isolation levels listed in Table 3-7. JDBC isolation levels are mapped to the appropriate DB2 transaction isolation levels as shown. The default isolation level is `Read Committed`.

Table 3-7 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.

Using Scrollable Cursors

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

Note: When the DB2 driver cannot support the requested result set type or concurrency, it automatically downgrades the cursor and generates one or more SQLWarnings with detailed information.

JTA Support

To use 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.

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

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.

Notes: 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.

Parameter Metadata Support

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

Insert and Update Statements

The DB2 driver supports returning parameter metadata for all types of SQL statements with the following 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 <>.

Select Statements

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

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

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

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

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

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

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

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

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

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

```
SELECT col1, col2 FROM foo WHERE col1 = ? and col2 > ?
SELECT ... WHERE colname = (SELECT col2 FROM t2
   WHERE col3 = ?)
SELECT ... WHERE colname LIKE ?
SELECT ... WHERE colname BETWEEN ? and ?
SELECT ... WHERE colname IN (?, ?, ?)
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE col1 < ?)
```

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

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

Joins are supported. For example:

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

Fully qualified names and aliases are supported. For example:

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

Stored Procedures

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

ResultSet Metadata Support

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

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.

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.

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:bea:db2://server1:50000;DatabaseName=jdbc;Database=acct;
User=test;Password=secret
```

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.

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 3-8 and Table 3-9 list these data types and describe how they are mapped to JDBC data types.

Table 3-8 DB2 Graphic Data Types

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

Table 3-9 New DB2 Data Types Supported for DB2 v9.1 for z/OS

DB2 Data Type	JDBC Data Type
Bigint	BIGINT
Binary	BINARY
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 "Returning and Inserting/Updating XML Data" on page 3-27.

For information about other data types supported by the DB2 driver, see "Data Types" on page 3-26.

SQL Procedures for z/OS

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

IPv6 Support

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

For more information about IPv6, see "Using IP Addresses" on page 2-5.

CHAPTER 4

The Informix Driver

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

- "FILETOBLOB Feature Support" on page 4-17
- "Auto-Generated Keys Support" on page 4-17

Informix Database Version Support

The WebLogic Type 4 JDBC Informix driver (the "Informix driver") supports the following databases:

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

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.

Informix URL

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

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

where:

- *`hostname`* is the TCP/IP address or TCP/IP host name of the server to which you are connecting. See "Using IP Addresses" on page 2-5 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 "Informix Connection Properties" on page 4-3.

For example:

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

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
CodePageOverride OPTIONAL	The code page the driver uses when converting character data. The specified code page overrides the default database code page or column collation. All Character data returned from or written to the database is converted using the specified code page.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 OPTIONAL	The number of times the driver retries connections to the database server until a successful connection is established. Valid values are 0 and any positive integer. If set to 0, the driver does not try to reconnect after the initial unsuccessful attempt. If a connection is not successfully established on the driver's first pass through the list of database servers, the driver retries all the servers in the list only once. If an application sets a login timeout value (for example, using DataSource.loginTimeout or DriverManager.loginTimeout), the login timeout takes precedence over this property. For example, if the login timeout expires, any connection attempts to alternate servers stop. The `ConnectionRetryDelay` property specifies the wait interval, in seconds, used between retry attempts The default is 5.
ConnectionRetryDelay OPTIONAL	The number of seconds the driver waits between connection retry attempts when ConnectionRetryCount is set to a positive integer. The default is 1.

Table 4-1 Informix Connection String Properties (Continued)

Property	Description
ConvertNull	{1 | 0}. Controls how data conversions are handled for null values. If set to 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. 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. The default is 1.
DatabaseName OPTIONAL	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.

Table 4-1 Informix Connection String Properties (Continued)

Property	Description
DBDate OPTIONAL	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 Valid values are: DMY2 Y4DM DMY4 Y4MD MDY2 Y2DM MDY4 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: Hyphen (-) Period (.) 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 Informix Connection String Properties (Continued)

Property	Description
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 "Performance Considerations" on page 4-11 for information about configuring this property for optimal performance. The default is 32767
InformixServer REQUIRED	The name of the Informix database server to which you want to connect.

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:bea: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.

Table 4-1 Informix Connection String Properties (Continued)

Property	Description
InsensitiveResultSetBufferSize OPTIONAL	{-1 | 0 | *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. The default is 2048 (KB)
JavaDoubleToString	{true | 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.
LoginTimeout OPTIONAL	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. If set to 0 (the default), the driver does not time out a connection request.

Table 4-1 Informix Connection String Properties (Continued)

Property	Description
Password REQUIRED	A case-insensitive password used to connect to your Informix database. A password is required only if security is enabled on your database. If so, contact your system administrator to obtain your password.
PortNumber REQUIRED	The TCP port on which the database server listens for connections. The default varies depending on operating system.The default varies depending on operating system. This property is supported only for data source connections.
QueryTimeout	{*positive integer* | -1 | 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	{0 | 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 "Performance Considerations" on page 4-11 for information about configuring this property for optimal performance. The default is 0.

Table 4-1 Informix Connection String Properties (Continued)

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.
UseDelimitedIdentifier	{true | false} Controls how the Informix server interprets double quote (") characters in SQL statements. 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.

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 the *Configuring and Managing WebLogic JDBC*.

Performance Considerations

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

- "FetchBufferSize" on page 4-11
- "InsensitiveResultSetBufferSize" on page 4-11

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.

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.

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.

See "ResultSet MetaData Support" on page 4-15 for more information about returning ResultSet metadata.

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

Table 4-2 Informix Data Types (Continued)

Informix Data Type	JDBC Data Type
NVARCHAR	VARCHAR
SERIAL	INTEGER
SERIAL8	BIGINT
SMALLFLOAT	REAL
SMALLINT	SMALLINT
TEXT	LONGVARCHAR
VARCHAR	VARCHAR

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

SQL Escape Sequences

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

Isolation Levels

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

Using Scrollable Cursors

The Informix driver supports scroll-sensitive result sets, scroll-insensitive result sets, and updatable result sets.

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

Parameter Metadata Support

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

Insert and Update Statements

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

Select Statements

The Informix driver supports returning parameter metadata for Select statements that contain parameters in ANSI SQL 92 entry-level predicates, for example, such as COMPARISON, BETWEEN, IN, LIKE, and EXISTS predicate constructs. Refer to the ANSI SQL reference for detailed syntax.

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

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

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

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

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

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

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

```
SELECT col1, col2 FROM foo WHERE col1 = ? and col2 > ?
SELECT ... WHERE colname = (SELECT col2 FROM t2 WHERE col3 = ?)
SELECT ... WHERE colname LIKE ?
SELECT ... WHERE colname BETWEEN ? and ?
SELECT ... WHERE colname IN (?, ?, ?)
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE col1 < ?)
```

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

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

Joins are supported. For example:

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

Fully qualified names and aliases are supported. For example:

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

When parameter metadata is requested for a column defined as an approximate numeric data type, the driver returns a scale of 255, which indicates the column has an approximate numeric data type and has no scale. For example, suppose we create a table where col2 is an approximate numeric data type with a precision of 20:

```
CREATE table fooTest(col1 int, col2 decimal(20))
```

The driver returns parameter metadata that indicates that col2 has a data type of decimal, a precision of 20, and a scale of 255.

Stored Procedures

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

ResultSet MetaData Support

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

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

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

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

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

```
SELECT E.id, E.name as EmployeeName, {fn UCASE(E.name)}

   AS upper FROM Employee E
```

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

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

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

Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

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.

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.

FILETOBLOB Feature Support

When converting a file to a Blob using the FILETOBLOB feature with the `SERVER` keyword and a file that exists on the server, the conversion works properly with a command similar to the following:

```
   st.executeUpdate("INSERT INTO doc_list VALUES (7, FILETOBLOB('c:\\temp\
\INSTSRV.EXE', 'SERVER'))");
```

You cannot use the `FILETOBLOB` function with the `CLIENT` keyword because the function relies on the Informix client software to handle the data transfer from the client side to the server side. With the WebLogic JDBC Driver for Informix, there is no underlying client software so there is no current implementation to handle this type of data transfer.

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.

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:bea:informix://server1:2003;InformixServer=ol_test;
DatabaseName=jdbc;Database=acct;User=test;Password=secret
```

CHAPTER 5

The MS SQL Server Driver

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

- "Using Scrollable Cursors" on page 5-47
- "Server-Side Updatable Cursors" on page 5-47
- "Installing Stored Procedures for JTA" on page 5-48
- "Distributed Transaction Cleanup" on page 5-49
- "Large Object (LOB) Support" on page 5-50
- "Batch Inserts and Updates" on page 5-51
- "Parameter Metadata Support" on page 5-51
- "ResultSet MetaData Support" on page 5-53
- "Rowset Support" on page 5-54
- "Auto-Generated Keys Support" on page 5-54
- "Null Values" on page 5-55
- "Database Connection Property" on page 5-56

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.

SQL Server Database Version Support

The WebLogic Type 4 JDBC MS SQL Server driver (the "SQL Server driver") supports the following database management system versions:

- Microsoft SQL Server 2005
- Microsoft SQL Server 2000
- Microsoft SQL Server 2000 Desktop Engine (MSDE 2000)
- Microsoft SQL Server 2000 Enterprise Edition (64-bit)
- Microsoft SQL Server 7.0

To use JDBC distributed transactions through JTA, you must install stored procedures for Microsoft SQL Server. See "Installing Stored Procedures for JTA" on page 5-48 for details.

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
```

Microsoft SQL Server URL

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

```
jdbc:bea:sqlserver://hostname:port[;property=value[;...]]
```

where:

- `hostname` is the TCP/IP address or TCP/IP host name of the server to which you are connecting. See "Using IP Addresses" on page 2-5 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 "SQL Server Connection Properties" on page 5-4.

For example:

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

See "Connecting to Named Instances" on page 5-3 for instructions on connecting to named instances.

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:bea: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:bea:sqlserver://server1\\instance1;User=test;Pasword=secret
```

SQL Server Connection Properties

Table 5-1 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 5-1 SQL Server Connection Properties

Property	Description
AlwaysReportTriggerResults OPTIONAL	{true | false}. 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. If set to 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 set to 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.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
AuthenticationMethod	{auto | kerberos | ntlm | userIdPassword}. Determines which authentication method the driver uses when establishing a connection. If set to auto (the default), 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 "Using the AuthenticationMethod Property" on page 5-34 for more information about using the default value. If set to kerberos, the driver uses Kerberos authentication. The driver ignores any user ID or password specified. This value is supported only when connecting to Microsoft SQL Server 2000 or higher. If set to ntlm, the driver uses NTLM authentication if the DLL required for NTLM authentication can be loaded. If the driver cannot load the DLL, the driver throws an exception. The driver ignores any user ID or password specified. If set to userIdPassword, the driver uses SQL Server authentication when establishing a connection. If a user ID is not specified, the driver throws an exception. 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 "Authentication" on page 5-33 for more information about using authentication with the SQL Server driver.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
CodePageOverride OPTIONAL	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 SendStringParametersAsUnicode property and generates a warning. The driver always sends parameters using the code page specified by `CodePageOverride` if this property is specified.
ConnectionRetryCount OPTIONAL	The number of times the driver retries connections to a database server until a successful connection is established. Valid values are 0 and any positive integer. If set to 0, the driver does not try to reconnect after the initial unsuccessful attempt. The `ConnectionRetryDelay` property specifies the wait interval, in seconds, used between attempts. The default is 5.
ConnectionRetryDelay OPTIONAL	The number of seconds the driver waits before retrying connection attempts when ConnectionRetryCount is set to a positive integer. The default is 1.
ConvertNull	{1 | 0}. Controls how data conversions are handled for null values. If set to 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. 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. The default is 1.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
DatabaseName OPTIONAL	The name of the database to which you want to connect. See also "Database Connection Property" on page 5-56.
DescribeParameters	{noDescribe | describeIfString}. 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. 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 set to noDescribe, the driver does not attempt to describe SQL parameters to determine the database data type. The driver sends String parameter values to the server based on the setting of the `SendStringParametersAsUnicode` property. If set to 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. The default is noDescribe.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
EnableCancelTimeout	{true | false}. 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. If set to true, the cancel request times out using the same timeout value, in seconds, that is set for the statement it cancels. For example, if your application 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 set to false (the default), the cancel request does not time out.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
EncryptionMethod	{noEncryption | SSL | requestSSL | loginSSL}. Determines whether SSL encryption is used to encrypt data and login requests transmitted over the network between the driver and database server. See "Data Encryption" on page 5-42 for information about choosing between encrypting data, including login requests, and only encrypting login requests. If set to SSL, the login request and 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 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. NOTE: If 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. See "Data Encryption" on page 5-42 for more information about configuring data encryption. See "Performance Considerations" on page 5-25 for information about configuring this property for optimal performance. The default is noEncryption.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
HostNameInCertificate OPTIONAL	{host_name | #SERVERNAME#}. 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. 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 "Data Encryption" on page 5-42 for information about configuring for authentication. The default is an empty string.
HostProcess OPTIONAL	The process ID of the application connecting to Microsoft SQL Server. The value is a string up to a maximum of 128 characters. The value of this property may be useful for database administration purposes. This value is stored in the hostprocess column of the: • sys.sysprocesses table (Microsoft SQL Server 2005) • master.dbo.sysprocesses table (Microsoft SQL Server 2000) Microsoft SQL Server 7 does not store this value. The default is 0.

Table 5-1 SQL Server Connection Properties (Continued)

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. The following connection URL sets the handling of null values to the Microsoft SQL Server default:
	`jdbc:bea:sqlserver://server1:1433;`
	`InitializationString=set ANSI_NULLS off;`
	`DatabaseName=test`
	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. The following connection URL sets the handling of null values to the Microsoft SQL Server default and allows delimited identifiers:
	`jdbc:bea:sqlserver://server1:1433;`
	`InitializationString=(set ANSI_NULLS off;`
	`set QUOTED_IDENTIFIER on);DatabaseName=test`
	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.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
InsensitiveResultSetBufferSize OPTIONAL	{-1 | 0 | x}. Determines the amount of memory used by the driver to cache insensitive result set data. It must have one of the following values: 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. See "Performance Considerations" on page 5-25 for information about configuring this property for optimal performance. The default is 2048 (KB).
JavaDoubleToString	{true | 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. 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.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description

LoadLibraryPath	Specifies the directory the driver looks in for the DLL used for NTLM authentication. The value is the fully qualified path of the directory that contains the DLL. When you install the driver, the NTLM DLLs are placed in the `*WL_HOME*/server/lib` subdirectory, where *`WL_HOME`* is the directory in which you installed WebLogic Server.
	
	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, you can set this 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 this property to specify the directory containing the NTLM authentication DLLs.
	
	`jdbc:bea:sqlserver://server3:1433;`
	`DatabaseName=test;LoadLibraryPath=C:\DataDirect\lib;`
	`User=test;Password=secret`
	
	See "Configuring NTLM Authentication" on page 5-40 for more information about NTLM authentication.

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

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
LongDataCacheSize	{-1 | 0 | x}. 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. 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 "Performance Considerations" on page 5-25 for information about configuring this property for optimal performance. The default is 2048.
NetAddress OPTIONAL	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.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
PacketSize	PacketSize={-1 | 0 | *x*}. 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. 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 "Performance Considerations" on page 5-25 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. See "Authentication" on page 5-33 for more information about configuring authentication.
PortNumber OPTIONAL	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. The default is 1433.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
ProgramName OPTIONAL	The name of the application connecting to Microsoft SQL Server. This value is a string up to a maximum of 128 characters. The value of this property may be useful for database administration purposes. This value is stored in the program_name column of the: • sys.sysprocesses table (Microsoft SQL Server 2005) • master.dbo.sysprocesses table (Microsoft SQL Server 2000) Microsoft SQL Server 7 does not store this value The default is an empty string.
QueryTimeout	{*positive integer* | -1 | 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).

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
ReceiveStringParameterType	{NVARCHAR | VARCHAR | DESCRIBE}. Specifies how the driver describes String stored procedure output parameters to the database. 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. The default is NVARCHAR

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
ResultSetMetaDataOptions	{0 | 1}. The SQL Server 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 "ResultSet MetaData Support" on page 5-53 for more information about returning ResultSet metadata. The default is 0.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
SelectMethod OPTIONAL	{direct | cursor}. 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. • 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 "Performance Considerations" on page 5-25 for information about configuring this property for optimal performance. The default is Direct.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
SendStringParametersAsUnicode OPTIONAL	{true | false}. Determines whether string parameters are sent to the Microsoft SQL Server database in Unicode or in the default character encoding of the database. 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 "Performance Considerations" on page 5-25 for information about configuring this property for optimal performance. The default is true.
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. 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 "Connecting to Named Instances" on page 5-3 for more information about connecting to named instances.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
SnapshotSerializable	{true | false}. For Microsoft SQL Server 2005 only. Allows your application to use Snapshot Isolation for connections. To configure Snapshot Isolation for connections, you must have your Microsoft SQL Server 2005 database configured for Snapshot Isolation, your application must have the transaction isolation level set to Serializable, and this property must be set to true. If set to false (the default) and your application has the transaction isolation level set to Serializable, the application uses the Serializable isolation level. This property is useful for applications that have the Serializable isolation level set. Using the `SnapshotSerializable` property in this case 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. See "Isolation Levels" on page 5-46 for details. See "Performance Considerations" on page 5-25 for information about configuring this property for optimal performance. The default is false.
TransactionMode	{implicit | explicit}. Controls how the driver delimits the start of a local transaction. 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. The default is implicit.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
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`.
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`.
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.
UseServerSideUpdatableCursors	{true | false}. Determines whether the driver uses server-side cursors when an updatable result set is requested. 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 "Server-Side Updatable Cursors" on page 5-47 for more information about using server-side updatable cursors. See "Performance Considerations" on page 5-25 for information about configuring this property for optimal performance. The default is false.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
ValidateServerCertificate	{true | false}. 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. 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 "Data Encryption" on page 5-42 for information about configuring for authentication.
WSID OPTIONAL	The workstation ID, which typically is the network name of the computer on which the application resides. The value is a string up to a maximum of 128 characters. The value of this property may be useful for database administration purposes and can be returned by sp_who and the Transact-SQL HOST_NAME function. This value is stored in the hostname column of the: • sys.sysprocesses table (Microsoft SQL Server 2005) • master.dbo.sysprocesses table (Microsoft SQL Server 2000) Microsoft SQL Server 7 does not store this value. The default is an empty string.

Table 5-1 SQL Server Connection Properties (Continued)

Property	Description
XATransactionGroup OPTIONAL	The transaction group ID that identifies any transactions initiated by the connection. This ID can be used for distributed transaction cleanup purposes. You can use the XAResource.recover method to roll back any transactions left in an unprepared state. When you call XAResource.recover, any unprepared transactions that match the ID on the connection used to call XAResource.recover are rolled back. 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 "Distributed Transaction Cleanup" on page 5-49 for more information about distributed transaction cleanup.
XMLDescribeType	{longvarchar | longvarbinary}. Determines whether the driver maps XML data to the LONGVARCHAR or LONGVARBINARY data type. 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 "Returning and Inserting/Updating XML Data" on page 5-30 for more information. The default is longvarchar.

Performance Considerations

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

- "EncryptionMethod" on page 5-26
- "InsensitiveResultSetBufferSize" on page 5-26
- "LongDataCacheSize" on page 5-26
- "PacketSize" on page 5-26
- "ResultSetMetaDataOptions" on page 5-27

- "SelectMethod" on page 5-27
- "SendStringParametersAsUnicode" on page 5-27
- "SnapshotSerializable" on page 5-27
- "UseServerSideUpdatableCursors" on page 5-28

EncryptionMethod

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

InsensitiveResultSetBufferSize

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

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.

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.

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 "ResultSet MetaData Support" on page 5-53 for more information about returning ResultSet metadata.

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.

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.

SnapshotSerializable

You must have your Microsoft SQL Server 2005 database configured for Snapshot Isolation for this connection property to work. See "Using the Snapshot Isolation Level (Microsoft SQL Server 2005 Only)" on page 5-46 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 feature can be useful if you want to consistently return the same result set even if another transaction has changed the data and 1) your application executes many read operations or 2) your application has long running transactions that could potentially block users from reading data. This feature has the potential to eliminate data contention between read operations and update operations. When this connection property is set to true (thereby, you are using Snapshot Isolation), performance is improved due to increased concurrency.

UseServerSideUpdatableCursors

In most cases, using server-side updatable cursors improves performance. However, this type of cursor cannot be used with insensitive result sets or with sensitive results sets that are not generated from a database table that contains a primary key.

See "Server-Side Updatable Cursors" on page 5-47 for more information about using server-side updatable cursors.

Data Types

Table 5-2 lists the data types supported by the SQL Server driver in SQL Server 7 and SQL Server 2000 and how they are mapped to the JDBC data types.

Table 5-2 Microsoft SQL Server Data Types

Microsoft SQL Server Data Type	JDBC Data Type
bigint[1]	BIGINT
bigint identity [1]	BIGINT
binary	BINARY
bit	BIT
char	CHAR
datetime	TIMESTAMP
decimal	DECIMAL
decimal() identity	DECIMAL
float	FLOAT
image	LONGVARBINARY

Table 5-2 Microsoft SQL Server Data Types (Continued)

Microsoft SQL Server Data Type	JDBC Data Type
int	INTEGER
int identity	INTEGER
money	DECIMAL
nchar	CHAR
ntext	LONGVARCHAR
numeric	NUMERIC
numeric() identity	NUMERIC
nvarchar	VARCHAR
nvarchar(max)[2]	LONGVARCHAR
real	REAL
smalldatetime	TIMESTAMP
smallint	SMALLINT
smallint identity	SMALLINT
smallmoney	DECIMAL
sql_variant [1]	VARCHAR
sysname	VARCHAR
text	LONGVARCHAR
timestamp	BINARY
tinyint	TINYINT
tinyint identity	TINYINT
uniqueidentifier	CHAR

Table 5-2 Microsoft SQL Server Data Types (Continued)

Microsoft SQL Server Data Type	JDBC Data Type
varbinary	VARBINARY
varbinary(max) [2]	LONGVARBINARY
varchar	VARCHAR
varchar(max) [2]	LONGVARCHAR
xml [2]	LONGVARCHAR

1. Supported only for Microsoft SQL Server 2000 and higher.
2. Supported only for Microsoft SQL Server 2005

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

Returning and Inserting/Updating XML Data

For Microsoft SQL Server 2005, the SQL Server driver supports the XML data type. By default, the driver maps the XML data type to the JDBC LONGVARCHAR data type, but you can choose to map the XML data type to the LONGVARBINARY data type by setting the `XMLDescribeType` connection property to a value of longvarbinary.

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 LONGVARCHAR data type. If the following connection URL mapped the XML data type to the LONGVARBINARY data type, the driver would return the XML data as binary data instead of character data:

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

Character Data

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

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

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

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

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

```
ResultSet.getAsciiStream()
```

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

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

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

Binary Data

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

Your application can use the following methods to return XML data as binary data:

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

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

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

Inserting/Updating XML Data

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

Character Data

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

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

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

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

```
PreparedStatement.setAsciiStream()
ResultSet.updateAsciiStream()
```

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

Binary Data

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

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

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

Authentication

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See "Authentication" on page 2-7 for an overview.

The SQL Server driver supports the following methods of authentication:

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

This method requires knowledge of how to configure your Kerberos environment and supports Windows Active Directory Kerberos only.

- NTLM authentication is a single sign-on Windows authentication method. This method provides authentication from Windows clients only and requires minimal configuration.

Except for NTLM authentication, which provides authentication for Windows clients only, these authentication methods provide authentication when the driver is running on any supported platform.

The `AuthenticationMethod` connection property controls which authentication mechanism the driver uses when establishing connections. See "Using the AuthenticationMethod Property" on page 5-34 for information about setting the value for this property.

Using the AuthenticationMethod Property

The `AuthenticationMethod` connection property controls which authentication mechanism the driver uses when establishing connections. When `AuthenticationMethod=auto`, the driver uses SQL Server authentication, Kerberos authentication, or NTLM authentication when establishing a connection based on the following criteria:

- If a user ID and password is specified, the driver uses SQL Server authentication when establishing a connection. The `User` property provides the user ID. The `Password` property provides the password.
- If a user ID and password is not specified and the driver is not running on a Windows platform, the driver uses Kerberos authentication when establishing a connection.
- If a user ID and password is not specified and the driver is running on a Windows platform, the driver uses NTLM authentication when establishing a connection if the driver can load the DLL required for NTLM authentication. If the driver cannot load the DLL, the driver uses Kerberos authentication.

When `AuthenticationMethod=kerberos`, the driver uses Kerberos authentication when establishing a connection. The driver ignores any values specified by the `User` property and `Password` properties.

When `AuthenticationMethod=ntlm`, the driver uses NTLM authentication when establishing a connection if the driver can load the DLL required for NTLM authentication. If the driver cannot load the DLL, the driver throws an exception. The driver ignores any values specified by the User and Password properties.

When `AuthenticationMethod=userIdPassword` (the default), the driver uses SQL Server authentication when establishing a connection. The `User` property provides the user ID. The

`Password` property provides the password. If a user ID is not specified, the driver throws an exception.

Configuring SQL Server Authentication

1. Set the `AuthenticationMethod` property to auto or userIdPassword (the default). See "Using the AuthenticationMethod Property" on page 5-34 for more information about setting a value for this property.

2. Set the `User` property to provide the user ID.

3. Set the `Password` property to provide the password.

Configuring Kerberos Authentication

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

Product Requirements

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

Table 5-3 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 2005 • 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.

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 "Using the AuthenticationMethod Property" on page 5-34 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:[BEA][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 "Permissions for Kerberos Authentication" on page 2-19 for an example.

See the following URL for more information about configuring and testing your environment for Windows authentication with the SQL Server driver:

http://www.datadirect.com/developer/jdbc/index.ssp

Specifying User Credentials for Kerberos Authentication (Delegation of Credentials)

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

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:bea:sqlserver://myServer:1433";
         con = DriverManager.getConnection(url);
        }
     catch (Exception except) {

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

        return con;
    }
});

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

... // do something with the results
```

Obtaining a Kerberos Ticket Granting Ticket

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

If the application uses Kerberos authentication from a Windows client, the application user is not required to log onto the Kerberos server and explicitly obtain a TGT. Windows Active Directory automatically obtains a TGT for the user.

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

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

where user is the application *user*.

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

Configuring NTLM Authentication

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

Product Requirements

Verify that your environment meets the requirements listed in Table 5-4 before you configure your environment for NTLM authentication.

Table 5-4 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 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 1.3 or higher must be installed.

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 "Using the AuthenticationMethod Property" on page 5-34 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:bea:sqlserver://server3:1521;
   DatabaseName=test;LoadLibraryPath=C:\DataDirect\lib;User=test;Password=
   secret
   ```
3. If using NTLM authentication with a Security Manager on a Java 2 Platform, security permissions must be granted to allow the driver to establish connections. See "Permissions for Establishing Connections" on page 2-16 for an example.

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 "Data Encryption Across the Network" on page 2-11 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.

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

Table 5-5 EncryptionMethod Property and Microsoft SQL Server Configurations

Value	No SSL Certificate	SSL Certificate	
		SSL Optional	SSL Required
noEncryption	Login request and data are not encrypted.	Login request and data are not encrypted.	Connection attempt fails.
SSL	Connection attempt fails.	Login request and data are encrypted.	Login request and data are encrypted.

Table 5-5 EncryptionMethod Property and Microsoft SQL Server Configurations (Continued)

Value	No SSL Certificate	SSL Certificate	
		SSL Optional	SSL Required
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.

Configuring SSL Encryption

1. Choose the type of encryption for your application:
   – If you want the driver to encrypt all data, including the login request, set the `EncryptionMethod` property to SSL or requestSSL.
   – If you want the driver to encrypt only the login request, set the `EncryptionMethod` property to loginSSL.
2. Specify the location and password of the truststore file used for SSL server authentication. Either set the TrustStore and TrustStore properties or their corresponding Java system properties (javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword, respectively).
3. To validate certificates sent by the database server, set the `ValidateServerCertificate` property to true.
4. Optionally, set the `HostNameInCertificate` property to a host name to be used to validate the certificate. The `HostNameInCertificate` property provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.

DML with Results (Microsoft SQL Server 2005)

The SQL Server driver supports the Microsoft SQL Server 2005 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();
   }
```

SQL Escape Sequences

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

Isolation Levels

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

- Read Committed with Locks * or Read Committed
- Read Committed with Snapshots *
- Read Uncommitted
- Repeatable Read
- Serializable
- Snapshot *

* Supported for Microsoft SQL Server 2005 only.

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

Using the Snapshot Isolation Level (Microsoft SQL Server 2005 Only)

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 5-1 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);
```

Using Scrollable Cursors

The SQL Server driver supports scroll-sensitive result sets, scroll-insensitive result sets, and updatable result sets.

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

Server-Side Updatable Cursors

The SQL Server driver can use client-side cursors or server-side cursors to support updatable result sets. By default, the SQL Server driver uses client-side cursors because this type of cursor can work with any result set type. Using server-side cursors typically can improve performance, but server-side cursors cannot be used with scroll-insensitive result sets or with scroll-sensitive result sets that are not generated from a database table that contains a primary key. To use server-side cursors, set the `UseServerSideUpdatableCursors` property to true.

When the `UseServerSideUpdatableCursors` property is set to true and a scroll-insensitive updatable result set is requested, the driver downgrades the request to a scroll-insensitive read-only result set. Similarly, when a scroll-sensitive updatable result set is requested and the table from which the result set was generated does not contain a primary key, the driver downgrades the request to a scroll-sensitive read-only result set. In both cases, a warning is generated.

When server-side updatable cursors are used with sensitive result sets that 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.

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:\bea\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.

Distributed Transaction Cleanup

Connections associated with distributed transactions can become orphaned if the connection to the server is lost before the transaction has completed. When connections associated with distributed transactions are orphaned, any locks held by the database for that transaction are maintained, which can cause data to become unavailable. By cleaning up distributed transactions, connections associated with those transactions are freed and any locks held by the database are released.

You can use the XAResource.recover method to clean up distributed transactions that have been prepared, but not committed or rolled back. Calling this method returns a list of active distributed transactions that have been prepared, but not committed or rolled back. An application can use the list returned by the XAResource.recover method to clean up those transactions by explicitly committing them or rolling them back. The list of transactions returned by the XAResource.recover method does not include transactions that are active and have not been prepared.

In addition, the SQL Server driver supports the following methods of distributed transaction cleanup:

- Transaction timeout sets a timeout value that is used to audit active transactions. Any active transactions that have a life span greater than the specified timeout value are rolled back. Setting a transaction timeout allows distributed transactions to be cleaned up automatically based on the timeout value.
- Explicit transaction cleanup allows you to explicitly roll back any transactions left in an unprepared state based on a transaction group identifier. Explicit transaction cleanup provides more control than transaction timeout over when distributed transactions are cleaned up.

Transaction Timeout

To set a timeout value for transaction cleanup, you use the XAResource.setTransactionTimeout method. Setting this value causes sqljdbc.dll on the server side to maintain a list of active transactions. Distributed transactions are placed in the list of active transactions when they are started and removed from this list when they are prepared, rolled back, committed, or forgotten using the appropriate XAResource methods.

When a timeout value is set for transaction cleanup using the XAResource.setTransactionTimeout method, sqljdbc.dll periodically audits the list of active transactions for expired transactions. Any active transactions that have a life span greater than the timeout value are rolled back. If an exception is generated when rolling back a transaction, the exception is written to the sqljdbc.log file, which is located in the same directory as the sqljdbc.dll file.

Setting the transaction timeout value too low means running the risk of rolling back a transaction that otherwise would have completed successfully. As a general guideline, set the timeout value to allow sufficient time for a transaction to complete under heavy traffic load.

Setting a value of 0 (the default) disables transaction timeout cleanup.

Explicit Transaction Cleanup

The SQL Server driver allows you to associate an identifier with a group of transactions using the `XATransactionGroup` connection property. When you specify a transaction group ID, all distributed transactions initiated by the connection are identified by this ID.

Setting this value causes sqljdbc.dll on the server side to maintain a list of active transactions. Distributed transactions are placed in the list of active transactions when they are started and removed from this list when they are prepared, rolled back, committed, or forgotten using the appropriate XAResource methods.

You can use the XAResource.recover method to roll back any transactions left in an unprepared state that match the transaction group ID on the connection used to call XAResource.recover. For example, if you specified `XATransactionGroup=ACCT200` and called the XAResource.recover method on the same connection, any transactions left in an unprepared state with a transaction group ID of ACCT200 would be rolled back.

If an exception is generated when rolling back a transaction, the exception is written to the sqljdbc.log file, which is located in the same directory as the sqljdbc.dll file.

When using explicit transaction cleanup, distributed transactions associated with orphaned connections, and the locks held by those connections, will persist until the application explicitly invokes them. As a general rule, applications should clean up orphaned connections at startup and when the application is notified that a connection to the server was lost.

Large Object (LOB) Support

Although Microsoft SQL Server does not define a Blob or Clob data type, the SQL Server driver allows you to return and update long data, specifically LONGVARBINARY and LONGVARCHAR data, using JDBC methods designed for Blobs and Clobs. When using these methods to update long data as Blobs or Clobs, the updates are made to the local copy of the data contained in the Blob or Clob object.

Retrieving and updating long data using JDBC methods designed for Blobs and Clobs provides some of the same advantages as retrieving and updating Blobs and Clobs. For example, using Blobs and Clobs:

- Provides random access to data
- Allows searching for patterns in the data, such as returning long data that begins with a specific character string

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

Batch Inserts and Updates

The SQL Server driver implementation for batch Inserts and Updates is JDBC 3.0 compliant. When the SQL Server driver detects an error in a statement or parameter set in a batch Insert or Update, it generates a BatchUpdateException and continues to execute the remaining statements or parameter sets in the batch. The array of update counts contained in the BatchUpdateException contain one entry for each statement or parameter set. Any entries for statements or parameter sets that failed contain the value Statement.EXECUTE_FAILED.

Parameter Metadata Support

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

Insert and Update Statements

The SQL Server driver supports returning parameter metadata for the following forms of Insert and Update statements:

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

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

Select Statements

The SQL Server driver supports returning parameter metadata for Select statements that contain parameters in ANSI SQL 92 entry-level predicates, for example, such as COMPARISON, BETWEEN, IN, LIKE, and EXISTS predicate constructs. Refer to the ANSI SQL reference for detailed syntax.

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

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

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

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

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

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

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

```
SELECT col1, col2 FROM foo WHERE col1 = ? and col2 > ?
SELECT ... WHERE colname = (SELECT col2 FROM t2
   WHERE col3 = ?)
SELECT ... WHERE colname LIKE ?
SELECT ... WHERE colname BETWEEN ? and ?
SELECT ... WHERE colname IN (?, ?, ?)
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE col1 < ?)
```

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

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

Joins are supported. For example:

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

Fully qualified names and aliases are supported. For example:

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

Stored Procedures

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

ResultSet MetaData Support

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

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

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

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

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

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

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

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

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

Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

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.

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.

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

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

  ```
  SET ANSI_NULLS OFF
  ```

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:bea:sqlserver://server1:1433;DatabaseName=jdbc;Database=acct;
User=test;Password=secret
```

The Oracle Driver (Deprecated)

Note: The WebLogic Type 4 JDBC Oracle driver described in this document has been deprecated as of release 10.3 of WebLogic Server. It will be removed in the next release of WebLogic Server. Instead of this deprecated driver, use the Oracle Thin Driver that is also provided with WebLogic Server. For details about the Oracle Thin Driver, see "Using Third-Party JDBC Drivers with WebLogic Server" in *Configuring and Managing WebLogic JDBC.*

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

Oracle Database Version Support

The WebLogic Type 4 JDBC Oracle driver (the "Oracle driver") supports:

- Oracle 9i R1 and R2
- Oracle 10*g* R1 and R2
- Oracle 11g

Oracle Driver Classes

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

- XA: `weblogic.jdbcx.oracle.OracleDataSource`
- Non-XA: `weblogic.jdbc.oracle.OracleDriver`

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

Oracle URL

The connection URL format for the Oracle driver is:

```
jdbc:bea:oracle://hostname:port[;property=value[;...]]
```

where:

- *`hostname`* is the TCP/IP address or TCP/IP host name of the server to which you are connecting. See "Using IP Addresses" on page 2-5 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 "Oracle Connection Properties" on page 6-3.

For example:

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

See "Using tnsnames.ora Files" on page 6-25 for instructions on retrieving connection information from an Oracle tnsnames.ora file.

Oracle Connection Properties

Table 6-1 lists the JDBC connection properties supported by the Oracle 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
```

All connection property names are case-insensitive. For example, Password is the same as password. Required properties are noted as such.

Table 6-1 Oracle Connection String Properties

Property	Description
AuthenticationMethod	{auto | kerberos | kerberosUIDPassword | ntlm | client | userIDPassword}. Determines which authentication method the driver uses when establishing a connection. If set to auto (the default), the driver uses user ID/password, Kerberos, or NTLM authentication when establishing a connection. The driver selects an authentication method based on a combination of criteria, such as whether the application provides a user ID, the driver is running on a Windows platform, and the driver can load the DLL required for NTLM authentication. See "Using the AuthenticationMethod Property" on page 6-40 for more information about using this value. If set to kerberos, the driver uses Kerberos authentication. The driver ignores any user ID or password specified. If set to kerberosUIDPassword, the driver first uses Kerberos to authenticate the user. Next, the driver reauthenticates the user using user ID/password authentication. If a user ID and password are not specified, the driver throws an exception. If either Kerberos or user ID/password authentication fails, the connection attempt fails and the driver throws an exception. If set to ntlm, the driver uses NTLM authentication if the DLL required for NTLM authentication can be loaded. If the driver cannot load the DLL, the driver throws an exception. The driver ignores any user ID or password specified. This value is supported for Windows clients only. If set to client, the driver uses the user ID of the user logged onto the system on which the driver is running to authenticate the user to the database. The Oracle database server relies on the client to authenticate the user and does not provide additional authentication. The driver ignores any user ID or password specified. If set to userIdPassword, the driver uses user ID/password authentication. If a user ID and password are not specified, the driver throws an exception. NOTE: The values type2 and none are deprecated, but are recognized for backward compatibility. We recommend that you use the ntlm and userIdPassword values, respectively, instead. The `User` property provides the user ID. The `Password` property provides the password. See "Authentication" on page 6-39 for more information. The default is auto.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
BatchPerformanceWorkaround OPTIONAL	{true | false}. Determines the method used to execute batch operations. If set to true, the native Oracle batch mechanism is used. The native Oracle batch mechanism does not return individual update counts for each statement or parameter set in the batch. For this reason, the driver returns a value of SUCCESS_NO_INFO (-2) for each entry in the returned update count array. If an application can accept not receiving update count information, setting this property to true can significantly improve performance. If set to false, the JDBC 3.0-compliant batch mechanism is used. The default is false. See "Batch Inserts and Updates" on page 6-50 for details. See "Performance Considerations" on page 6-22 for information about configuring this property for optimal performance.
CatalogIncludesSynonyms DEPRECATED	This property is recognized for compatibility with existing data sources, but we recommend that you use the CatalogOptions property instead to include synonyms in result sets.
CatalogOptions OPTIONAL	{0 | 1 | 2 | 3}. Determines the type of information included in result sets returned from catalog functions. If set to 0, result sets contain neither synonyms or remarks. If set to 1, result sets contain remarks information returned from the DatabaseMetaData methods: getTables() and getColumns(). If set to 2 (the default), result sets contain synonyms returned from the DatabaseMetaData methods: getColumns(), getImportedKeys(), getExportedKeys(), getPrimaryKey(), getProcedures(), getProcedureColumns(), and getIndexInfo(). If set to 3, result sets contain remarks and synonyms (as described in options 1 and 2). The default is 2. See "Performance Considerations" on page 6-22 for information about configuring this property for optimal performance.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
CodePageOverride	{UTF8 | SJIS | ENHANCED_SJIS | ENHANCED_SJIS_ORACLE | MS932}. The code page to be used by the driver to convert Character data. The specified code page overrides the default database code page or column collation. All Character data returned from or written to the database is converted using the specified code page. This option has no effect on how the driver converts character data to the national character set. 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 set to UTF8, the driver uses the UTF-8 code page to send data to the Oracle server as Unicode. The UTF8 code page converts data from the Java String format UTF-16 to UTF-8. If you specify this value, the driver forces the value of the `WireProtocolMode` property to 2. If set to SJIS, the driver uses the SHIFT-JIS code page to convert character data to the JA16SJIS character set. If set to ENHANCED_SJIS, the driver uses the ENHANCED_SJIS code page to convert character data from the Java String format UTF-16 to SJIS as defined by the ICU character conversion library. In addition, it maps the following MS-932 characters to the corresponding SJIS encoding for those characters: \UFF5E Wave dash \U2225 Double vertical line \UFFE0 Cent sign \UFF0D Minus sign \UFFE1 Pound sign \UFFE2 Not sign This value is provided for backward compatibility. Only use this value when the Oracle database character set is SHIFT_JIS. If set to ENHANCED_SJIS_ORACLE, the driver uses the ENHANCED_SJIS_ORACLE code page to convert Character data from the Java String format UTF-16 to Oracle's definition of SJIS. When the driver connects to an Oracle database with a JA16SJIS character set, the driver uses this code page by default. The ENHANCED_SJIS_ORACLE code page is a super set of the MS932 code page. Only use this value when the Oracle database character set is SHIFT_JIS.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
CodePageOverride (*cont.*)	If set to MS932, the driver uses the Microsoft MS932 code page to convert Character data from the Java String format UTF-16 to SJIS. This value is provided for backward compatibility because earlier versions of the driver used the MS932 code page when converting Character data to JA16SJIS. Only use this value when the Oracle database character set is SHIFT_JIS.
CommitBehavior	{serverDefault | waitImmediate | waitBatch | noWaitImmediate | noWaitBatch}. Typically, redo changes generated by update transactions are written to disk immediately when an transaction is committed, and the session waits for the disk write to complete before returning control to the application. Oracle 10g R2 can let the log writer write the redo changes to disk in its own time instead of immediately and return control to the application before the disk write is complete instead of waiting. This property controls this behavior by setting the value of the Oracle COMMIT_WRITE session parameter. Not waiting for redo log changes to be written to disk improves performance for applications that have both of the following characteristics: • Applications that perform update operations. • Applications where data integrity is not critical. For example, most banking applications cannot tolerate data loss in the event that the server has a problem writing the redo log changes to disk or fails during the process, but many logging applications for diagnostic purposes can. If set to serverDefault (the default), the driver uses the redo log behavior set by the database server. If set to waitImmediate, the commit operation does not return control to the application until redo changes are written to disk. Redo changes are written to disk immediately. Use this value if your application processes multiple update transactions one at a time. If set to waitBatch, the commit operation does not return control to the application until redo changes are written to disk. The write task may be deferred by the server until additional transactions are ready to be written to disk. Use this value if your application processes multiple update transactions simultaneously. Using this value when an application performs only a few transactions decreases performance

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
CommitBehavior (*cont.*)	If set to noWaitImmediate, redo changes are written to disk immediately, but the commit operation returns control to the application without waiting for this operation to complete. Use this value if your application processes multiple update transactions one at time and data integrity is not critical. If set to noWaitBatch, the redo write task may be deferred by the server until additional transactions are ready to be written to disk, but the commit operation returns control to the application without waiting for this operation to complete. Use this value if your application processes multiple update transactions simultaneously and data integrity is not critical. See "Performance Considerations" on page 6-22 for information about configuring this property for optimal performance.
ConnectionRetryCount OPTIONAL	The number of times the driver retries connections to the database server until a successful connection is established. Valid values are 0 and any positive integer. If set to 0, the driver does not try to reconnect after the initial unsuccessful attempt. If an application sets a login timeout value (for example, using DataSource.loginTimeout or DriverManager.loginTimeout), the login timeout takes precedence over this property. The ConnectionRetryDelay property specifies the wait interval, in seconds, used between attempts. The default is 5.
ConnectionRetryDelay OPTIONAL	The number of seconds the driver waits before retrying connections to the database server when ConnectionRetryCount is set to a positive integer. The ConnectionRetryCount property specifies the number of times the driver will attempt to connect to the database server. The default is 1.
ConvertNull	{1 | 0}. Controls how data conversions are handled for null values. If set to 1 (the default), the driver checks the data type being requested against the data type of the column from which the data is being returned. If a conversion between the requested type and column type is not defined, the driver generates an "unsupported data conversion" exception regardless 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. The default is 1.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
EnableCancelTimeout	{true | false}. 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. If set to true, the cancel request times out using the same timeout value, in seconds, that is set for the statement it cancels. For example, if your application calls Statement.setQueryTimeout(5) on a statement and that statement is cancelled because its timeout value was exceeded, a cancel request is sent that also will time out if its execution exceeds 5 seconds. If the cancel request times out, because the server is down, for example, the driver throws an exception indicating that the cancel request was timed out and the connection is no longer valid. If set to false (the default), the cancel request does not time out.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description

EncryptionMethod	{noEncryption | SSL}. Determines whether SSL encryption is used to encrypt and decrypt data transmitted over the network between the driver and database server.
	
	If set to noEncryption (the default), 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 set to SSL, data is encrypted using SSL. If the database server does not support SSL, the connection fails and the driver throws an exception. When SSL is enabled, the following properties also apply:
	
	`HostNameInCertificate`
	
	`KeyStore` (for SSL client authentication)
	
	`KeyStorePassword` (for SSL client authentication)
	
	`KeyPassword` (for SSL client authentication)
	
	`TrustStore`
	
	`TrustStorePassword`
	
	`ValidateServerCertificate`
	
	See "Using tnsnames.ora Files" on page 6-25 for information about enabling SSL encryption using a tnsnames.ora file.
	
	Note: Connection hangs can occur if the driver attempts to connect to a database server that does not support SSL. You may want to set a login timeout using the `LoginTimeout` property to avoid problems when connecting to a server that does not support SSL.
	
	See "Data Encryption" on page 6-48 for more information about configuring data encryption.
	
	See "Performance Considerations" on page 6-22 for information about configuring this property for optimal performance.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
FetchTSWTZasTimestamp OPTIONAL	{true | false}. If set to true, allows column values with the TIMESTAMP WITH TIME ZONE data type (Oracle9i or higher) to be returned as a JDBC TIMESTAMP data type. If set to false, column values with the TIMESTAMP WITH TIME ZONE data type must be retrieved as a string. The default is false. See "TIMESTAMP WITH TIME ZONE Data Type" on page 6-34 for more information.
HostNameInCertificate	{host_name | #SERVERNAME#}. 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. 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 in the SubjectAlternativeName or if the certificate does not have a SubjectAlternativeName, the driver compares the host name with the Common Name (CN) part of the certificate's Subject name. If the values do not match, the connection fails and the driver throws an exception. If #SERVERNAME# is specified, the driver compares the server name specified in the connection URL or data source of the connection to the DNSName value of the SubjectAlternativeName in the certificate. If a DNSName value does not exist in the SubjectAlternativeName or if the certificate does not have a SubjectAlternativeName, the driver compares the host name to the CN 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 uses the server name specified in the connection URL or data source of the connection to validate the certificate. If SSL encryption or certificate validation is not enabled, this property is ignored. See "Data Encryption" on page 6-48 for information about configuring for authentication. The default is an empty string.

Table 6-1 Oracle Connection String Properties (Continued)

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. 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:bea:oracle://server1:1521;`
	`ServiceName=ORCL;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.

InsensitiveResultSetBuffer Size OPTIONAL	{-1 | 0 | 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. The default is 2048 (KB). See "Performance Considerations" on page 6-22 for information about configuring this property for optimal performance.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
JavaDoubleToString	{true | 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.
KeyStore	Specifies the directory of the keystore file to be used when SSL is enabled using the `EncryptionMethod` property 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 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.
KeyStorePassword	Specifies the password used to access the keystore file when SSL is enabled using the `EncryptionMethod` property 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 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.
KeyPassword	Specifies the password used to access the individual keys in the keystore file when SSL is enabled using the `EncryptionMethod` property and SSL client authentication is enabled on the database server. This property is useful if any of the keys in the keystore file have a different password than the keystore file.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
LoginTimeout OPTIONAL	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. If set to 0 (the default), the driver does not time out a connection request.
Password	A case-insensitive password used to connect to your Oracle database. A password is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your password. See "Authentication" on page 6-39 for information about configuring for authentication.
PortNumber OPTIONAL	The TCP port of the Oracle listener running on the Oracle database server. The default is 1521,which is the default port number the Oracle database software uses during its installation. This property is supported only for data source connections. If using a tnsnames.ora file to provide connection information, do not specify this property. See "Using tnsnames.ora Files" on page 6-25 for information about specifying a port number for the Oracle listener using a tnsnames.ora file.
QueryTimeout	{positive integer | -1 | 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).

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
ResultSetMetaDataOptions	{0 | 1}. The Oracle 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 "Performance Considerations" on page 6-22 for information about configuring this property for optimal performance. The default is 0.
SendFloatParametersAsString	{true | false}. Determines whether FLOAT, BINARY_FLOAT, and BINARY_DOUBLE parameters are sent to the database server as a string or as a floating point number. If set to true, the driver sends FLOAT, BINARY_FLOAT, and BINARY_DOUBLE parameters to the database server as string values. If set to false (the default), the driver sends FLOAT, BINARY_FLOAT, and BINARY_DOUBLE parameters to the database server as floating point numbers. When Oracle overloaded stored procedures are used, this value ensures that the database server can determine the correct stored procedure to call based on the parameter's data type. NOTE: Numbers larger than 1.0E127 or smaller than 1.0E-130 cannot be converted to Oracle's number format for Oracle 9i databases using floating point numbers. When a number larger than 1.0E127 or smaller than 1.0E-130 is encountered, the driver throws an exception. If your application uses numbers in this range against an Oracle 9i database, set this property to true. The default is false.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
ServerName OPTIONAL	Specifies either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the Oracle server. For example, 122.23.15.12 or OracleAppServer. If using a tnsnames.ora file to provide connection information, do not specify this property. This property is supported only for data source connections. See "Performance Considerations" on page 6-22 for information about specifying a server name using a tnsnames.ora file.
ServerType OPTIONAL	{Shared | Dedicated}. Specifies whether the connection is established using a shared or dedicated server process (UNIX) or thread (Windows). If set to Shared, the server process to be used is retrieved from a pool. The socket connection between the client and server is made to a dispatcher process on the server. This setting allows there to be fewer processes than the number of connections, reducing the need for server resources. Use this value when a server must handle many users with fewer server resources. If set to Dedicated, a server process is created to service only that connection. When that connection ends, so does the process (UNIX) or thread (Windows). The socket connection is made directly between the application and the dedicated server process or thread. When connecting to UNIX servers, a dedicated server process can provide significant performance improvement, but uses more resources on the server. When connecting to Windows servers, the server resource penalty is insignificant. Use this value if you have a batch environment with low numbers of users. If unspecified, the driver uses the server type set on the server. If using a tnsnames.ora file to provide connection information, do not specify this property. See "Using tnsnames.ora Files" on page 6-25 for information about specifying the server type using a tnsnames.ora file. See "Performance Considerations" on page 6-22 for information about specifying the server type using a `tnsnames.ora` file.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
ServiceName OPTIONAL	The database service name that specifies the database used for the connection. This property is mutually exclusive with the `SID` property. The service name is a string that is the global database name-a name that typically comprises the database name and domain name. For example: `sales.us.acme.com` This property is useful to specify connections to an Oracle Real Application Clusters (RAC) system rather than a specific Oracle instance because the nodes in a RAC system share a common service name. If using a tnsnames.ora file to provide connection information, do not specify this property. See "Performance Considerations" on page 6-22 for information about specifying the database service name using a tnsnames.ora file.
SID OPTIONAL	The Oracle System Identifier that refers to the instance of the Oracle database running on the server. This property is mutually exclusive with the `ServiceName` property. The default is ORCL, which is the default SID that is configured when installing your Oracle database. If using a tnsnames.ora file to provide connection information, do not specify this property. See "Performance Considerations" on page 6-22 for information about specifying an Oracle SID using a tnsnames.ora file.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
SysLoginRole	{sysdba | sysoper}. Specifies whether the user is logged on the database with the Oracle system privilege SYSDBA or the Oracle system privilege SYSOPER. For example, you may want the user to be granted the SYSDBA privilege to allow the user to create or drop a database. If set to sysdba, the user is logged on the database with the Oracle system privilege SYSDBA. If set to sysoper, the user is logged on the database with the Oracle system privilege SYSOPER. Refer to your Oracle documentation for information about which operations are authorized for the SYSDBA and SYSOPER system privileges. NOTE: The user must be granted SYSDBA or SYSOPER system privileges before the connection is attempted by the driver. If not, the driver throws an exception and the connection attempt fails. If this property is set to an empty string or is not specified, the user is logged in without SYSDBA or SYSOPER privileges. The default is an empty string.
SupportLinks	{true | false}. Determines whether the driver supports Oracle linked servers, which means a mapping has been defined in one Oracle server to another Oracle server. If set to true, the driver supports Oracle linked servers. When Oracle linked servers are supported, the driver does not support distributed transactions. If set to false (the default), the driver does not support Oracle linked servers. In addition, the driver supports distributed transactions. In most cases, setting this property to false provides the best performance.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description

TNSNamesFile
 	
OPTIONAL
	The path and filename to the tnsnames.ora file from which connection information is retrieved. The tnsnames.ora file contains connection information that is mapped to Oracle net service names. Using a tnsnames.ora file to centralize connection information simplifies maintenance when changes occur.
	
	The value of this property must be a valid path and filename to a tnsnames.ora file.
	
	If you specify this property, you also must specify the `TNSServerName` property.
	
	If this property is specified, do not specify the following properties to prevent connection information conflicts:
	
	`PortNumber`
	
	`ServerName`
	
	`ServerType`
	
	`ServiceName`
	
	`SID`
	
	If any of these properties are specified in addition to this property, the driver throws an exception. See "Using tnsnames.ora Files" on page 6-25 for information about using tnsnames.ora files to connect.

TNSServerName
 	
OPTIONAL
	The Oracle net service name used to reference the connection information in a tnsnames.ora file. The value of this property must be a valid net service name entry in the tnsnames.ora file specified by the `TNSNamesFile` property.
	
	If this property is specified, you also must specify the `TNSNamesFile` property.
	
	If this property is specified, do not specify the following properties to prevent connection information conflicts:
	
	`PortNumber`
	
	`ServerName`
	
	`ServerType`
	
	`ServiceName`
	
	`SID`
	
	If any of these properties are specified in addition to this property, the driver throws an exception. See "Using tnsnames.ora Files" on page 6-25 for information about using tnsnames.ora files to connect.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
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 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`.
TrustStorePassword	Specifies the password 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 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`.
User	The case-insensitive default user name used to connect to your Oracle database. A user name is required only if user ID/password authentication is enabled on your database. If so, contact your system administrator to obtain your user name. See“Authentication” on page 6-39 for information about configuring for authentication.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
ValidateServerCertificate	{true | false}. 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. If set to true (the default), 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. If set to false, 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. Truststore information is specified using the TrustStore and TrustStorePassword properties or by using Java system properties. See "Data Encryption" on page 6-48 for information about configuring for authentication. The default is true.

Table 6-1 Oracle Connection String Properties (Continued)

Property	Description
WireProtocolMode	{1 | 2}. Specifies whether the driver optimizes network traffic to the Oracle server for result sets for repeating data in some or all columns, and for inserts and updates of images, pictures, long text, or binary data (Blob and Clob data). If set to 1 (the default), the driver operates in normal wire protocol mode without optimizing network traffic for result sets for repeating data in some or all columns, and inserts and updates of Blob and Clob data. If set to 2, the driver optimizes network traffic to the Oracle server for: Result sets containing multiple rows that have repeating data in some or all columns. Specifically, if the same column contains identical data across multiple consecutive rows in the result set, setting this value can improve performance. Setting this value may degrade performance for single row result sets or result sets that do not contain repeating data. Inserts and updates of Blob and Clob data. See "Performance Considerations" on page 6-22 for information about configuring this property for optimal performance. The default is1.

Performance Considerations

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

- "BatchPerformanceWorkaround" on page 6-23
- "CatalogOptions" on page 6-23
- "CommitBehavior" on page 6-23
- "EncryptionMethod" on page 6-23
- "InsensitiveResultSetBufferSize" on page 6-24
- "ResultSetMetaDataOptions" on page 6-24
- "ServerType" on page 6-24
- "WireProtocolMode" on page 6-24

BatchPerformanceWorkaround

The driver can use a JDBC 3.0-compliant batch mechanism or the native Oracle batch mechanism to execute batch operations. If your application does not use update count information, performance can be improved by using the native Oracle batch environment. The JDBC 3.0-compliant mechanism returns individual update counts for each statement or parameter set in the batch as required by the JDBC 3.0 specification. The native Oracle batch mechanism does not return individual update counts for each statement or parameter set in the batch. For this reason, when the native Oracle batch mechanism is used, the driver returns a value of SUCCESS_NO_INFO (-2) in the returned update count array.

CatalogOptions

Retrieving synonym and remarks information is very expensive with Oracle. If your application does not need to return this information, the driver can improve performance. Standard JDBC behavior is to include synonyms in the result set of calls to the following DatabaseMetaData methods: `getColumns()`, `getProcedures()`, `getProcedureColumns()`, and `getIndexInfo()`. In addition, the driver can include Remarks information in the result sets of calls to the following DatabaseMetaData methods: `getTables()` and `getColumns()`.

CommitBehavior

Typically, redo changes generated by update transactions are written to disk immediately when the transaction is committed, and the session waits for the disk write to complete before returning control to the application. Oracle 10g R2 can let the log writer write the redo changes to disk in its own time instead of immediately and return control to the application before the disk write is complete instead of waiting. Not waiting for the disk write improves performance for applications that perform update operations and where data integrity is not critical. For example, most banking applications cannot tolerate data loss in the event that the server has a problem writing the redo changes to disk or fails during the process, but many logging applications for diagnostic purposes can.

EncryptionMethod

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

InsensitiveResultSetBufferSize

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

ResultSetMetaDataOptions

By default, the Oracle 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 "ResultSet MetaData Support" on page 6-52 for more information about returning ResultSet metadata.

ServerType

When using a dedicated server connection, a server process on UNIX (a thread on Windows) is created to serve only your application connection. When you disconnect, the process goes away. The socket connection is made directly between your application and this dedicated server process. This can provide considerable performance improvements, but will use significantly more resources on UNIX servers. Because this is a thread on Oracle servers running on Windows platforms, the additional resource usage on the server is significantly less. The `ServerType` property should be set to dedicated when you have a batch environment with lower numbers of connections, your Oracle server has excess processing capacity and memory available when at maximum load, or if you have a performance-sensitive application that would be degraded by sharing Oracle resources with other applications.

WireProtocolMode

Set this property to 2 if:

- Your application executes Select statements that return more than one row, the rows returned have repeating data in some or all of the columns, and the repeated data is in consecutive rows (for example, the data in column1/row1 is the same as the data in column1/row2), or

- Your application updates or inserts images, pictures, or long text or binary data.

In either of these cases, performance can be improved by setting this property to 2.

When set to 2, the driver optimizes network traffic to the Oracle server for repeating or long data.

If your application:

- Returns single row result sets or result sets that do not contain repeating data, or
- Does not update or insert long data,

then this property should be set to 1; otherwise, performance may be degraded.

Using tnsnames.ora Files

The `tnsnames.ora` file is used to map connection information for each Oracle service to a logical alias. The Oracle driver allows you to retrieve basic connection information from a `tnsnames.ora` file, including:

- Oracle server name and port
- Oracle System Identifier (SID) or Oracle service name
- Server process type (shared or dedicated)
- Connection failover instructions
- Client load balancing instructions
- Data encryption instructions

In a `tnsnames.ora` file, connection information for an Oracle service is associated with an alias, or Oracle net service name. Each net service name entry contains connect descriptors that define listener and service information. The following example in Listing 6-1 shows connection information in a `tnsnames.ora` file configured for the net service name entries, FITZGERALD.SALES and ARMSTRONG.ACCT.

Listing 6-1 tnsnames.ora Example

```
FITZGERALD.SALES =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = server1)(PORT = 1521))
       (CONNECT_DATA =
          (SID = ORCL)
       )
  )
ARMSTRONG.ACCT =
  (DESCRIPTION =
  (ADDRESS_LIST=
     (FAILOVER = on)
     (LOAD_BALANCE = on)
     (ADDRESS= (PROTOCOL = TCP)(HOST = server1)(PORT = 1521))
     (ADDRESS= (PROTOCOL = TCP)(HOST = server2)(PORT = 1521))
     (ADDRESS= (PROTOCOL = TCP)(HOST = server3)(PORT = 1521))
)
  (CONNECT_DATA=
     (SERVICE_NAME = acct.us.yourcompany.com)
    )
)
```

Using this example, if the Oracle driver referenced the Oracle net service name entry FITGERALD.SALES, the driver would connect to the Oracle database instance identified by the Oracle SID ORCL (`SID=ORCL`). Similarly, if the Oracle driver referenced ARMSTRONG.ACCT, the driver would connect to the Oracle database identified by the service name acct.us.yourcompany.com (`SERVICE_NAME=acct.us.yourcompany.com`). In addition, the driver would enable connection failover (`FAILOVER=on`) and client load balancing (`LOAD_BALANCE=on`).

Typically, a `tnsnames.ora` file is installed when you install an Oracle database. By default, the `tnsnames.ora` file is located in the `ORACLE_HOME\network\admin` directory on Windows and the `$ORACLE_HOME/network/admin` directory on UNIX.

Connecting to the Database

To retrieve connection information from an Oracle tnsnames.ora file with the Oracle driver, you must inform the driver which tnsnames.ora file (using the `TNSNamesFile` property) and Oracle service name entry (using the `TNSServerName` property) to use so that the driver can reference the correct connection information. For example:

```
   <JDBCConnectionPool
   DriverName="weblogic.jdbc.oracle.OracleDriver"
   Name="myDriver"
   PasswordEncrypted="{3DES}r8a+P5qIVJzgiWQDTAN/OA=="
   Properties="TNSServerName=myTNSServerName;user=user;TNSNamesFile=/usr/l
ocal/network/admin/tnsnames.ora"

   Targets="myserver"
   TestConnectionsOnReserve="true"
   TestTableName="SQL SELECT 1 FROM DUAL"
   URL="jdbc:bea:oracle:TNSNamesFile=/usr/local/network/admin/tnsnames.ora
"
   XAPasswordEncrypted="" />
```

The URL specifies the path and filename of the `tnsnames.ora` file (`jdbc:bea:oracle:TNSNamesFile=/usr/local/network/admin/tnsnames.ora`) and the Properties specifies the server name (`TNSServerName=myTNSServerName`) to use for the connection.

Notes:

- The connection URL does not specify the server name and port of the database server; that information is specified in the tnsnames.ora file referenced by the `TNSNamesFile` property.
- If coding a path on Windows to the tnsnames.ora file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: `TNSNamesFile=c:\\oracle92\\NETWORK\\ADMIN\\tnsnames.ora`.

If using tnsnames.ora files with a Security Manager on a Java 2 Platform, read permission must be granted to the tnsnames.ora file. See "Granting Access to Oracle tnsnames.ora Files" on page 2-18 for an example.

Configuring the tnsnames.ora File

If using a `tnsnames.ora` file to retrieve connection information, do not specify the following connection properties to prevent connection information conflicts:

- `EncryptionMethod`
- `ServerName`
- `ServiceName`

- `PortNumber`
- `ServerType`
- `SID`

If any of these properties are specified in addition to the `TNSNamesFile` and `TNSServerName` properties, the driver throws an exception.

Table 6-2 lists the Oracle driver properties that correspond to tnsnames.ora connect descriptor parameters. If using a `tnsnames.ora` file, do not specify any of the driver properties listed to prevent connection information conflicts.

Table 6-2 Oracle Driver Property Mappings to tnsnames.ora Connect Descriptor Parameters

Oracle Driver Property	tnsnames.ora Attribute

PortNumber = *`port`*	PORT = *`port`*
	
	The ADDRESS_LIST parameter contains connection information for one or multiple servers, using the ADDRESS parameter to specify the primary and alternate servers. The PORT parameter is used within the ADDRESS parameter to specify the port number for each server entry. For example:
	
	`(ADDRESS_LIST=`
	`   (ADDRESS= (PROTOCOL = TCP)(HOST = server1)`
	`      (PORT = 1521))`
	`   ...`
	` )`
	
	A port of 1521, the default port number when installing an Oracle database, is specified for server1.

Table 6-2 Oracle Driver Property Mappings to tnsnames.ora Connect Descriptor Parameters (Continued)

Oracle Driver Property	tnsnames.ora Attribute

EncryptionMethod={noEncryption | SSL}	PROTOCOL={TCP | TCPS}
	The ADDRESS_LIST parameter contains connection information for one or multiple servers, using the ADDRESS parameter to specify the primary and alternate servers. The PROTOCOL parameter is used within the ADDRESS parameter to specify the network protocol to be used. It also is used to specify whether data is encrypted and decrypted when transmitted over the network between the driver and the server.
	For example, the following entry specifies that the TCP/IP protocol will be used with no encryption:
	`(ADDRESS_LIST=`
	`   (ADDRESS= (PROTOCOL = TCP)(HOST = server1)`
	`      (PORT = 1521))`
	`   ...`
	` )`
	A port of 1521, the default port number when installing an Oracle database, is specified for server1.
	The following entry specifies that the TCP/IP protocol will be used with SSL encryption:
	`(ADDRESS_LIST=`
	`   (ADDRESS= (PROTOCOL = TCPS)(HOST = server1)`
	`      (PORT = 2484))`
	`   ...`
	` )`
	A port of 2484, the port number recommended by Oracle for SSL, is specified for server1.
	NOTE: Truststore information must still be specified using either the TrustStore and TrustStorePassword properties or Java system properties. Optionally, you can specify the ValidateServerCertificate and HostNameInCertificate properties.

Table 6-2 Oracle Driver Property Mappings to tnsnames.ora Connect Descriptor Parameters (Continued)

Oracle Driver Property	tnsnames.ora Attribute

ServerName = *`server_name`*	HOST = *`server_name`*
	The ADDRESS_LIST parameter contains connection information for one or multiple servers, using the ADDRESS parameter to specify the primary and alternate servers. The HOST parameter is used within the ADDRESS parameter to specify the server name for each server entry. The server entry can be an IP address or a server name. For example:
	`(ADDRESS_LIST=`
	`   (ADDRESS= (PROTOCOL = TCP)(HOST = server1)`
	`      (PORT = 1521))`
	`   ...`
	` )`
	The server name server1 is specified in the first server entry.

ServerType = {shared | dedicated}	SERVER = {shared | dedicated}.
	If SERVER=shared is specified in the CONNECT_DATA parameter in the tnsnames.ora file, the server process (UNIX) or thread (Windows) to be used is retrieved from a pool. For example:
	`(CONNECT_DATA=`
	`   (SERVER=shared)`
	`)`
	When SERVER=shared, this setting allows there to be fewer processes than the number of connections, reducing the need for server resources.
	When SERVER=dedicated, a server process is created to service only that connection. When that connection ends, so does the process (UNIX) or thread (Windows).

Table 6-2 Oracle Driver Property Mappings to tnsnames.ora Connect Descriptor Parameters (Continued)

Oracle Driver Property	tnsnames.ora Attribute

ServiceName = `service_name`	SERVICE_NAME = `service_name`
	
	The database service name that specifies the database used for the connection. The service name is a string that is the global database name—a name that typically comprises the database name and domain name. For example:
	
	`sales.us.acme.com`
	
	The service name is specified in the CONNECT_DATA parameter. For example:
	
	`(CONNECT_DATA=`
	`   (SERVICE_NAME=sales.us.acme.com)`
	`)`
	
	This parameter is mutually exclusive with the SID attribute.

SID = *SID*	SID = *SID*
	
	The Oracle System Identifier (SID) that refers to the instance of the Oracle database running on the server. The default Oracle SID that is configured when installing your Oracle database software is ORCL. The SID is specified in the CONNECT_DATA parameter. For example:
	
	`(CONNECT_DATA=`
	`   (SID=ORCL)`
	`)`
	
	This parameter is mutually exclusive with the SERVICE_NAME attribute.

For more information about configuring tnsnames.ora files, refer to your Oracle documentation.

Data Types

Table 6-3 lists the data types supported by the Oracle driver and describes how they are mapped to the JDBC data types.

Table 6-3 Oracle Data Types

Oracle Data Type	JDBC Data Type
BFILE	BLOB
BINARY_DOUBLE[1]	DOUBLE
BINARY_FLOAT[1]	REAL
BLOB	BLOB
CHAR	CHAR
CLOB	CLOB
DATE	TIMESTAMP
FLOAT(n)	DOUBLE
LONG	LONGVARCHAR
LONG RAW	LONGVARBINARY
NCHAR	CHAR
NCLOB	CLOB
NUMBER	DECIMAL
NUMBER (p, s)	DECIMAL
NVARCHAR2	VARCHAR
RAW	VARBINARY
TIMESTAMP[2]	TIMESTAMP
TIMESTAMP WITH LOCAL TIME ZONE[2]	TIMESTAMP
TIMESTAMP WITH TIME ZONE[2]	TIMESTAMP
UROWID[2]	VARCHAR

Table 6-3 Oracle Data Types (Continued)

Oracle Data Type	JDBC Data Type
VARCHAR2[2]	VARCHAR
XMLType[2]	CLOB

1. Supported only for Oracle 10g.
2. Supported only for Oracle 9i and higher.

See "Returning and Inserting/Updating XML Data" on page 6-35 for more information about the XMLType data type. See Appendix B, "GetTypeInfo," for a description of the data types returned by the getTypeInfo() method.

Using Oracle Date/Time Data Types

Oracle9i and higher supports the following date/time data types: TIMESTAMP, TIMESTAMP WITH LOCAL TIME ZONE, and TIMESTAMP WITH TIME ZONE. To understand how the Oracle driver supports these data types, you first must understand the values the Oracle driver assigns to the Oracle date/time session parameters.

Date/Time Session Parameters

At connection time, the Oracle driver sets the following date/time session parameters:

Session Parameter	Description
TIME_ZONE	The Oracle session time zone. The Oracle driver sets the time zone to the current time zone as reported by the JVM.
NLS_TIMESTAMP_FORMAT	The default timestamp format. The Oracle driver uses the JDBC timestamp escape format: YYYY-MM_DD HH24:MI:SS.FF
NLS_TIMESTAMP_TZ_FORMAT	The default timestamp with time zone format. The Oracle driver uses the JDBC timestamp escape format with the time zone field appended: YYYY-MM_DD HH24:MI:SS.FF TZH:TZM

TIMESTAMP Data Type

The Oracle `TIMESTAMP` data type is mapped to the JDBC `TIMESTAMP` data type.

TIMESTAMP WITH LOCAL TIME ZONE Data Type

The Oracle `TIMESTAMP WITH LOCAL TIME ZONE` data type is mapped to the `TIMESTAMP` JDBC data type.

When retrieving `TIMESTAMP WITH LOCAL TIME ZONE` columns, the value returned to the user is converted to the time zone specified by the `TIME_ZONE` session parameter.

When setting `TIMESTAMP WITH LOCAL TIME ZONE` columns:

- Using a timestamp (using `PreparedStatement.setTimestamp`, for example), the value set is converted to the time zone specified by the `TIME_ZONE` session parameter.
- Using a string (using `PreparedStatement.setString`, for example), the string is passed as-is to the server. The supplied string must be in the format specified by the `NLS_TIMESTAMP_TZ_FORMAT` session parameter. If not, the Oracle server generates an error when it attempts to convert the string to the `TIMESTAMP WITH LOCAL TIME ZONE` type.

TIMESTAMP WITH TIME ZONE Data Type

By default, the Oracle `TIMESTAMP WITH TIME ZONE` data type is mapped to the `VARCHAR` JDBC data type.

When retrieving `TIMESTAMP WITH TIME ZONE` values as a string (using `resultSet.getString`, for example), the value is returned as the string representation of the timestamp including time zone information. The string representation is formatted as:

'*YYYY-MM-DD HH24:MI:SS.FF TZH:TZM*'

where:

- *YYYY* is the 4-digit year.
- *MM* is the month.
- *DD* is the day.
- *HH24* is the hour in 24-hour format.
- *MI* is the minutes.

- *SS* is the seconds.
- *FF* is the fractional seconds.
- *TZH* is the time zone hours and *TZM* is the time zone minutes. The time zone is represented as the difference in hours and minutes between the time zone and GMT..

By default, retrieving `TIMESTAMP WITH TIME ZONE` values as a timestamp (using `resultSet.getTimeStamp`, for example) is not supported because the time zone information stored in the database would be lost when the data is converted to a timestamp. To provide backward compatibility with existing applications, you can use the `FetchTSWTZasTimestamp` property to allow `TIMESTAMP WITH TIME ZONE` values to be returned as a timestamp. The default value of the `FetchTSWTSasTimestamp` property is false, which disables retrieving `TIMESTAMP WITH TIME ZONE` values as timestamps.

When setting `TIMESTAMP WITH TIME ZONE` columns:

- Using a timestamp (using `PreparedStatement.setTimestamp()`, for example), the value set is converted to the time zone specified by the `TIME_ZONE` session parameter.
- Using a string (using `PreparedStatement.setString()`, for example), the string is passed as-is to the server. The supplied string must be in the format specified by the `NLS_TIMESTAMP_TZ_FORMAT` session parameter. If not, the Oracle server generates an error when it attempts to convert the string to the `TIMESTAMP WITH TIME ZONE` type.

Returning and Inserting/Updating XML Data

For Oracle 9i and higher, the Oracle driver supports the Oracle XMLType data type. The driver maps the Oracle XMLType data type to the JDBC CLOB data type.

Returning XML Data

The driver can return XML data as character data. For example, given a database table defined as:

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

the driver can return the XML data as character data using the following code:

```
String sql="SELECT xmlCol FROM xmlTable";
ResultSet rs=stmt.executeQuery(sql)
String charXML=rs.getString(1)
```

The result set column is described with a column type of CLOB and the column type name is xmlType.

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

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

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

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

```
ResultSet.getAsciiStream()
```

The driver converts the XML data returned from the database server from the character set encoding used by the database server to the ISO-8859-1 (latin1) encoding.

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

Inserting/Updating XML Data

When inserting to or updating XMLType columns, the data to be inserted or updated must be the XMLType data type. Oracle provides the xmltype() function to construct an XMLType data object. The xmlData argument of the xmltype() function can be specified as a string literal or a parameter marker. If specified as a parameter marker, the parameter value can be set using the following methods:

```
PreparedStatement.setString()
PreparedStatement.setCharacterStream()
PreparedStatement.setClob()
PreparedStatement.setAsciiStream()
```

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

The following code inserts data into an XMLType column using a statement with a string literal as the xmlData argument of the xmltype() function:

```
//  Insert xml data as a literal
String sql = "INSERT INTO XMLTable VALUES (1, xmltype('" +
   "<emp><empNo>123</empNo><empName>Mark</empName></emp>'))";

Statement stmt = con.createStatement();
stmt.executeUpdate(sql);
The following code inserts data into an XMLType column using a prepared
statement:
//  Insert xml data as a String parameter
String xmlStr = "<emp><empNo>234</empNo><empName>Trish</empName></emp>";
String sql = "INSERT INTO XMLTable VALUES (?, xmltype(?))";

PreparedStatement prepStmt = con.prepareStatement(sql);
prepStmt.setInt(1, 2);
prepStmt.setString(2, xmlStr);
prepStmt.executeUpdate();
```

REF CURSOR Data Type

REF CURSOR is the Oracle data type for a cursor variable. Because JDBC does not support a cursor variable data type, the Oracle driver returns REF CURSOR output parameters and return values to the application as result sets. The Oracle driver automatically converts the REF CURSOR data to a result set, which can be returned using getResultSet() or getMoreResults(). Because REF CURSOR data is returned as result sets and not as output parameters, REF CURSOR output parameters are not included in results from DatabaseMetaData.getProcedureColumns() calls.

In your application, omit any parameter markers for the REF CURSOR and do not declare an output parameter for the REF CURSOR as shown in the following examples. These examples reference the following stored procedure definition:

```
CREATE PACKAGE foo_pkg AS
   TYPE EmpCurTyp IS REF CURSOR RETURN fooTbl%ROWTYPE;
   PROCEDURE selectEmployeeManager(empId IN INT, empCursor OUT EmpCurTyp,
      mgrCursor out EmpCurTyp);
   FUNCTION selectEmployee2 (empId IN INT) return EmpCurTyp;
END foo_pkg;
Example 1: Calling a Stored Procedure That Returns a Single REF CURSOR
// Call a function that accepts an input parameter
// and returns a REF CURSOR as the return value. Omit the
// placeholder for the refcursor return value parameter.
// The REF CURSOR is returned as a result set.
sql = "{call foo_pkg.selectEmployee2(?)}";

callStmt = con.prepareCall(sql);
callStmt.setInt(1, 2);
moreResults = callStmt.execute();

while (true) {

   if (moreResults) {

      //  Get the result set that represents the REF CURSOR
      resultSet = callStmt.getResultSet();
      displayResults(resultSet);

      resultSet.close();
      resultSet = null;

      System.out.println();
   }
   else {

         updateCnt = callStmt.getUpdateCount();
         if (updateCnt == -1) {
            break;
         }
         System.out.println("Update Count: " + updateCnt);
     }
     moreResults = callStmt.getMoreResults();
}
```

Example 2: Calling a Stored Procedure that Returns Multiple REF CURSORs

```
// Call the stored procedure that accepts an input parameter
// and returns two REF CURSORs. Omit the placeholder for
// REF CURSOR parameters. The REF CURSORs are returned as
// result sets.
sql = "{call foo_pkg.selectEmployeeManager(?)}";

callStmt = con.prepareCall(sql);
callStmt.setInt(1, 2);
moreResults = callStmt.execute();

while (true) {

   if (moreResults) {

      //  Get the result set that represents the REF CURSOR
      resultSet = callStmt.getResultSet();
      displayResults(resultSet);
      resultSet.close();
   }
   else {

         updateCnt = callStmt.getUpdateCount();
         if (updateCnt == -1) {
            break;
         }
      }

      moreResults = callStmt.getMoreResults();
}
```

Authentication

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See "Authentication" on page 2-7 for an overview.

The Oracle driver supports the following methods of authentication:

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

  This method requires knowledge of how to configure your Kerberos environment and supports Windows Active Directory Kerberos and MIT Kerberos.
- NTLM authentication is a single sign-on OS authentication method for Windows environments. This method provides authentication from Windows clients only and requires minimal configuration.
- Client authentication uses the user ID of the user logged onto the system on which the driver is running to authenticate the user to the database. The Oracle database server relies on the client to authenticate the user and does not provide additional authentication.

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

Except for NTLM authentication, which provides authentication for Windows clients only, these authentication methods provide authentication when the driver is running on any supported platform.

The `AuthenticationMethod` connection property controls which authentication mechanism the driver uses when establishing connections. See "Using the AuthenticationMethod Property" on page 6-40 for information about setting the value for this property.

Using the AuthenticationMethod Property

The `AuthenticationMethod` connection property controls which authentication mechanism the driver uses when establishing connections. When `AuthenticationMethod=auto` (the default), the driver uses user ID/password, Kerberos, or NTLM authentication when establishing a connection based on the following criteria:

- If a user ID and password is specified, the driver uses user ID/password authentication when establishing a connection. The `User` property provides the user ID. The `Password` property provides the password.
- If a user ID and password is not specified and the driver is not running on a Windows platform, the driver uses Kerberos authentication when establishing a connection.

- If a user ID and password is not specified and the driver is running on a Windows platform, the driver uses NTLM authentication when establishing a connection if the driver can load the DLL required for NTLM authentication. If the driver cannot load the DLL, the driver uses Kerberos authentication.

When `AuthenticationMethod=kerberos`, the driver uses Kerberos authentication when establishing a connection. The driver ignores any values specified by the User and Password properties.

When `AuthenticationMethod=kerberosUIDPassword`, the driver first uses Kerberos when establishing a connection. Next, the driver reauthenticates the user using user ID/password authentication. The `User` property provides the user ID. The `Password` property provides the password. If a user ID and password are not specified, the driver throws an exception. If either Kerberos or user ID/password authentication fails, the connection attempt fails and the driver throws an exception.

When `AuthenticationMethod=ntlm`, the driver uses NTLM authentication when establishing a connection if the driver can load the DLL required for NTLM authentication. If the driver cannot load the DLL, the driver throws an exception. The driver ignores any values specified by the User and Password properties.

When `AuthenticationMethod=client`, the driver uses client authentication when establishing a connection. The Oracle database server relies on the client to authenticate the user and does not provide additional authentication. The driver ignores any values specified by the User and Password properties.

When `AuthenticationMethod=userIdPassword`, the driver uses user ID/password authentication when establishing a connection. The `User` property provides the user ID. The `Password` property provides the password. If a user ID is not specified, the driver throws an exception.

Configuring User ID/Password Authentication

1. Set the `AuthenticationMethod` property to auto or userIdPassword. See "Using the AuthenticationMethod Property" on page 6-40 for more information about setting a value for this property.
2. Set the `User` property to provide the user ID.
3. Set the `Password` property to provide the password.

Configuring Kerberos Authentication

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

Product Requirements

Verify that your environment meets the requirements listed in Table 6-4 before you configure the driver for Kerberos authentication.

Table 6-4 Kerberos Authentication Requirements for the Oracle Driver

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: • Oracle 10*g* (R1 and R2) • Oracle 9i (R2)
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.

Configuring the Driver

During installation, WebLogic Server installs the following files required for Kerberos authentication in the server/lib subdirectory of your WebLogic Server installation directory:

- 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 Java Authentication and Authorization Service (JAAS) login module for Kerberos authentication. This file is configured to load automatically

unless the java.security.auth.login.config system property is set to load another configuration file.

Note: Do not modify the JDBCDriverLogin.conf file.

To configure the driver:

1. Set the driver's `AuthenticationMethod` property to auto (the default) or kerberos. See "Using the AuthenticationMethod Property" on page 6-40 for more information about setting a value for this property.

2. Modify the krb5.conf file to contain your Kerberos realm name and the KDC name for that Kerberos realm by editing the file with a text editor or by specifying the system properties, java.security.krb5.realm and java.security.krb5.kdc.

   Note: In Windows Active Directory, the Kerberos realm name is the Windows domain name and the KDC name is the Windows domain controller name.

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

```
[libdefaults]
   default_realm = XYZ.COM

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

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

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

The krb5.conf file installed by WebLogic Server is configured to load automatically unless the java.security.krb5.conf system property is set to point to another Kerberos configuration file.

3. If using Kerberos authentication with a Security Manager on a Java 2 Platform, you must grant security permissions to the application and driver. See "Permissions for Kerberos Authentication" on page 2-19 for an example.

Specifying User Credentials for Kerberos Authentication (Delegation of Credentials)

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

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 jjavax.security.auth.Subject used for authentication as shown in the following example.

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

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

LoginContext lc = null;
Subject subject = null;

try {

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

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

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

    public Object run() {

        Connection con = null;
    try {

         Class.forName("com.ddtek.jdbc.oracle.OracleDriver");
         String url = "jdbc:bea:oracle://myServer:1521";
         con = DriverManager.getConnection(url);
        }
     catch (Exception except) {

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

        return con;
    }
});

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

... // do something with the results
```

Obtaining a Kerberos Ticket Granting Ticket

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

If the application uses Kerberos authentication from a Windows client and the Kerberos authentication is provided by Windows Active Directory, the application user is not required to log onto the Kerberos server and explicitly obtain a TGT. Windows Active Directory automatically obtains a TGT for the user.

The application user must explicitly obtain a TGT in the following cases:

- If the application uses Kerberos authentication from a UNIX or Linux client
- If the application uses Kerberos authentication from a Windows client and Kerberos authentication is provided by MIT Kerberos

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

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

where *`user`* is the application user.

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

Configuring NTLM Authentication

This section provides requirements and instructions for configuring NTLM authentication for the Oracle driver.

Product Requirements

Verify that your environment meets the requirements listed in Table 6-5 before you configure the driver for NTLM authentication.

Table 6-5 NTLM Authentication Requirements for the Oracle Driver

Component	Requirements
Database server	The database server must be administered by the same domain controller that administers the client and must be running one of the following databases: • Oracle 10*g* (R1 and R2) • Oracle 9i (R1 and R2)
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 1 or higher • Windows 2000 Service Pack 4 or higher • Windows NT 4.0 In addition, J2SE 1.3 or higher must be installed.

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 or ntlm. See“Using the AuthenticationMethod Property” on page 6-40 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:datadirect:oracle://server3:1521;
     ServiceName=ORCL;LoadLibraryPath=C:\DataDirect\lib;
     User=test;Password=secret
     ```
3. If using NTLM authentication with a Security Manager on a Java 2 Platform, security permissions must be granted to allow the driver to establish connections. See “Permissions for Establishing Connections” on page 2-16 for an example.

Configuring Client Authentication

Set the `AuthenticationMethod` property to client. See “Using the AuthenticationMethod Property” on page 6-40 for more information about setting a value for this property.

Data Encryption

The Oracle driver supports SSL for data encryption. SSL secures the integrity of your data by encrypting information and providing authentication. See “Data Encryption Across the Network” on page 2-11 for an overview.

See “Using tnsnames.ora Files” on page 6-25 for information about configuring a tnsnames.ora file for SSL encryption.

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

To configure SSL encryption:

1. Set the `EncryptionMethod` property to SSL.
2. Specify the location and password of the truststore file used for SSL server authentication. Either set the TrustStore and TrustStore properties or their corresponding Java system properties (javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword, respectively).
3. To validate certificates sent by the database server, set the `ValidateServerCertificate` property to true.
4. Optionally, set the `HostNameInCertificate` property to a host name to be used to validate the certificate. The `HostNameInCertificate` property provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested.
5. If your database server is configured for SSL client authentication, configure your keystore information:
    a. Specify the location and password of the keystore file. Either set the KeyStore and KeyStore properties or their corresponding Java system properties (javax.net.ssl.keyStore and javax.net.ssl.keyStorePassword, respectively).
    b. If any key entry in the keystore file is password-protected, set the `KeyPassword` property to the key password.

SQL Escape Sequences

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

Isolation Levels

The Oracle driver supports the `Read Committed` and `Serializable` isolation levels. The default is `Read Committed`.

Using Scrollable Cursors

The Oracle driver supports scroll-sensitive result sets, scroll-insensitive result sets, and updatable result sets.

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

Batch Inserts and Updates

The Oracle driver provides two mechanisms for supporting batch operations:

- The first mechanism uses native Oracle batch functionality. This mechanism typically is the faster of the two mechanisms, but it is not compliant with the JDBC specification because the native Oracle functionality returns a single update count for all operations in the batch. Because that single update count cannot be resolved into individual update counts for the driver, the driver returns a value of SUCCESS_NO_INFO (-2) for each entry in the update count array. The JDBC specification requires individual update counts to be returned for each operation in the batch.
- The second mechanism uses code that resides in the driver to execute the batch operations and complies with the JDBC specification, but it is slower than using native Oracle batch functionality.

The `BatchPerformanceWorkaround` property determines which batch mechanism is used. If the value of the `BatchPerformanceWorkaround` property is true, the native Oracle batch mechanism is used; otherwise, the JDBC-compliant mechanism is used. The default value of the `BatchPerformanceWorkaround` property is false.

Parameter Metadata Support

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

Insert and Update Statements

The Oracle driver supports returning parameter metadata for the following forms of Insert and Update statements:

- `INSERT INTO foo VALUES (?, ?, ?)`
- `INSERT INTO foo (col1, col2, col3) VALUES (?, ?, ?)`

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

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

Select Statements

The Oracle driver supports returning parameter metadata for Select statements that contain parameters in ANSI SQL 92 entry-level predicates, for example, such as COMPARISON, BETWEEN, IN, LIKE, and EXISTS predicate constructs. Refer to the ANSI SQL reference for detailed syntax.

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

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

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

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

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

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

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

```
SELECT col1, col2 FROM foo WHERE col1 = ? and col2 > ?
SELECT ... WHERE colname = (SELECT col2 FROM t2 WHERE col3 = ?)
SELECT ... WHERE colname LIKE ?
SELECT ... WHERE colname BETWEEN ? and ?
SELECT ... WHERE colname IN (?, ?, ?)
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE col1 < ?)
```

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

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

Joins are supported. For example:

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

Fully qualified names and aliases are supported. For example:

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

When parameter metadata is requested for a column defined as NUMBER with no precision and scale argument, the driver returns a precision of 0 and a scale of 0 to indicate that the precision and scale of the column are unknown.

Stored Procedures

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

ResultSet MetaData Support

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

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

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

```
SELECT id, name FROM Employee

SELECT E.id, E.name FROM Employee E

SELECT E.id, E.name AS EmployeeName FROM Employee E

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

```
SELECT id, name, location, phone FROM Employee, EmployeeInfo WHERE id =
   empId
```

```
SELECT Employee.id, Employee.name, EmployeeInfo.location,
   EmployeeInfo.phone FROM Employee, EmployeeInfo WHERE Employee.id =
   EmployeeInfo.id
```

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

```
SELECT E.id, E.name as EmployeeName, {fn UCASE(E.name)}

   AS upper FROM Employee E
```

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

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

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

Executing Insert/Update/Delete Statements with a RETURNING Clause

For Oracle 8.1.6 and higher, the Oracle driver supports executing Insert, Update, and Delete statements with the RETURNING clause, which allows your application to return inserted, updated, or deleted values of a row into a variable and eliminate the need to execute additional statements to return this information.

The driver returns the values for each column named in the RETURNING clause as an output parameter. Your application must execute the Insert, Update, or Delete statement with the RETURNING clause using a CallableStatement object. In addition, your application must specify the data type of each returned value using the CallableStatement.registerOutParameter() method. The registered data type for a returned value must match the data type of the database column. For example, if the database column is defined with a JDBC type of CHAR, the data type of the returned value for that column must be registered as Types.CHAR.

The RETURNING clause can return a single row or multiple rows. The method your application uses to retrieve the values of returned columns depends on the number of rows the RETURNING clause returns as shown in the following examples.

Example A: Retrieving a Result Value From an Insert/Update/Delete of a Single Row

Given the table defined by:

```
CREATE TABLE employees (id int, name varchar(30))
```

You can use the following Insert statement with the RETURNING clause to return the updated ID for Smith:

```
String sql = "INSERT INTO employees VALUES(100, 'Smith')
   RETURNING id INTO ?";
CallableStatement callStmt = con.prepareCall(sql);
callStmt.registerOutParameter(1, Types.INTEGER);
int updateCnt = callStmt.executeUpdate();
int newId = callStmt.getInt(1);
System.out.println("The id of the inserted row is: " + newId);
```

The database server returns a single result value for the requested column. An application can retrieve the result value using any of the following CallableStatement methods: getInt(), getString(), getObject(), and so on. The object type returned by getObject() is based on the data type specified in the registerOutParameter() call for the returned columns. Refer to the JDBC specification for details about JDBC data type to Java object mapping.

Example B: Retrieving Result Values From an Insert/Update/Delete of Multiple Rows

Given the table defined by:

```
CREATE TABLE employees (id int, name varchar(30))
```

You can use the following Update statement with the RETURNING clause to return all rows with an updated ID value.

```
String sql = "UPDATE employees SET id = id + 1000" +
                "RETURNING id INTO ?";
CallableStatement callStmt = con.prepareCall(sql);
callStmt.registerOutParameter(1, Types.INTEGER);
int updateCnt = callStmt.executeUpdate();
Integer[] newIds = (int []) callStmt.getArray(1).getArray();
```

```
for (int index = 0; index < newIds.length; index++) {
    System.out.println("New Id value: " + newIds[index]);
}
```

The database server returns multiple result values for the requested column. An application can retrieve the result values using the CallableStatement.getArray() method.

Note: If you use the CallableStatement.getxxx() methods to retrieve the result values, the driver only returns the first result value for the requested column.

The data type of the returned array, and the data type of the array elements, match the data type specified in the registerOutParameter() call for the returned column. The elements of the array are an object type. For example, if the application registered the data type of the returned value as Types.INTEGER, the elements of the array are returned as Integer objects. The result set generated by the CallableStatement.getArray() method is a forward-only result set with a result set concurrency of read only. It contains a single column and has a row for each entry in the array.

Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

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.

Auto-Generated Keys Support

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

An application can return values of auto-generated keys when it executes an Insert statement. How you return these values depends on whether you are using an Insert statement that contains parameters:

- When using an `Insert` statement that contains no parameters, the Oracle driver supports the following form of the `Statement.execute()` and `Statement.executeUpdate()` methods to instruct the driver to return values of auto-generated keys:
  - `Statement.execute (String` *`sql`*`, int` *`autoGeneratedKeys`*`)`
  - `Statement.execute(String` *`sql`*`, int[]` *`columnIndexes`*`)`
  - `Statement.execute(String` *`sql`*`, String[]` *`columnNames`*
  - `Statement.executeUpdate (String` *`sql`*`, int` *`autoGeneratedKeys`*`)`
  - `Statement.executeUpdate(String` *`sql`*`, int[]` *`columnIndexes`*`)`
  - `Statement.executeUpdate(String` *`sql`*`, String[]` *`columnNames`*`)`
- When using an `Insert` statement that contains parameters, the Oracle driver supports the following form of the `Connection.prepareStatement()` method to instruct the driver to return values of auto-generated keys:
  - `Connection.prepareStatement(String` *`sql`*`, int` *`autoGeneratedKeys`*`)`
  - `Connection.prepareStatement(String` *`sql`*`, int[]` *`columnIndexes`*`)`
  - `Connection.prepareStatement(String` *`sql`*`, String[]` *`columnNames`*`)`

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.

Server Result Set Caching

The Oracle driver now supports server result set caching, a feature introduced in Oracle 11g that allows query results to be cached in memory. To specify that query results be cached in memory, use a result set cache hint in the query. For example:

```
SELECT /*+ result_cache */ * FROM employees
```

For more information about server result set caching, refer to your Oracle 11g documentation.

XQuery Support

The Oracle driver supports the XQuery functions supported by Oracle 11g.

Refer to your Oracle documentation for more information about Oracle’s XQuery support.

CHAPTER 7

The Sybase Driver

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

- “Database Version Support” on page 7-2
- “Driver Classes” on page 7-2
- “Sybase URL” on page 7-2
- “Sybase Connection Properties” on page 7-3
- “Performance Considerations” on page 7-18
- “Data Types” on page 7-20
- “Authentication” on page 7-22
- “Data Encryption” on page 7-28
- “SQL Escape Sequences” on page 7-29
- “Isolation Levels” on page 7-29
- “Using Scrollable Cursors” on page 7-29
- “Large Object (LOB) Support” on page 7-29
- “Batch Inserts and Updates” on page 7-30
- “Parameter Metadata Support” on page 7-30

Database Version Support

The WebLogic Type 4 JDBC driver for Sybase (the "Sybase driver") supports the following database versions:

- Sybase Adaptive Server Enterprise 15.0
- Sybase Adaptive Server Enterprise 12.0, 12.5, 12.5.1, 12.5.2, 12.5.3, and 12.5.4
- Sybase Adaptive Server 11.5 and 11.9

Note: XA connections are supported with the Sybase Adaptive Server Enterprise 12.0 and later versions only. XA connections are not supported on Sybase Adaptive Server 11.5 and 11.9.

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.

Sybase URL

The connection URL format for the Sybase driver is:

```
jdbc:bea:sybase://hostname:port[;property=value[;...]]
```

where:

- `hostname` is the TCP/IP address or TCP/IP host name of the server to which you are connecting. See "Using IP Addresses" on page 2-5 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"Sybase Connection Properties" on page 7-3.

For example:

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

Sybase Connection Properties

Table 7-1 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 7-1 Sybase Connection Properties

Property	Description
AuthenticationMethod	{kerberos | userIdPassword}. Determines which authentication method the driver uses when establishing a connection. If set to 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 set to 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 "Authentication" on page 7-22 for more information about using authentication with the Sybase driver. The default is userIdPassword.
BatchPerformanceWorkaround OPTIONAL	{true | false}. Determines the method used to execute batch operations. If set to true, the driver uses the native Sybase batch mechanism. In most cases, using the native Sybase batch functionality provides significantly better performance, but the driver may not always be able to return update counts for the batch. If set to false (the default), the driver uses the JDBC 3.0-compliant batch mechanism. The default is false. See "Batch Inserts and Updates" on page 7-30.
CodePageOverride OPTIONAL	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.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description
ConnectionRetryCount OPTIONAL	The number of times the driver retries connections to a database server until a successful connection is established. Valid values are 0 and any positive integer. If set to 0, the driver does not try to reconnect after the initial unsuccessful attempt. If an application sets a login timeout value (for example, using DataSource.loginTimeout), the login timeout takes precedence over this property. The `ConnectionRetryDelay` property specifies the wait interval, in seconds, used between attempts. The default is 5.
ConnectionRetryDelay OPTIONAL	The number of seconds the driver waits before retrying connections to a database server when ConnectionRetryCount is set to a positive integer. The default is 1.
ConvertNull	{1 | 0}. Controls how data conversions are handled for null values. If set to 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. 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. The default is 1.
DatabaseName OPTIONAL	The name of the database to which you want to connect. See also "Database Connection Property" on page 7-34.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description
EnableCancelTimeout	{true | false}. 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. If set to true, the cancel request times out using the same timeout value, in seconds, that is set for the statement it cancels. For example, if your application calls Statement.setQueryTimeout(5) on a statement and that statement is cancelled because its timeout value was exceeded, a cancel request is sent that also will time out if its execution exceeds 5 seconds. If the cancel request times out, because the server is down, for example, the driver throws an exception indicating that the cancel request was timed out and the connection is no longer valid. If set to false (the default), the cancel request does not time out.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description

EncryptionMethod	{noEncryption | SSL}. Determines whether SSL encryption is used to encrypt and decrypt data transmitted over the network between the driver and database server.
	
	If set to noEncryption (the default), 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 set to SSL, data is encrypted using SSL. If the database server does not support SSL, the connection fails and the driver throws an exception. When SSL is enabled, the following properties also apply:
	
	`HostNameInCertificate`
	
	`TrustStore`
	
	`TrustStorePassword`
	
	`ValidateServerCertificate`
	
	NOTE: Connection hangs can occur if the driver attempts to connect to a database server that does not support SSL. You may want to set a login timeout using the `LoginTimeout` property to avoid problems when connecting to a server that does not support SSL.
	
	See "Data Encryption" on page 7-28 for more information about configuring data encryption.
	
	See "Performance Considerations" on page 7-18 for information about configuring this property for optimal performance.
	
	The default is noEncryption.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description
ErrorBehavior	{exception | warning | raiseerrorwarning}. Determines how the driver handles errors returned from stored procedures. If set to exception (the default), the driver throws an exception when it encounters stored procedure errors, including RAISERRORs. If set to warning, the driver returns stored procedure errors, including RAISERRORs, as SQLWarnings. If set to raiseerrorwarning, the driver returns RAISERRORs as SQLWarnings and throws exceptions for other stored procedure errors. By default, previous versions of the Sybase driver converted errors returned from a stored procedure into SQLWarnings. Applications that relied on the driver converting errors to warnings can revert to that behavior by setting `ErrorBehavior=warning`. The default is exception.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description
HostNameInCertificate	{host_name | #SERVERNAME#}. 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. 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 or if the certificate does not have a SubjectAlternativeName, the driver compares the host name with the Common Name (CN) part of the certificate's Subject name. If the values do not match, the connection fails and the driver throws an exception. If #SERVERNAME# is specified, the driver compares the server name specified in the connection URL or data source of the connection to the DNSName value of the SubjectAlternativeName in the certificate. If a DNSName value does not exist or if the certificate does not have a SubjectAlternativeName, 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 uses the server name specified in the connection URL or data source of the connection to validate the certificate. If SSL encryption or certificate validation is not enabled, this property is ignored. See "Data Encryption" on page 7-28 for information about configuring for authentication. The default is an empty string.

Table 7-1 Sybase Connection Properties (Continued)

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. The following connection URL sets the handling of null values to the Sybase default:
	`jdbc:bea:sybase://server1:5000;`
	`InitializationString=set ANSINULL off;`
	`DatabaseName=test`
	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. The following connection URL sets the handling of null values to the Sybase default and allows delimited identifiers:
	`jdbc:bea:sybase://server1:5000;`
	`InitializationString=(set ANSINULL off;`
	`set QUOTED_IDENTIFIER on);DatabaseName=test`
	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.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description
InsensitiveResultSetBufferSize OPTIONAL	{-1 | 0 | *x*}. Determines the amount of memory used by the driver to cache insensitive result set data. It must have one of the following values: 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. The default is 2048 (KB). See "Performance Considerations" on page 7-18 for information about configuring this property for optimal performance.
JavaDoubleToString	{true | 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.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description
LoginTimeout OPTIONAL	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. If set to 0 (the default), the driver does not time out a connection request.
LongDataCacheSize	{-1 | 0 | x}. 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. 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 "Performance Considerations" on page 7-18 for information about configuring this property for optimal performance. The default is 2048.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description
PacketSize	PacketSize={0 | x}. 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. If set to 0, the driver uses the default maximum packet size used by the database server. If set to x, an integer from 1 to 1024, 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 SSL encryption is enabled using the `EncryptionMethod` property, any value set for the `PacketSize` property is ignored. See “Performance Considerations” on page 7-18 for information about configuring this property for optimal performance. The default is 0.
Password	The case-sensitive 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.
PortNumber	The TCP port of the primary database server that is listening for connections to the Sybase database. The default varies depending on operating system. This property is supported only for data source connections.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description
PrepareMethod OPTIONAL	{StoredProc | StoredProclfParam | Direct}. Determines whether stored procedures are created on the server for prepared statements. If set to StoredProc, a stored procedure is created when the statement is prepared and is executed when the prepared statement is executed. If set to 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 set to Direct, a stored procedure is not created for the prepared statement and the statement is executed directly. A stored procedure may be created if parameter metadata is requested. The default is StoredProclfParam. 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. See "Performance Considerations" on page 7-18 for information about configuring this property for optimal performance.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description
QueryTimeout	{*positive integer* | -1 | 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.
ResultSetMetaDataOptions	{0 | 1}. The Sybase 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 “ResultSet MetaData Support” on page 7-30 for more information about returning ResultSet metadata.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description
SelectMethod OPTIONAL	{Direct | Cursor}. 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. 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. 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 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 "Performance Considerations" on page 7-18 for information about configuring this property for optimal performance. The default is Direct.
ServerName	Specifies either the IP address or the server name (if your network supports named servers) of the primary database server. For example, 122.23.15.12 or SybaseServer. This property is supported only for data source connections.

Table 7-1 Sybase Connection Properties (Continued)

Property	Description

ServicePrincipalName	Specifies the case-sensitive service principal name to be used by the driver for Kerberos authentication. For Sybase, the service principal name is the name of a server configured in your Sybase interfaces file. If you set this property, you also must set the value of the `AuthenticationMethod` property to Kerberos.
	
	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`
	
	and
	
	`server/sybase125ase1`
	
	When Kerberos authentication is not used, this property is ignored.
	
	See “Authentication” on page 7-22 for more information about using authentication with the Sybase driver.

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`.
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`.
User	The case-insensitive user name used to connect to your Sybase database. A user name is required only if security is enabled on your database. If so, contact your system administrator to get your user name.

Performance Considerations

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

- "BatchPerformanceWorkaround" on page 7-18
- "EncryptionMethod" on page 7-18
- "InsensitiveResultSetBufferSize" on page 7-19
- "LongDataCacheSize" on page 7-19
- "PacketSize" on page 7-19
- "PrepareMethod" on page 7-19
- "ResultSetMetaDataOptions" on page 7-19
- "SelectMethod" on page 7-20

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.

EncryptionMethod

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

InsensitiveResultSetBufferSize

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

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.

PacketSize

Typically, it is optimal for the client to use the maximum packet size that the server allows. This reduces the total number of round trips required to return data to the client, thus improving performance. Therefore, performance can be improved if this property is set to the maximum packet size of the database server.

PrepareMethod

If your application executes prepared statements multiple times, this property should be set to `StoredProc` to improve performance because, once created, executing a stored procedure is faster than executing a single SQL Statement. If your application does not execute prepared statements multiple times, this property should be set to `Direct`. In this case, performance decreases if a stored procedure is created because a stored procedure incurs more overhead on the server than executing a single SQL statement.

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 "ResultSet MetaData Support" on page 7-30 for more information about returning ResultSet metadata.

SelectMethod

In most cases, using server-side database cursors impacts performance negatively. However, if the following statements are true for your application, the best setting for this property is cursor, which means use server-side database cursors:

- Your application contains queries that return large amounts of data.
- Your application executes a SQL statement before processing or closing a previous large result set and does this multiple times.
- Large result sets returned by your application use forward-only cursors.

Data Types

Table 7-2 lists the data types supported by the Sybase driver and how they are mapped to JDBC data types.

Table 7-2 Sybase Data Types

Sybase Data Type	JDBC Data Type
BIGINT[1]	BIGINT
BINARY	BINARY
BIT	BIT
CHAR	CHAR
DATE[2]	DATE
DATETIME	TIMESTAMP
DECIMAL	DECIMAL
FLOAT	FLOAT
IMAGE	LONGVARBINARY

Table 7-2 Sybase Data Types (Continued)

Sybase Data Type	JDBC Data Type
INT	INTEGER
MONEY	DECIMAL
NCHAR	CHAR
NUMERIC	NUMERIC
NVARCHAR	VARCHAR
REAL	REAL
SMALLDATETIME	TIMESTAMP
SMALLINT	SMALLINT
SMALLMONEY	DECIMAL
SYSNAME	VARCHAR
TEXT	LONGVARCHAR
TIME[2]	TIME
TIMESTAMP	VARBINARY
TINYINT	TINYINT
UNICHAR[2]	CHAR
UNITEXT[1]	LONGVARCHAR
UNIVARCHAR[2]	VARCHAR
UNSIGNED BIGINT[1]	DECIMAL
UNSIGNED INT[1]	BIGINT
UNSIGNED SMALLINT[1]	INTEGER
VARBINARY	VARBINARY
VARCHAR	VARCHAR

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

Note: FOR USERS OF SYBASE 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.

Authentication

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See "Authentication" on page 2-7 for an overview.

The Sybase driver supports the following methods of authentication:

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

  This method requires knowledge of how to configure your Kerberos environment and supports Windows Active Directory Kerberos and MIT Kerberos.

The driver's `AuthenticationMethod` connection property controls which authentication mechanism the driver uses when establishing connections. See "Using the AuthenticationMethod Property" on page 7-22 for information about setting the value for this property.

Using the AuthenticationMethod Property

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

When `AuthenticationMethod=kerberos`, the driver uses Kerberos authentication when establishing a connection. The driver ignores any values specified by the User and Password properties.

When `AuthenticationMethod=userIdPassword` (the default), the driver uses user ID/password authentication when establishing a connection. The `User` property provides the user ID. The `Password` property provides the password. If a user ID is not specified, the driver throws an exception.

Configuring User ID/Password Authentication

1. Set the `AuthenticationMethod` property to userIdPassword. See "Using the AuthenticationMethod Property" on page 7-22 for more information about setting a value for this property.
2. Set the `User` property to provide the user ID.
3. Set the `Password` property to provide the password.

Configuring Kerberos Authentication

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

Product Requirements

Verify that your environment meets the requirements listed in Table 7-3 before you configure the driver for Kerberos authentication.

Table 7-3 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.

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 "Using the AuthenticationMethod Property" on page 7-22 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:[BEA][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“Permissions for Kerberos Authentication” on page 2-19 for an example.

Specifying User Credentials for Kerberos Authentication (Delegation of Credentials)

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

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:bea:sybase://myServer:5000";
         con = DriverManager.getConnection(url);
        }
     catch (Exception except) {

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

        return con;
    }
});

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

... // do something with the results
```

Obtaining a Kerberos Ticket Granting Ticket

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

If the application uses Kerberos authentication from a Windows client and the Kerberos authentication is provided by Windows Active Directory, the application user is not required to log onto the Kerberos server and explicitly obtain a TGT. Windows Active Directory automatically obtains a TGT for the user.

The application user must explicitly obtain a TGT in the following cases:

- If the application uses Kerberos authentication from a UNIX or Linux client
- If the application uses Kerberos authentication from a Windows client and Kerberos authentication is provided by MIT Kerberos

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

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

where user is the application *`user`*.

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

Data Encryption

The Sybase driver supports SSL for data encryption. SSL secures the integrity of your data by encrypting information and providing authentication. See "Data Encryption Across the Network" on page 2-11 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.

SQL Escape Sequences

See "SQL Escape Sequences for JDBC" on page C-1 for information about the SQL escape sequences supported by the Sybase driver.

Isolation Levels

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

Using Scrollable Cursors

The Sybase driver supports scroll-sensitive result sets only on result sets returned from tables created with an identity column. The Sybase driver also supports scroll-insensitive result sets and updatable result sets.

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

Large Object (LOB) Support

Although Sybase does not define a `Blob` or `Clob` data type, the Sybase driver allows you to return and update long data, specifically `LONGVARBINARY` and `LONGVARCHAR` data, using JDBC methods designed for Blobs and Clobs. When using these methods to update long data as Blobs or Clobs, the updates are made to the local copy of the data contained in the `Blob` or `Clob` object.

Retrieving and updating long data using JDBC methods designed for Blobs and 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.

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. For more information about specifying connection properties, see "Sybase Connection Properties" on page 7-3.

Parameter Metadata Support

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

ResultSet MetaData Support

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

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

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

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

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

```
SELECT E.id, E.name as EmployeeName, {fn UCASE(E.name)}

   AS upper FROM Employee E
```

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

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

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

Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

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.

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.

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

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

  ```
  SET ANSINULL OFF
  ```

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 "Set Up the Sybase Server for XA Support" in *Programming WebLogic JTA.*

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:bea:sybase://server1:1433;DatabaseName=jdbc;Database=acct;
User=test;Password=secret
```

JDBC Support

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

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

Supported Functionality

The following tables list functionality supported for each JDBC object.

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.

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.
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.

Table A-3 Blob Object (Continued)

Blob Object Methods	Version Introduced	Supported	Comments
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.

Table A-3 Blob Object (Continued)

Blob Object Methods	Version Introduced	Supported	Comments
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.
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.

Table A-3 Blob Object (Continued)

Blob Object Methods	Version Introduced	Supported	Comments
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.

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.
BigDecimal getBigDecimal (int)	2.0 Core	Yes	
BigDecimal getBigDecimal (int, int)	1.0	Yes	
BigDecimal getBigDecimal (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Blob getBlob (int)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.

Table A-4 CallableStatement Object (Continued)

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
Blob getBlob (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
boolean getBoolean (int)	1.0	Yes	
boolean getBoolean (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
byte getByte (int)	1.0	Yes	
byte getByte (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
byte [] getBytes (int)	1.0	Yes	
byte [] getBytes (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Clob getClob (int)	2.0 Core	Yes	
Clob getClob (String)	3.0	Yes	Supported for the SQL Server driver only using with data types that map to the JDBC LONGVARCHAR data type. All other drivers throw "unsupported method" exception.
Date getDate (int)	1.0	Yes	
Date getDate (int, Calendar)	2.0 Core	Yes	
Date getDate (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.

Table A-4 CallableStatement Object (Continued)

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
Date getDate (String, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
double getDouble (int)	1.0	Yes	
double getDouble (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
float getFloat (int)	1.0	Yes	
float getFloat (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
int getInt (int)	1.0	Yes	
int getInt (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
long getLong (int)	1.0	Yes	
long getLong (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Object getObject (int)	1.0	Yes	
Object getObject (int, Map)	2.0 Core	Yes	Map ignored.
Object getObject (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Object getObject (String, Map)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception. Map ignored.

Table A-4 CallableStatement Object (Continued)

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
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	
short getShort (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
String getString (int)	1.0	Yes	
String getString (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Time getTime (int)	1.0	Yes	
Time getTime (int, Calendar)	2.0 Core	Yes	
Time getTime (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Time getTime (String, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Timestamp getTimestamp (int)	1.0	Yes	
Timestamp getTimestamp (int, Calendar)	2.0 Core	Yes	
Timestamp getTimestamp (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.

Table A-4 CallableStatement Object (Continued)

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
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.
void registerOutParameter (int, int)	1.0	Yes	
void registerOutParameter (int, int, int)	1.0	Yes	
void registerOutParameter (int, int, String)	2.0 Core	Yes	String/typename ignored.
void registerOutParameter (String, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void registerOutParameter (String, int, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void registerOutParameter (String, int, String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception. String/typename ignored.
void setArray (int, Array)	2.0 Core	No	Throws “unsupported method” exception.
void setAsciiStream (String, InputStream, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.

Table A-4 CallableStatement Object (Continued)

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
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, int)	3.0	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 setDate (String, Date)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setDate (String, Date, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setDouble (String, double)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setFloat (String, float)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.

Table A-4 CallableStatement Object (Continued)

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
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 setNull (String, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw “unsupported method” exception
void setNull (String, int, String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw “unsupported method” exception.
void setObject (String, Object)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw “unsupported method” exception.
void setObject (String, Object, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw “unsupported method” exception.
void setObject (String, Object, int, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw “unsupported method” exception.
void setShort (String, short)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw “unsupported method” exception.
void 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.

Table A-4 CallableStatement Object (Continued)

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
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	

Clob Object

Table A-5 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.
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.

Table A-5 Clob Object (Continued)

Clob Object (Continued) Methods	Version Introduced	Supported	Comments
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.

Table A-5 Clob Object (Continued)

Clob Object (Continued) Methods	Version Introduced	Supported	Comments
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.
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.

Connection Object

Table A-6 Connection Object

Connection Object Methods	Version Introduced	Supported	Comments
void clearWarnings ()	1.0	Yes	
void close ()	1.0	Yes	When a connection is closed while a transaction is still active, that transaction is rolled back.
void commit ()	1.0	Yes	
Statement createStatement ()	1.0	Yes	
Statement createStatement (int, int)	2.0 Core	Yes	ResultSet.TYPE_SCROLL_ SENSITIVE downgraded to TYPE_SCROLL_INSENSITIVE for the DB2 driver.
Statement createStatement (int, int, int)	3.0	No	Throws "unsupported method" exception.
boolean getAutoCommit ()	1.0	Yes	

Table A-6 Connection Object (Continued)

Connection Object Methods	Version Introduced	Supported	Comments
String getCatalog ()	1.0	Yes	Supported for all drivers except Oracle, which does not have the concept of a catalog. The Oracle driver returns an empty string.
String getClientInfo ()	4.0	Yes	
String getClientInfo (String)	4.0	Yes	
int getHoldability ()	3.0	Yes	
DatabaseMetaData getMetaData ()	1.0	Yes	
int getTransactionIsolation ()	1.0	Yes	
Map getTypeMap ()	2.0 Core	Yes	Always returns empty java.util.HashMap.
SQLWarning getWarnings ()	1.0	Yes	
boolean isClosed ()	1.0	Yes	
boolean isReadOnly ()	1.0	Yes	
boolean isValid ()	4.0	Yes	
String nativeSQL (String)	1.0	Yes	Always returns same String as passed in.
CallableStatement prepareCall (String)	1.0	Yes	
CallableStatement prepareCall (String, int, int)	2.0 Core	Yes	ResultSet.TYPE_SCROLL_ SENSITIVE downgraded to TYPE_SCROLL_INSENSITIVE for the DB2 driver.

Table A-6 Connection Object (Continued)

Connection Object Methods	Version Introduced	Supported	Comments
CallableStatement prepareCall (String, int, int, int)	3.0	No	Throws "unsupported method" exception.
PreparedStatement prepareStatement (String)	1.0	Yes	
PreparedStatement prepareStatement (String, int)	3.0	Yes	
PreparedStatement prepareStatement (String, int, int)	2.0 Core	Yes	ResultSet.TYPE_SCROLL_ SENSITIVE downgraded to TYPE_SCROLL_INSENSITIVE for the DB2 driver.
PreparedStatement prepareStatement (String, int, int, int)	3.0	No	Throws "unsupported method" exception.
PreparedStatement prepareStatement (String, int[])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws “unsupported method” exception.
PreparedStatement prepareStatement (String, String [])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws “unsupported method” exception.
void releaseSavepoint (Savepoint)	3.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
void rollback ()	1.0	Yes	

Table A-6 Connection Object (Continued)

Connection Object Methods	Version Introduced	Supported	Comments
void rollback (Savepoint)	3.0	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
void setAutoCommit (boolean)	1.0	Yes	
void setCatalog (String)	1.0	Yes	Supported for all drivers except Oracle, which does not have the concept of a catalog. The Oracle driver ignores any value set by the catalog parameter.
String setClientInfo (Properties)	4.0	Yes	
String setClientInfo (String, String)	4.0	Yes	
void setHoldability (int)	3.0	Yes	Holdability parameter value is ignored.
void setReadOnly (boolean)	1.0	Yes	
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.

Table A-6 Connection Object (Continued)

Connection Object Methods	Version Introduced	Supported	Comments
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	
void setTypeMap (Map)	2.0 Core	Yes	Ignored.

DatabaseMetaData Object

Table A-7 DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
boolean allProceduresAreCallable ()	1.0	Yes	
boolean allTablesAreSelectable ()	1.0	Yes	
boolean dataDefinitionCausesTransaction Commit ()	1.0	Yes	
boolean dataDefinitionIgnoredInTransactions ()	1.0	Yes	
boolean deletesAreDetected (int)	2.0 Core	Yes	
boolean doesMaxRowSizeIncludeBlobs ()	1.0	Yes	Not supported by the SQL Server and Sybase drivers.
getAttributes (String, String, String, String)	3.0	Yes	Empty result set is returned.

Table A-7 DababaseMetaData Object (Continued)

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
ResultSet getAttributes (String, String, String, String)	3.0	No	Throws “unsupported method” exception.
ResultSet getBestRowIdentifier (String, String, String, int, boolean)	1.0	Yes	
ResultSet getCatalogs ()	1.0	Yes	
String getCatalogSeparator ()	1.0	Yes	
String getCatalogTerm ()	1.0	Yes	
String getClientInfoProperties ()	4.0	Yes	
ResultSet getColumnPrivileges (String, String, String, String)	1.0	Yes	
ResultSet getColumns (String, String, String, String)	1.0	Yes	
Connection getConnection ()	2.0 Core	Yes	
ResultSet getCrossReference (String, String, String, String, String, String)	1.0	Yes	
int getDatabaseMajorVersion ()	3.0	Yes	
int getDatabaseMinorVersion ()	3.0	Yes	
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	

Table A-7 DababaseMetaData Object (Continued)

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

Table A-7 DababaseMetaData Object (Continued)

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
int getMaxConnections ()	1.0	Yes	
int getMaxCursorNameLength ()	1.0	Yes	
int getMaxIndexLength ()	1.0	Yes	
int getMaxProcedureNameLength ()	1.0	Yes	
int getMaxRowSize ()	1.0	Yes	
int getMaxSchemaNameLength ()	1.0	Yes	
int getMaxStatementLength ()	1.0	Yes	
int getMaxStatements ()	1.0	Yes	
int getMaxTableNameLength ()	1.0	Yes	
int getMaxTablesInSelect ()	1.0	Yes	
int getMaxUserNameLength ()	1.0	Yes	
String getNumericFunctions ()	1.0	Yes	
ResultSet getPrimaryKeys (String, String, String)	1.0	Yes	
ResultSet getProcedureColumns (String, String, String, String)	1.0	Yes	
ResultSet getProcedures (String, String, String)	1.0	Yes	
String getProcedureTerm ()	1.0	Yes	
int getResultSetHoldability ()	3.0	Yes	
ResultSet getSchemas ()	1.0	Yes	
String getSchemaTerm ()	1.0	Yes	
String getSearchStringEscape ()	1.0	Yes	
String getSQLKeywords ()	1.0	Yes	

Table A-7 DababaseMetaData Object (Continued)

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

Table A-7 DababaseMetaData Object (Continued)

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
boolean nullsAreSortedAtStart ()	1.0	Yes	
boolean nullsAreSortedHigh ()	1.0	Yes	
boolean nullsAreSortedLow ()	1.0	Yes	
boolean othersDeletesAreVisible (int)	2.0 Core	Yes	
boolean othersInsertsAreVisible (int)	2.0 Core	Yes	
boolean othersUpdatesAreVisible (int)	2.0 Core	Yes	
boolean ownDeletesAreVisible (int)	2.0 Core	Yes	
boolean ownInsertsAreVisible (int)	2.0 Core	Yes	
boolean ownUpdatesAreVisible (int)	2.0 Core	Yes	
boolean storesLowerCaseIdentifiers ()	1.0	Yes	
boolean storesLowerCaseQuoted Identifiers ()	1.0	Yes	
boolean storesMixedCaseIdentifiers ()	1.0	Yes	
boolean storesMixedCaseQuoted Identifiers ()	1.0	Yes	
boolean storesUpperCaseIdentifiers ()	1.0	Yes	
boolean storesUpperCaseQuoted Identifiers ()	1.0	Yes	
boolean supportsAlterTableWith AddColumn ()	1.0	Yes	
boolean supportsAlterTableWith DropColumn ()	1.0	Yes	
boolean supportsANSI92EntryLevelSQL ()	1.0	Yes	
boolean supportsANSI92FullSQL ()	1.0	Yes	

Table A-7 DababaseMetaData Object (Continued)

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
boolean supportsANSI92Intermediate SQL ()	1.0	Yes	
boolean supportsBatchUpdates ()	2.0 Core	Yes	
boolean supportsCatalogsInData Manipulation ()	1.0	Yes	
boolean supportsCatalogsInIndex Definitions ()	1.0	Yes	
boolean supportsCatalogsInPrivilege Definitions ()	1.0	Yes	
boolean supportsCatalogsInProcedure Calls ()	1.0	Yes	
boolean supportsCatalogsInTable Definitions ()	1.0	Yes	
boolean supportsColumnAliasing ()	1.0	Yes	
boolean supportsConvert ()	1.0	Yes	
boolean supportsConvert (int, int)	1.0	Yes	
boolean supportsCoreSQLGrammar ()	1.0	Yes	
boolean supportsCorrelatedSubqueries ()	1.0	Yes	
boolean supportsDataDefinitionAndData ManipulationTransactions ()	1.0	Yes	
boolean supportsDataManipulation TransactionsOnly ()	1.0	Yes	
boolean supportsDifferentTableCorrelation Names ()	1.0	Yes	
boolean supportsExpressionsIn OrderBy ()	1.0	Yes	

Table A-7 DababaseMetaData Object (Continued)

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
boolean supportsExtendedSQLGrammar ()	1.0	Yes	
boolean supportsFullOuterJoins ()	1.0	Yes	
boolean supportsGetGeneratedKeys ()	3.0	Yes	
boolean supportsGroupBy ()	1.0	Yes	
boolean supportsGroupByBeyondSelect ()	1.0	Yes	
boolean supportsGroupByUnrelated ()	1.0	Yes	
boolean supportsIntegrityEnhancement Facility ()	1.0	Yes	
boolean supportsLikeEscapeClause ()	1.0	Yes	
boolean supportsLimitedOuterJoins ()	1.0	Yes	
boolean supportsMinimumSQLGrammar ()	1.0	Yes	
boolean supportsMixedCaseIdentifiers ()	1.0	Yes	
boolean supportsMixedCaseQuoted Identifiers ()	1.0	Yes	
boolean supportsMultipleOpenResults ()	3.0	Yes	
boolean supportsMultipleResultSets ()	1.0	Yes	
boolean supportsMultipleTransactions ()	1.0	Yes	
boolean supportsNamedParameters ()	3.0	Yes	
boolean supportsNonNullableColumns ()	1.0	Yes	
boolean supportsOpenCursorsAcross Commit ()	1.0	Yes	
boolean supportsOpenCursorsAcross Rollback ()	1.0	Yes	

Table A-7 DababaseMetaData Object (Continued)

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
boolean supportsOpenStatementsAcross Commit ()	1.0	Yes	
boolean supportsOpenStatementsAcross Rollback ()	1.0	Yes	
boolean supportsOrderByUnrelated ()	1.0	Yes	
boolean supportsOuterJoins ()	1.0	Yes	
boolean supportsPositionedDelete ()	1.0	Yes	
boolean supportsPositionedUpdate ()	1.0	Yes	
boolean supportsResultSetConcurrency (int, int)	2.0 Core	Yes	
boolean supportsResultSetHoldability (int)	3.0	Yes	
boolean supportsResultSetType (int)	2.0 Core	Yes	
boolean supportsSavePoints ()	3.0	Yes	
boolean supportsSchemasInData Manipulation ()	1.0	Yes	
boolean supportsSchemasInIndex Definitions ()	1.0	Yes	
boolean supportsSchemasIn PrivilegeDefinitions ()	1.0	Yes	
boolean supportsSchemasInProcedure Calls ()	1.0	Yes	
boolean supportsSchemasInTable Definitions ()	1.0	Yes	
boolean supportsSelectForUpdate ()	1.0	Yes	
boolean supportsStoredProcedures ()	1.0	Yes	
boolean supportsSubqueriesIn Comparisons ()	1.0	Yes	

Table A-7 DababaseMetaData Object (Continued)

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
boolean supportsSubqueriesInExists ()	1.0	Yes	
boolean supportsSubqueriesInIns ()	1.0	Yes	
boolean supportsSubqueriesIn Quantifieds ()	1.0	Yes	
boolean supportsTableCorrelationNames ()	1.0	Yes	
boolean supportsTransactionIsolationLevel (int)	1.0	Yes	
boolean supportsTransactions ()	1.0	Yes	
boolean supportsUnion ()	1.0	Yes	
boolean supportsUnionAll ()	1.0	Yes	
boolean updatesAreDetected (int)	2.0 Core	Yes	
boolean usesLocalFilePerTable ()	1.0	Yes	
boolean usesLocalFiles ()	1.0	Yes	

Driver Object

Table A-8 Driver Object

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

ParameterMetaData Object

Table A-9 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	
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.

Table A-9 ParameterMetaData Object

ParameterMetaData Object Methods	Version Introduced	Supported	Comments
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.
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 jdbcCompliant ()	1.0	Yes	

PreparedStatement Object

Table A-10 PreparedStatement Object

PreparedStatement Object Methods	Version Introduced	Supported	Comments
void addBatch ()	2.0 Core	Yes	
void clearParameters ()	1.0	Yes	
boolean execute ()	1.0	Yes	
ResultSet executeQuery ()	1.0	Yes	
int executeUpdate ()	1.0	Yes	
ResultSetMetaData getMetaData ()	2.0 Core	Yes	
ParameterMetaData getParameterMetaData ()	3.0	Yes	
void setArray (int, Array)	2.0 Core	No	Throws "unsupported method" exception.
void setAsciiStream (int, InputStream, int)	1.0	Yes	
void setBigDecimal (int, BigDecimal)	1.0	Yes	
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.

Table A-10 PreparedStatement Object (Continued)

PreparedStatement Object Methods	Version Introduced	Supported	Comments
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 setBoolean (int, boolean)	1.0	Yes	
void setByte (int, byte)	1.0	Yes	
void setBytes (int, byte [])	1.0	Yes	When used with Blobs, the DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.
void setCharacterStream (int, Reader, int)	2.0 Core	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	
void setDate (int, Date, Calendar)	2.0 Core	Yes	
void setDouble (int, double)	1.0	Yes	

Table A-10 PreparedStatement Object (Continued)

PreparedStatement Object Methods	Version Introduced	Supported	Comments
void setFloat (int, float)	1.0	Yes	
void setInt (int, int)	1.0	Yes	
void setLong (int, long)	1.0	Yes	
void setNull (int, int)	1.0	Yes	
void setNull (int, int, String)	2.0 Core	Yes	
void setObject (int, Object)	1.0	Yes	
void setObject (int, Object, int)	1.0	Yes	
void setObject (int, Object, int, int)	1.0	Yes	

Table A-10 PreparedStatement Object (Continued)

PreparedStatement 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 throws an "unsupported method" exception. The Oracle, SQL Server, and Sybase drivers support setting a timeout value, in seconds, for a statement. If the execution of the statement exceeds the timeout value, the statement is timed out by the database server, and the driver throws an exception indicating that the statement was timed out.
void setRef (int, Ref)	2.0 Core	No	Throws "unsupported method" exception.
void setShort (int, short)	1.0	Yes	
void setString (int, String)	1.0	Yes	

Table A-10 PreparedStatement Object (Continued)

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

Ref Object

Table A-11 Ref Object

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

ResultSet Object

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
boolean absolute (int)	2.0 Core	Yes	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
void afterLast ()	2.0 Core	Yes	
void beforeFirst ()	2.0 Core	Yes	
void cancelRowUpdates ()	2.0 Core	Yes	
void clearWarnings ()	1.0	Yes	
void close ()	1.0	Yes	
void deleteRow ()	2.0 Core	Yes	
int findColumn (String)	1.0	Yes	
boolean first ()	2.0 Core	Yes	
Array getArray (int)	2.0 Core	No	Throws "unsupported method" exception.
Array getArray (String)	2.0 Core	No	Throws "unsupported method" exception.
InputStream getAsciiStream (int)	1.0	Yes	
InputStream getAsciiStream (String)	1.0	Yes	
BigDecimal getBigDecimal (int)	2.0 Core	Yes	
BigDecimal getBigDecimal (int, int)	1.0	Yes	
BigDecimal getBigDecimal (String)	2.0 Core	Yes	
BigDecimal getBigDecimal (String, int)	1.0	Yes	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
void afterLast ()	2.0 Core	Yes	
void beforeFirst ()	2.0 Core	Yes	
void cancelRowUpdates ()	2.0 Core	Yes	
void clearWarnings ()	1.0	Yes	
void close ()	1.0	Yes	
void deleteRow ()	2.0 Core	Yes	
int findColumn (String)	1.0	Yes	
boolean first ()	2.0 Core	Yes	
Array getArray (int)	2.0 Core	No	Throws "unsupported method" exception.
Array getArray (String)	2.0 Core	No	Throws "unsupported method" exception.
InputStream getAsciiStream (int)	1.0	Yes	
InputStream getAsciiStream (String)	1.0	Yes	
BigDecimal getBigDecimal (int)	2.0 Core	Yes	
BigDecimal getBigDecimal (int, int)	1.0	Yes	
BigDecimal getBigDecimal (String)	2.0 Core	Yes	
BigDecimal getBigDecimal (String, int)	1.0	Yes	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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.

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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.
Blob getBlob (String)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries. The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARBINARY data type.
boolean getBoolean (int)	1.0	Yes	
boolean getBoolean (String)	1.0	Yes	
byte getByte (int)	1.0	Yes	
byte getByte (String)	1.0	Yes	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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	
Reader getCharacterStream (String)	2.0 Core	Yes	
Clob getClob (int)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with data types that map to the JDBC LONGVARCHAR data type.

Table A-12 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	
String getCursorName ()	1.0	No	Throws "unsupported method" exception.
Date getDate (int)	1.0	Yes	
Date getDate (int, Calendar)	2.0 Core	Yes	
Date getDate (String)	1.0	Yes	
Date getDate (String, Calendar)	2.0 Core	Yes	
double getDouble (int)	1.0	Yes	
double getDouble (String)	1.0	Yes	
int getFetchDirection ()	2.0 Core	Yes	
int getFetchSize ()	2.0 Core	Yes	
float getFloat (int)	1.0	Yes	
float getFloat (String)	1.0	Yes	
int getInt (int)	1.0	Yes	
int getInt (String)	1.0	Yes	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
long getLong (int)	1.0	Yes	
long getLong (String)	1.0	Yes	
ResultSetMetaData getMetaData ()	1.0	Yes	
Object getObject (int)	1.0	Yes	Returns a Long object when called on DB2 Bigint columns.
Object getObject (int, Map)	2.0 Core	Yes	
Object getObject (String)	1.0	Yes	
Object getObject (String, Map)	2.0 Core	Yes	Map ignored.
Ref getRef (int)	2.0 Core	No	Throws "unsupported method" exception.
Ref getRef (String)	2.0 Core	No	Throws "unsupported method" exception.
int getRow ()	2.0 Core	Yes	
short getShort (int)	1.0	Yes	
short getShort (String)	1.0	Yes	
Statement getStatement ()	2.0 Core	Yes	
String getString (int)	1.0	Yes	
String getString (String)	1.0	Yes	
Time getTime (int)	1.0	Yes	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
Time getTime (int, Calendar)	2.0 Core	Yes	
Time getTime (String)	1.0	Yes	
Time getTime (String, Calendar)	2.0 Core	Yes	
Timestamp getTimestamp (int)	1.0	Yes	
Timestamp getTimestamp (int, Calendar)	2.0 Core	Yes	
Timestamp getTimestamp (String)	1.0	Yes	
Timestamp getTimestamp (String, Calendar)	2.0 Core	Yes	
int getType ()	2.0 Core	Yes	
InputStream getUnicodeStream (int)	1.0	No	Throws "unsupported method" exception. This method was deprecated in JDBC 2.0.
InputStream getUnicodeStream (String)	1.0	No	Throws "unsupported method" exception. This method was deprecated in JDBC 2.0.
URL getURL (int)	3.0	No	Throws "unsupported method" exception.
URL getURL (String)	3.0	No	Throws "unsupported method" exception.
SQLWarning getWarnings ()	1.0	Yes	
void insertRow ()	2.0 Core	Yes	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
boolean isAfterLast ()	2.0 Core	Yes	
boolean isBeforeFirst ()	2.0 Core	Yes	
boolean isFirst ()	2.0 Core	Yes	
boolean isLast ()	2.0 Core	Yes	
boolean last ()	2.0 Core	Yes	
void moveToCurrentRow ()	2.0 Core	Yes	
void moveToInsertRow ()	2.0 Core	Yes	
boolean next ()	1.0	Yes	
boolean previous ()	2.0 Core	Yes	
void refreshRow ()	2.0 Core	Yes	
boolean relative (int)	2.0 Core	Yes	
boolean rowDeleted ()	2.0 Core	Yes	
boolean rowInserted ()	2.0 Core	Yes	
boolean rowUpdated ()	2.0 Core	Yes	
void setFetchDirection (int)	2.0 Core	Yes	
void setFetchSize (int)	2.0 Core	Yes	
void updateArray (int, Array)	3.0	No	Throws "unsupported method" exception.

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
void updateArray (String, Array)	3.0	No	Throws "unsupported method" exception.
void updateAsciiStream (int, InputStream, int)	2.0 Core	Yes	
void updateAsciiStream (String, InputStream, int)	2.0 Core	Yes	
void updateBigDecimal (int, BigDecimal)	2.0 Core	Yes	
void updateBigDecimal (String, BigDecimal)	2.0 Core	Yes	
void updateBinaryStream (int, InputStream, int)	2.0 Core	Yes	
void updateBinaryStream (String, InputStream, int)	2.0 Core	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.

Table A-12 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 updateBoolean (int, boolean)	2.0 Core	Yes	
void updateBoolean (String, boolean)	2.0 Core	Yes	
void updateByte (int, byte)	2.0 Core	Yes	
void updateByte (String, byte)	2.0 Core	Yes	
void updateBytes (int, byte [])	2.0 Core	Yes	
void updateBytes (String, byte [])	2.0 Core	Yes	
void updateCharacterStream (int, Reader, int)	2.0 Core	Yes	
void updateCharacterStream (String, Reader, int)	2.0 Core	Yes	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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 (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 updateDate (int, Date)	2.0 Core	Yes	
void updateDate (String, Date)	2.0 Core	Yes	
void updateDouble (int, double)	2.0 Core	Yes	
void updateDouble (String, double)	2.0 Core	Yes	
void updateFloat (int, float)	2.0 Core	Yes	
void updateFloat (String, float)	2.0 Core	Yes	
void updateInt (int, int)	2.0 Core	Yes	
void updateInt (String, int)	2.0 Core	Yes	
void updateLong (int, long)	2.0 Core	Yes	
void updateLong (String, long)	2.0 Core	Yes	
void updateNull (int)	2.0 Core	Yes	

Table A-12 ResultSet Object

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

ResultSetMetaData Object

Table A-13 ResultSetMetaData Object

ResultSetMetaData Object Methods	Version Introduced	Supported	Comments
String getCatalogName (int)	1.0	Yes	
String getColumnClassName (int)	2.0 Core	Yes	
int getColumnCount ()	1.0	Yes	
int getColumnDisplaySize (int)	1.0	Yes	
String getColumnLabel (int)	1.0	Yes	
String getColumnName (int)	1.0	Yes	
int getColumnType (int)	1.0	Yes	
String getColumnTypeName (int)	1.0	Yes	
int getPrecision (int)	1.0	Yes	
int getScale (int)	1.0	Yes	
String getSchemaName (int)	1.0	Yes	

Table A-13 ResultSetMetaData Object (Continued)

ResultSetMetaData Object (Continued)	Version Introduced	Supported	Comments
String getTableName (int)	1.0	Yes	For versions 3.4 and higher: • By default, `getTableName` returns an empty string for the Oracle, Informix, and SQL Server Type 4 drivers. • To return a table name for the Oracle, 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 Administration Console Online Help.
boolean isAutoIncrement (int)	1.0	Yes	
boolean isCaseSensitive (int)	1.0	Yes	
boolean isCurrency (int)	1.0	Yes	
boolean isDefinitelyWritable (int)	1.0	Yes	
int isNullable (int)	1.0	Yes	
boolean isReadOnly (int)	1.0	Yes	
boolean isSearchable (int)	1.0	Yes	
boolean isSigned (int)	1.0	Yes	
boolean isWritable (int)	1.0	Yes	

SavePoint Object

Table A-14 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.

Statement Object

Table A-15 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-15 Statement Object (Continued)

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 Oracle, SQL Server, and Sybase drivers cancel the execution of the statement. If the statement is cancelled by the database server, the driver throws an exception indicating that it was cancelled.
void clearBatch ()	2.0 Core	Yes	
void clearWarnings ()	1.0	Yes	
void close ()	1.0	Yes	
boolean execute (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.
boolean execute (String, int)	3.0	Yes	
boolean execute (String, int [])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws "unsupported method" exception.

Table A-15 Statement Object (Continued)

Statement Object Methods	Version Introduced	Supported	Comments
boolean execute (String, String [])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws "unsupported method" exception.
int [] executeBatch ()	2.0 Core	Yes	
ResultSet executeQuery (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.
int executeUpdate (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.
int executeUpdate (String, int)	3.0	Yes	
int executeUpdate (String, int [])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws "unsupported method" exception.
int executeUpdate (String, String [])	3.0	Yes	Supported for the Oracle and SQL Server drivers. For all other drivers, throws "unsupported method" exception.
Connection getConnection ()	2.0 Core	Yes	
int getFetchDirection ()	2.0 Core	Yes	
int getFetchSize ()	2.0 Core	Yes	

Table A-15 Statement Object (Continued)

Statement Object Methods	Version Introduced	Supported	Comments
ResultSet getGeneratedKeys ()	3.0	Yes	The DB2, SQL Server, and Sybase drivers return the last value inserted into an identity column. If an identity column does not exist in the table, the drivers return an empty result set. The Informix driver returns the last value inserted into a Serial or Serial8 column. If a Serial or Serial8 column does not exist in the table, the driver returns an empty result set. The Oracle driver returns the ROWID of the last row inserted.
int getMaxFieldSize ()	1.0	Yes	
int getMaxRows ()	1.0	Yes	
boolean getMoreResults ()	1.0	Yes	
boolean getMoreResults (int)	3.0	Yes	
int getQueryTimeout ()	1.0	Yes	The DB2 driver returns the timeout value, in seconds, set for the statement with DB2 v8.x and higher for Linux/UNIX/Windows and DB2 v8.1 for z/OS. The DB2 driver returns 0 with other DB2 versions. The Informix driver returns 0. The Oracle, SQL Server, and Sybase drivers return the timeout value, in seconds, set for the statement.
ResultSet getResultSet ()	1.0	Yes	
int getResultSetConcurrency ()	2.0 Core	Yes	

Table A-15 Statement Object (Continued)

Statement Object Methods	Version Introduced	Supported	Comments
int getResultSetHoldability ()	3.0	Yes	
int getResultSetType ()	2.0 Core	Yes	
int getUpdateCount ()	1.0	Yes	
SQLWarning getWarnings ()	1.0	Yes	
void setCursorName (String)	1.0	No	Throws "unsupported method" exception.
void setEscapeProcessing (boolean)	1.0	Yes	Ignored.
void setFetchDirection (int)	2.0 Core	Yes	
void setFetchSize (int)	2.0 Core	Yes	
void setMaxFieldSize (int)	1.0	Yes	
void setMaxRows (int)	1.0	Yes	

Table A-15 Statement Object (Continued)

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 throws an "unsupported method" exception. The Oracle, 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.

Struct Object

Table A-16 Struct Object

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

XAConnection Object

Table A-17 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.

XADataSource Object

Table A-18 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.

XAResource Object

Table A-19 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.

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.

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 = bigint *	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bigint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
* Supported only for DB2 for Linux/UNIX/Windows, DB2 for iSeries, and DB2 v9.1 for z/OS	
TYPE_NAME = binary*	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION = 255
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = BINARY(X'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ')	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = binary	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
* Supported only for DB2 v9.1 for z/OS	
TYPE_NAME = blob *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 2004 (BLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = BLOB	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-1 getTypeInfo for DB2 (Continued)

* Supported only for DB2 v8.1 and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.	
TYPE_NAME = char	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION =
FIXED_PREC_SCALE = false	254 (DB2 for Linux/UNIX/Windows),
LITERAL_PREFIX = ’	255 (DB2 for z/OS),
LITERAL_SUFFIX = ’	32765 (DB2 for iSeries)
LOCAL_TYPE_NAME = char	SEARCHABLE = 3
MAXIMUM_SCALE = NULL	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = char for bit data	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION =
FIXED_PREC_SCALE = false	254 (DB2 for Linux/UNIX/Windows),
LITERAL_PREFIX = X’	255 (DB2 for z/OS),
LITERAL_SUFFIX = ’	32765 (DB2 for iSeries)
LOCAL_TYPE_NAME = char for bit data	SEARCHABLE = 3
MAXIMUM_SCALE = NULL	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = clob	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = clob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-1 getTypeInfo for DB2 (Continued)

TYPE_NAME = date	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {d‘	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ‘}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = dbclob *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS =	NUM_PREC_RADIX = NULL
(length) (DB2 for Linux/UNIX/Windows and	PRECISION = 2147483647
DB2 for z/OS	SEARCHABLE = 1
(length) CCSID 13488 (DB2 V5R2, V5R3 for iSeries)	SQL_DATA_TYPE = NULL
DATA_TYPE = 2005 (DBCLOB)	SQL_DATETIME_SUB = NULL
FIXED_PREC_SCALE = false	UNSIGNED_ATTRIBUTE = NULL
LITERAL_PREFIX = ’	
LITERAL_SUFFIX = ’	
LOCAL_TYPE_NAME = dbclob	
MAXIMUM_SCALE = NULL	
* Supported only for DB2 v8.x and higher for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and higher for iSeries.	
TYPE_NAME = decfloat*	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *precision*	NUM_PREC_RADIX = 10
DATA_TYPE = -3 (DECIMAL)	PRECISION = 34
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = NULL	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	
* Supported only for DB2 v9.1 for z/OS	

Table B-1 getTypeInfo for DB2 (Continued)

TYPE_NAME = decimal	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = (*precision,scale*)	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 31
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = decimal	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 31	
TYPE_NAME = double	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 8 (DOUBLE)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = double	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	
TYPE_NAME = float	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 6 (FLOAT)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

Table B-1 getTypeInfo for DB2 (Continued)

TYPE_NAME = graphic	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = length	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION =
FIXED_PREC_SCALE = false	127 (DB2 for Linux/UNIX/Windows),
LITERAL_PREFIX = G'	127 (DB2 for z/OS),
LITERAL_SUFFIX = '	16352 (DB2 for iSeries)
LOCAL_TYPE_NAME = char	SEARCHABLE = 3
MAXIMUM_SCALE = NULL	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = integer	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = integer	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
TYPE_NAME = long varchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION =
FIXED_PREC_SCALE = false	32700 (DB2 for Linux/UNIX/Windows,*
LITERAL_PREFIX = ’	32704 (DB2 for z/OS) *
LITERAL_SUFFIX = ’	32700 (DB2 for iSeries) *
LOCAL_TYPE_NAME = long varchar	SEARCHABLE = 1
MAXIMUM_SCALE = NULL	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

* 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.

Table B-1 getTypeInfo for DB2 (Continued)

TYPE_NAME = long varchar for bit data	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION =
FIXED_PREC_SCALE = false	32700 (DB2 for Linux/UNIX/Windows),
LITERAL_PREFIX = X’	32698 (DB2 for z/OS),
LITERAL_SUFFIX = ’	32739 (DB2 for iSeries)
LOCAL_TYPE_NAME = long varchar for bit data	SEARCHABLE = 1
MAXIMUM_SCALE = NULL	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = long vargraphic	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 16352
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = G'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = longvarchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = numeric	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = (*precision, scale*)	NUM_PREC_RADIX =10
DATA_TYPE = 2 (NUMERIC)	PRECISION = 31
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = numeric	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 31	

Table B-1 getTypeInfo for DB2 (Continued)

TYPE_NAME = real	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float(4)	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	
TYPE_NAME = rowid *	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = not null generated always	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (Binary)	PRECISION = 40
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = rowid	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	
* Supported only for DB2 for z/OS and DB2 V5R2 and higher for iSeries.	
TYPE_NAME = smallint	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 5 (SMALLINT)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

Table B-1 getTypeInfo for DB2 (Continued)

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

TYPE_NAME = varbinary*	

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

* Supported only for DB2 v9.1 for z/OS	

Table B-1 getTypeInfo for DB2 (Continued)

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

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

Table B-1 getTypeInfo for DB2 (Continued)

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

TYPE_NAME = xml*	

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

* Supported only for DB2 V9.1 for Linux/UNIX/Windows.and DB2 v9.1 for z/OS.	

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_NAME = blob	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2004 (BLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = blob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = boolean	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -7 (BIT)	PRECISION = 1
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = boolean	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	
TYPE_NAME = byte	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = byte	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-2 getTypeInfo for Informix

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

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 getTypeInfo for Informix

TYPE_NAME = datetime hour to second	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 92 (TIME)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {t’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime hour to second	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	
TYPE_NAME = datetime year to day	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {d’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime year to day	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = datetime year to fraction(5)	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 5
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 25
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime hour to fraction(5)	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 5	

Table B-2 getTypeInfo for Informix

TYPE_NAME = datetime year to second	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime hour to second	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	
TYPE_NAME = decimal	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *precision, scale*	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 32
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = decimal	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 32	
TYPE_NAME = float	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 6 (FLOAT)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

Table B-2 getTypeInfo for Informix

TYPE_NAME = int8	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = int8	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
TYPE_NAME = integer	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = integer	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
TYPE_NAME = lvarchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL (Informix 9.2, 9.3), *max length* (Informix 9.4, 10)	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 2048 (Informix 9.2, 9.3), 32739 (Informix 9.4, 10)
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = lvarchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-2 getTypeInfo for Informix

TYPE_NAME = money	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *precision,scale*	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 32
FIXED_PREC_SCALE = true	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = money	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 32	
TYPE_NAME = nchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 32766
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = nvarchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *max length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 254
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nvarchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-2 getTypeInfo for Informix

TYPE_NAME = serial	
AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = start	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = serial	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
TYPE_NAME = serial8	
AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = serial8	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
TYPE_NAME = smallfloat	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallfloat	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

Table B-2 getTypeInfo for Informix

TYPE_NAME = smallint	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 5 (SMALLINT)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
TYPE_NAME = text	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = text	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = varchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *max length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 254
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Oracle Driver (Deprecated)

Table B-3 provides getTypeInfo results for all Oracle databases supported by the Oracle driver. See Chapter 6, "The Oracle Driver (Deprecated)."

Table B-3 getTypeInfo for Oracle

TYPE_NAME = bfile	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2004 (BLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bfile	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = binary_float *	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = binary_float	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	
* Supported only for Oracle 10g.	
TYPE_NAME = binary_double *	
AUTO_INCREMENT = false	MINIMUM_SCALE =NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 8 (DOUBLE)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = binary_double	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	
* Supported only for Oracle 10g.	

Table B-3 getTypeInfo for Oracle (Continued)

TYPE_NAME = blob	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2004 (BLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = blob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = char	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 2000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = char	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = clob	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = clob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-3 getTypeInfo for Oracle (Continued)

TYPE_NAME = date	

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 = date
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

TYPE_NAME = long	

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
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 = long raw	

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = -4 (LONGVARBINARY)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = '
LITERAL_SUFFIX = '
LOCAL_TYPE_NAME = long raw
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 getTypeInfo for Oracle (Continued)

TYPE_NAME = nchar	

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
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 = 2000
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = nclob	

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = NULL
DATA_TYPE = 2005 (CLOB)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = N'
LITERAL_SUFFIX = '
LOCAL_TYPE_NAME = nclob
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 = number (p, s)	

AUTO_INCREMENT = false
CASE_SENSITIVE = false
CREATE_PARAMS = *precision, scale*
DATA_TYPE = 3 (DECIMAL)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = number
MAXIMUM_SCALE = 127
	MINIMUM_SCALE = -84
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 38
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

Table B-3 getTypeInfo for Oracle (Continued)

TYPE_NAME = number	

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 = number
MAXIMUM_SCALE = 127
	MINIMUM_SCALE = -84
	NULLABLE = 1
	NUM_PREC_RADIX = 10
	PRECISION = 38
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

TYPE_NAME = nvarchar2	

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = *max length*
DATA_TYPE = 12 (VARCHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = N’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = nvarchar2
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

TYPE_NAME = raw	

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = *max length*
DATA_TYPE = -3 (VARBINARY)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = raw
MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 2000
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B-3 getTypeInfo for Oracle (Continued)

TYPE_NAME = timestamp *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *fractional_seconds_precision*	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 9	
* Supported only for Oracle 9i and higher.	
TYPE_NAME = timestamp with local time zone *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *fractional_seconds_precision*	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX =’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp with local time zone	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 9	
* Supported only for Oracle 9i and higher.	
TYPE_NAME = timestamp with time zone *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *fractional_seconds_precision*	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp with time zone	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 9	
*Supported only for Oracle 9i and higher.	

Table B-3 getTypeInfo for Oracle (Continued)

TYPE_NAME = urowid	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *max length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = urowid	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
* Supported only for Oracle 9i and higher.	
TYPE_NAME = varchar2	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *max length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar2	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = xmltype *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = xmltype(’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’)	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = xmltype	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
* Supported only for Oracle 9i (R2) and higher.	

SQL Server Driver

Table B-4 provides getTypeInfo results for for all Microsoft SQL Server databases supported by the SQL Server driver. See Chapter 5, “The MS SQL Server Driver.”

Table B-4 getTypeInfo for SQL Server

TYPE_NAME = bigint *	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bigint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
* Supported only for Microsoft SQL Server 2000 and higher.	
TYPE_NAME = bigint identity *	
AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bigint identity	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
* Supported only for Microsoft SQL Server 2000 and higher.	
TYPE_NAME = binary	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = binary	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-4 getTypeInfo for SQL Server (Continued)

TYPE_NAME = bit	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -7 (BIT)	PRECISION = 1
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bit	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	
TYPE_NAME = char	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = char	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = datetime	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 3
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 23
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 3	

Table B-4 getTypeInfo for SQL Server (Continued)

TYPE_NAME = decimal	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *precision,scale*	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION =
FIXED_PREC_SCALE = false	28 (SQL Server 7) *,
LITERAL_PREFIX = NULL	38 (SQL Server 2000 and SQL Server 2005) *
LITERAL_SUFFIX = NULL	SEARCHABLE = 2
LOCAL_TYPE_NAME = decimal	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE =	SQL_DATETIME_SUB = NULL
28 (SQL Server 7), *	UNSIGNED_ATTRIBUTE = false
38 (SQL Server 2000 and SQL Server 2005) *	
* Configurable server option for Microsoft SQL Server 2000 and higher.	
TYPE_NAME = decimal() identity	
AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = *precision*	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION =
FIXED_PREC_SCALE = false	28 (SQL Server 7),
LITERAL_PREFIX = NULL	38 (SQL Server 2000 and SQL Server 2005)
LITERAL_SUFFIX = NULL	SEARCHABLE = 2
LOCAL_TYPE_NAME = decimal() identity	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE = 0	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false
TYPE_NAME = float	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 2
DATA_TYPE = 6 (FLOAT)	PRECISION = 53
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

Table B-4 getTypeInfo for SQL Server (Continued)

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

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-4 getTypeInfo for SQL Server (Continued)

TYPE_NAME = money	
AUTO_INCREMENT = false	MINIMUM_SCALE = 4
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 19
FIXED_PREC_SCALE = true	SEARCHABLE = 2
LITERAL_PREFIX = $	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = money	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 4	
TYPE_NAME = nchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = ntext	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 1073741823
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = ntext	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-4 getTypeInfo for SQL Server (Continued)

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),*
38 (SQL Server 2000 and SQL Server 2005) *
	MINIMUM_SCALE = 0
	NULLABLE = 1
	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

* Configurable server option for Microsoft SQL Server 2000 and higher.	
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-4 getTypeInfo for SQL Server (Continued)

TYPE_NAME = nvarchar(max) *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 1073741823
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nvarchar(max)	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
* Supported only for Microsoft SQL Server 2005.	
TYPE_NAME = real	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 2
DATA_TYPE = 7 (REAL)	PRECISION = 24
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = real	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	
TYPE_NAME = smalldatetime	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 16
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smalldatetime	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

Table B-4 getTypeInfo for SQL Server (Continued)

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

Table B-4 getTypeInfo for SQL Server (Continued)

TYPE_NAME = sql_variant *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 12 (VARCHAR)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = sql_variant	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	
* Supported only for Microsoft SQL Server 2000 and higher.	
TYPE_NAME = sysname	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 128
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = sysname	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = text	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = text	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-4 getTypeInfo for SQL Server (Continued)

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

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

Table B-4 getTypeInfo for SQL Server (Continued)

TYPE_NAME = uniqueidentifier	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 1(CHAR)	PRECISION = 36
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = uniqueidentifier	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = varbinary	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *max length*	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varbinary	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = varbinary(max) *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varbinary(max)	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
* Supported only for Microsoft SQL Server 2005.	

Table B-4 getTypeInfo for SQL Server (Continued)

TYPE_NAME = varchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = varchar(max) *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar(max)	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
* Supported only for Microsoft SQL Server 2005.	
TYPE_NAME = xml *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 1073741823
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = xml	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
* Supported only for Microsoft SQL Server 2005.	

Sybase Driver

Table B-5 provides getTypeInfo results for all Sybase databases supported by the Sybase driver (see Chapter 7, “The Sybase Driver”).

Table B-5 getTypeInfo for Sybase

TYPE_NAME = bigint *	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bigint	UNSIGNED_ATTRIBU
MAXIMUM_SCALE = 0	
* Supported only for Sybase 15.	
TYPE_NAME = binary	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION =
FIXED_PREC_SCALE = false	255 (Sybase 11.*x*, 12.0)*
LITERAL_PREFIX = 0x	2048 (Sybase 12.5 and higher) *
LITERAL_SUFFIX = NULL	SEARCHABLE = 2
LOCAL_TYPE_NAME = binary	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE = NULL	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
* For Sybase 12.5.1 and higher, precision is determined by the server page size.	
TYPE_NAME = bit	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -7 (BIT)	PRECISION = 1
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bit	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

Table B-5 getTypeInfo for Sybase

TYPE_NAME = char	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION =
FIXED_PREC_SCALE = false	255 (Sybase 11.*x*, 12.0)*
LITERAL_PREFIX = ’	2048 (Sybase 12.5 and higher) *
LITERAL_SUFFIX = ’	SEARCHABLE = 3
LOCAL_TYPE_NAME = char	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE = NULL	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL
* For Sybase 12.5.1 and higher, precision is determined by the server page size.	
TYPE_NAME = date *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
* Supported only for Sybase 12.5.1 and higher.	
TYPE_NAME = datetime	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 3
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 23
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 3	

Table B-5 getTypeInfo for Sybase

TYPE_NAME = decimal	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *precision,scale*	NUM_PREC_RADIX = NULL
DATA_TYPE = 3 (DECIMAL)	PRECISION = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = decimal	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 38	
TYPE_NAME = float	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 6 (FLOAT)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	
TYPE_NAME = image	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = image	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-5 getTypeInfo for Sybase

TYPE_NAME = int

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = int	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

TYPE_NAME = money

AUTO_INCREMENT = false	MINIMUM_SCALE = 4
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 3 (DECIMAL)	PRECISION = 19
FIXED_PREC_SCALE = true	SEARCHABLE = 2
LITERAL_PREFIX = $	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = money	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 4	

TYPE_NAME = nchar

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION =
FIXED_PREC_SCALE = false	255 (Sybase 11.*x*, 12.0*)
LITERAL_PREFIX = ’	2048 (Sybase 12.5 and higher) *
LITERAL_SUFFIX = ’	SEARCHABLE = 3
LOCAL_TYPE_NAME = nchar	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE = NULL	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

* For Sybase 12.5.1 and higher, precision is determined by the server page size.

Table B-5 getTypeInfo for Sybase

TYPE_NAME = numeric

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *precision,scale*	NUM_PREC_RADIX = NULL
DATA_TYPE = 2 (NUMERIC)	PRECISION = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = numeric	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 38	

TYPE_NAME = nvarchar

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *max length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION =
FIXED_PREC_SCALE = false	255 (Sybase 11.x, 12.0)*
LITERAL_PREFIX = ’	2048 (Sybase 12.5 and higher) *
LITERAL_SUFFIX = ’	SEARCHABLE = 3
LOCAL_TYPE_NAME = nvarchar	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE = NULL	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

* For Sybase 12.5.1 and higher, precision is determined by the server page size.

TYPE_NAME = real

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = real	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

Table B-5 getTypeInfo for Sybase

TYPE_NAME = smalldatetime	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 3
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 16
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smalldatetime	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 3	
TYPE_NAME = smallint	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 5 (SMALLINT)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
TYPE_NAME = smallmoney	
AUTO_INCREMENT = false	MINIMUM_SCALE = 4
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 3 (DECIMAL)	PRECISION = 10
FIXED_PREC_SCALE = true	SEARCHABLE = 2
LITERAL_PREFIX = $	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallmoney	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 4	

Table B-5 getTypeInfo for Sybase

TYPE_NAME = sysname

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *max length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 30
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = sysname	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = text

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = text	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = time *

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 3
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 92 (TIME)	PRECISION = 12
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = time	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 3	

* Supported only for Sybase 12.5.1 and higher.

Table B-5 getTypeInfo for Sybase

TYPE_NAME = timestamp

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX =0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = tinyint

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -6 (TINTYINT)	PRECISION = 3
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = tinyint	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

TYPE_NAME = unsigned bigint *

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 3 (DECIMAL)	PRECISION = 20
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = unsigned bigint	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

* Supported only for Sybase 15.

Table B-5 getTypeInfo for Sybase

TYPE_NAME = unsigned int *

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -5 (BIGINT)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = unsigned int	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

* Supported only for Sybase 15.

TYPE_NAME = unsigned smallint *

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 4 (INTEGER)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = unsigned smallint	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

* Supported only for Sybase 15.

TYPE_NAME = unichar *

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION =2048
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = unichar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

* Supported only for Sybase 12.5 and higher.

Table B-5 getTypeInfo for Sybase

TYPE_NAME = unitext

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = unitext	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

* Supported only for Sybase 15.

TYPE_NAME = univarchar *

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *max length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 2048
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = univarchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

* Supported only for Sybase 12.5 and higher.

TYPE_NAME = varbinary

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *max length*	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION =
FIXED_PREC_SCALE = false	255 (Sybase 11.*x*, 12.0)*
LITERAL_PREFIX = 0x	2048 (Sybase 12.5 and higher) *
LITERAL_SUFFIX = NULL	SEARCHABLE = 2
LOCAL_TYPE_NAME = varbinary	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE = NULL	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

* For Sybase 12.5.1 and higher, precision is determined by the server page size.

Table B-5 getTypeInfo for Sybase

TYPE_NAME = varchar

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = *max length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION =
FIXED_PREC_SCALE = false	255 (Sybase 11.*x*, 12.0)*
LITERAL_PREFIX = ’	2048 (Sybase 12.5 and higher) *
LITERAL_SUFFIX = ’	SEARCHABLE = 3
LOCAL_TYPE_NAME = varchar	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE = NULL	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

* For Sybase 12.5.1 and higher, precision is determined by the server page size.

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.

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...]`

Example:

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

Scalar Functions

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

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

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

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 or CEIL	HOUR	DLURLSERVER
	HEX	COS	JULIAN_DAY	DLVALUE
	INSERT	COSH	MICROSECOND	EVENT_MON_STATE
	LCASE or LOWER	COT	MIDNIGHT_SECONDS	GENERATE_UNIQUE
	LCASE (SYSFUN schema)	DECIMAL	MINUTE	NODENUMBER
	LEFT	DEGREES	MONTH	NULLIF
	LENGTH	DIGITS	MONTHNAME	PARTITION
	LOCATE	DOUBLE	NOW	RAISE_ERROR
	LONG_VARCHAR	EXP	QUARTER	TABLE_NAME
	LONG_VARGRAPHIC	FLOAT	SECOND	TABLE_SCHEMA
	LTRIM	FLOOR	TIME	TRANSLATE
	LTRIM (SYSFUN schema)	INTEGER	TIMESTAMP	TYPE_ID
	POSSTR	LN	TIMESTAMP_ISO	TYPE_NAME
	REPEAT	LOG	TIMESTAMPDIFF	TYPE_SCHEMA
	REPLACE	LOG10	WEEK	VALUE
	RIGHT	MOD	YEAR	
	RTRIM	POWER		
	RTRIM (SYSFUN schema)	RADIANS		
		RAND		
		REAL		

Table C-2 Scalar Functions Supported (Continued)

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

Table C-2 Scalar Functions Supported (Continued)

Data Store	String Functions	Numeric Functions	Timedate Functions	System Functions
Oracle	ASCII	ABS	CURDATE	IFNULL
	BIT_LENGTH	ACOS	DAYNAME	USER
	CHAR	ASIN	DAYOFMONTH	
	CONCAT	ATAN	DAYOFWEEK	
	INSERT	ATAN2	DAYOFYEAR	
	LCASE	CEILING	HOUR	
	LEFT	COS	MINUTE	
	LENGTH	COT	MONTH	
	LOCATE	EXP	MONTHNAME	
	LOCATE2	FLOOR	NOW	
	LTRIM	LOG	QUARTER	
	OCTET_LENGTH	LOG10	SECOND	
	REPEAT	MOD	WEEK	
	REPLACE	PI	YEAR	
	RIGHT	POWER		
	RTRIM	ROUND		
	SOUNDEX	SIGN		
	SPACE	SIN		
	SUBSTRING	SQRT		
	UCASE	TAN		
		TRUNCATE		

Table C-2 Scalar Functions Supported (Continued)

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 Scalar Functions Supported (Continued)

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		

Outer Join Escape Sequences

JDBC supports the SQL92 left, right, and full outer join syntax. The escape sequence for outer joins is:

```
{oj outer-join}
```

where *outer-join* is:

```
table-reference {LEFT | RIGHT | FULL} OUTER JOIN

{table-reference | outer-join} ON search-condition
```

where:

table-reference is a database table name.

search-condition is the join condition you want to use for the tables.

Example:

```
SELECT Customers.CustID, Customers.Name, Orders.OrderID, Orders.Status

   FROM {oj Customers LEFT OUTER JOIN

      Orders ON Customers.CustID=Orders.CustID}

   WHERE Orders.Status='OPEN'
```

Table 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
Oracle	Left outer joins Right outer joins Nested outer joins

Table C-3 Outer Join Escape Sequences Supported (Continued)

Data Store	Outer Join Escape Sequences
SQL Server	Left outer joins Right outer joins Full outer joins Nested outer joins
Sybase	Left outer joins Right outer joins Nested outer joins

LIKE Escape Character Sequence for Wildcards

You can specify the character to be used to escape wildcard characters (% and _, for example) in LIKE clauses. The escape sequence for escape characters is:

```
{escape 'escape-character'}
```

where `escape-character` is the character used to escape the wildcard character.

For example. the following SQL statement specifies that an asterisk (*) be used as the escape character in the LIKE clause for the wildcard character %:

```
SELECT col1 FROM table1 WHERE col1 LIKE '*%%' {escape '*'}
```

Procedure Call Escape Sequences

A procedure is an executable object stored in the data store. Generally, it is one or more SQL statements that have been precompiled. The escape sequence for calling a procedure is:

```
   {[?=]call procedure-name[([parameter][,parameter]...)]}
```

where:

procedure-name specifies the name of a stored procedure.

parameter specifies a stored procedure parameter.

Note: For DB2 for Linux/UNIX/Windows, a catalog name cannot be used when calling a stored procedure. Also, for DB2 v8.1 and v8.2 for Linux/UNIX/Windows, literal parameter values are supported for stored procedures. Other supported DB2 versions do not support literal parameter values for stored procedures.

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 3.0-compliant, including support for the JDBC 2.0 Optional Package. WebLogic JDBC Spy also supports logging for supported JDBC 4.0 features.
- 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.

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. Follow these instructions for modifying your data source configuration:

1. Before you start the server, add `WL_HOME/server/lib/wlspy.jar` to your CLASSPATH, where *`WL_HOME`* is the directory in which you installed the WebLogic Server software.
2. 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:bea:DB2://db2host:50000;spyAttributes=(log=(file)d:\spy.log;timest
   amp=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:bea:db2://bangpcdb2:50000;spyAttributes=(log=(file)db2-spy.ou
   t;load=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>
```

3. Stop and restart WebLogic Server.

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`.
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 | no | 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.

Table D-1 WebLogic JDBC Spy URL Attributes (Continued)

Key-Value Pair	Description
logTName={yes | no}	Specifies whether WebLogic JDBC Spy logs the name of the current thread. The default is no.
timestamp={yes | no}	Specifies whether a timestamp should be included on each line of the WebLogic JDBC Spy log.

WebLogic JDBC Spy Log Example

The superscript Numbers are note indicators. See the notes following the example for the referenced text.

```
All rights reserved.[1]

registerDriver:driver[className=weblogic.jdbcspy.SpyDriver,
context=null,weblogic.jdbcspy.SpyDriver@1ec49f][2]

*Driver.connect(jdbc:spy:{jdbc:bea: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:bea: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
spy>> OK (Statement[1])[7]

spy>> Statement[1].executeQuery(String sql)
spy>> sql = select empno,ename,job from emp where empno=7369
spy>> OK (ResultSet[1])[8]
spy>> ResultSet[1].getMetaData()
spy>> OK (ResultSetMetaData[1])[9]
spy>> ResultSetMetaData[1].getColumnCount()
spy>> OK (3)[10]
spy>> ResultSetMetaData[1].getColumnLabel(int column)
spy>> column = 1
spy>> OK (EMPNO)[11]
spy>> ResultSetMetaData[1].getColumnLabel(int column)
spy>> column = 2
spy>> OK (ENAME)[12]
spy>> ResultSetMetaData[1].getColumnLabel(int column)
spy>> column = 3
spy>> OK (JOB)[13]
spy>> ResultSet[1].next()
spy>> OK (true)[14]
spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 1
spy>> OK (7369)[15]
spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 2
spy>> OK (SMITH)[16]
spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 3
spy>> OK (CLERK)[17]
spy>> ResultSet[1].next()
spy>> OK (false)[18]
spy>> ResultSet[1].close()
spy>> OK[19]
spy>> Connection[1].close()
spy>> OK[20]
```

NOTES:

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.

7, 8 The statement “select empno,ename,job from emp where empno=7369” is created.

9, 10, 11, 12, 13 Some metadata is requested.

14, 15, 16, 17 The first row is fetched and its data retrieved.

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

19 After fetching all data, the result set is closed.

20 The application finishes and disconnects.