BEA WebLogic Server®

Type 4 JDBC Drivers

Version 9.1
Document Revised: February 17, 2006

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

1. Introduction and Roadmap

2. Using WebLogic Type 4 JDBC Drivers

3. The DB2 Driver

4. The Informix Driver

5. The MS SQL Server Driver

6. The Oracle Driver

7. The Sybase Driver

A. JDBC Support

B. GetTypeInfo

C. SQL Escape Sequences for JDBC

D. Tracking JDBC Calls with WebLogic JDBC Spy

CHAPTER 1

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 J2EE 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 BEA 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," 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 BEA WebLogic Type 4 JDBC drivers.
- Appendix C, "SQL Escape Sequences for JDBC," describes the scalar functions supported for the BEA 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 BEA 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, BEA Systems 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 J2EE 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 J2EE features, and highlights BEA-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 Platform.

JDBC Examples in the WebLogic Server Distribution

WebLogic Server 9.0 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 9.0 Start menu.

New and Changed JDBC Features in This Release

The following sections describe new features and changes for WebLogic Server Type 4 JDBC Drivers in WebLogic Server 9.1:

For a comprehensive listing of the new WebLogic JDBC features introduced in release 9.1, see "What's New in WebLogic Server 9.1" in *Release Notes*.

Note: If you are not familiar with the new features provided in version 9.0 of WebLogic Server, see the What's New in WebLogic Server 9.0 section of the *WebLogic Server Release Notes*.

Changes for All Drivers

- All drivers have been certified with J2SE 5.0.
- `ResultSet` metadata support.
- New `JavaDoubleToString` connection option allows you to choose the optimal conversion algorithm when converting double or float values to string values.
- New `ResultSetMetaDataOptions` connection option allows the driver to return table name information in `ResultSet` metadata for `Select` statements.

For more information, refer to the documentation for each driver.

Changes for DB2

- Support for new DB2 v8.2 for Linux/UNIX/Windows features, including:
  - Maximum length of SQL statements increased to 2MB
  - Ability to create tables with a Unicode character set in a database with a non-Unicode default character set
  - Nested savepoints
- Support for new DB2 v8.1 for z/OS features, including:
  - Cancel and query timeout support
  - Extended identifier lengths

- Support for new DB2 v5R3 for iSeries features, including support for new UTF-8 and UTF-16 encodings.
- Driver now supports parameter metadata for all supported DB2 platforms and versions.
- New EnableCancelTimeout connection option that provides the ability to time out cancel requests.
- Performance improvements.

For detailed information on these features, see "The DB2 Driver" on page 3-1.

Changes for Informix

- Parameter metadata support.
- New `DBDate` connection option sets the Informix DBDate server environment variable.

For detailed information on these features, see "The Informix Driver" on page 4-1.

Changes for Oracle

- New Windows-specific Type 2 OS authentication method supports NTLM authentication.
- Parameter metadata support.
- New `EnableCancelTimeout` connection option provides the ability to time out cancel requests.
- New `SendFloatParametersAsString` connection option allows the driver to send float and double parameters to the server as a string or floating point number.
- New `WireProtocolMode` connection option allows the driver to optimize network traffic to the Oracle server for result sets. Oracle 10g database users may need to set `WireProtocolMode=2` to prevent performance degradation when using parameterized queries (prepared and callable statements) involving big tables. See "Oracle Connection Properties" on page 6-2.
- Improved character set support.
- Improved LOB performance.

For detailed information on these features, see "The Oracle Driver" on page 6-1.

Changes for SQL Server

- New Type 2 OS authentication method supports both Kerberos and NTLM authentication.
- New `PacketSize` connection option allows you to fine-tune the size of the packet the driver uses to communicate with the database server.
- New `EnableCancelTimeout` connection option that provides the ability to time out cancel requests.
- Parameter metadata support.
- Performance improvements.

For detailed information on these features, see "The MS SQL Server Driver" on page 5-1.

Changes for Sybase

- Support for Sybase 12.5.3.
- New `EnableCancelTimeout` connection option that provides the ability to time out cancel requests.
- New `UseAlternateProductInfo` connection option allows you to specify whether the driver will perform additional processing to return more accurate information for the `DatabaseMetaData.getDatabaseProductName()` and `DatabaseMetaData.getDatabaseProductVersion()` methods.
- New `ErrorBehavior` connection option allows you to fine-tune how errors that are returned from Stored Procedures are handled.

For detailed information on these features, see "The Sybase Driver" on page 7-1.

CHAPTER 2

Using WebLogic Type 4 JDBC Drivers

BEA 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 BEA 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 J2EE 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 BEA WebLogic Type 4 JDBC drivers:

JDBC Specification Compliance

BEA WebLogic Type 4 JDBC drivers are compliant with the JDBC 3.0 specification. They also support several JDBC 2.0 Standard Extension Features. 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.

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`
  - For Sybase: `wlsybase.jar`

License Requirements

BEA WebLogic Type 4 JDBC drivers are licensed for use with WebLogic Server and with WebLogic Server clients only. At least one of the following files must be in your classpath for the license check to succeed:

- `weblogic.jar`
- `wlclient.jar`

Supported Databases

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

Table 2-1

Driver	Supported Databases
DB2	• DB2 Universal Database (UDB) v7.*x*, v8.1, and v8.2 on Linux, UNIX, and Windows via DRDA • DB2 UDB v7.x and v8.1 for z/OS via DRDA • DB2 UDB V5R1, V5R2, and V5R3 for iSeries
Informix	Informix 9.4 and higher
Oracle	• Oracle 9i (R1 and R2) • Oracle 10*g* (R1)
SQL Server	• Microsoft SQL Server 7.0, SQL Server 2000 (including SP1, SP2, and SP3a) • Microsoft SQL Server 2000 Desktop Engine (MSDE 2000) • SQL Server 2000 Enterprise Edition (64-bit)
Sybase	• Sybase Adaptive Server 11.5 and 11.9 • Sybase Adaptive Server Enterprise 12.0, 12.5, 12.5.1, 12.5.2, and 12.5.3 • Sybase Adaptive Server Enterprise 15

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 BEA 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 4-2.
- For the MS SQL Server driver, see "SQL Server Connection Properties" on page 5-3.
- For the Oracle driver, see "Oracle Connection Properties" on page 6-2.
- For the Sybase driver, see "Sybase Connection Properties" on page 7-2.

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, if 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 failover, see the appropriate driver chapter:

- "DB2 Connection Properties" on page 3-3

- "Informix Connection Properties" on page 4-2
- "SQL Server Connection Properties" on page 5-3
- "Oracle Connection Properties" on page 6-2
- "Sybase JTA Support" on page 7-14
- . See "JDBC Data Source: Configuration: Transaction" in *Administration Console Online Help*.

Required Permissions for the Java Security Manager

Using the BEA 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 Java Virtual Machine 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 BEA 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 BEA 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 2-11.

Granting Access to Temporary Files

Access to the temporary directory specified by the Java Virtual Machine 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 2-11.

Granting Access to Oracle tnsnames.ora Files

If you are using an Oracle tnsnames.ora file to connect with the BEA 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 2-11.

See "Performance Considerations" on page 6-9 for more information about using `tnsnames.ora` files to connect to Oracle databases.

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 3-21.
- For Microsoft SQL Server, see "Installing Stored Procedures for JTA" on page 5-13.
- For Oracle, see "Batch Inserts and Updates" on page 6-26.
- For Sybase, see "Sybase JTA Support" on page 7-14

Unicode Support

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

CHAPTER 3

The DB2 Driver

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

Database Version Support

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

- DB2 Universal Database (UDB) v7.*x*, v8.1, and v8.2 on Linux, UNIX, and Windows via DRDA
- DB2 UDB v7.x and v8.1 for z/OS via DRDA
- DB2 UDB V5R1, V5R2, and V5R3 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 class for the BEA WebLogic Type 4 JDBC DB2 driver is:

```
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

To connect to a DB2 database, use the appropriate URL format:

- DB2 on Windows NT, Windows 2000, Windows 2003, Windows XP, UNIX, Linux, and Linux/s3901:

```
jdbc:bea:db2://db2_server_name:port;DatabaseName=your_database
```

- DB2 on OS/390, z/OS, iSeries, and AS/4001:

```
jdbc:bea:db2://db2_server_name:port;Location=db2_location;CollectionId=
your_collectionname
```

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. 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 3-1 DB2 Connection Properties

Property	Description
AddToCreateTable OPTIONAL	A string that is automatically added to all Create Table 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

Property	Description
BatchPerformanceWorkaround OPTIONAL	{true | false}. For DB2 UDB 8.1, the native DB2 batch mechanism is used. 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 "Performance Workaround for Batch Inserts and Updates" on page 3-22 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}. When set to `true`, synonyms are included in the result sets returned from the following DatabaseMetaData methods: `getColumns`, `getProcedureColumns`, and `getIndexInfo`. When set to `false`, synonyms are omitted from result sets. The default is `true`.
CatalogSchema OPTIONAL	The DB2 schema to use for catalog functions. The value must be the name of a valid DB2 schema. The default is SYSCAT for DB2 UDB, SYSIBM for DB2 OS/390, and QSYS2 for DB2 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 "Using a Non-Default Schema for Catalog Methods" on page 3-18 for the required views of catalog tables.

Table 3-1 DB2 Connection Properties

Property	Description
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 Java Virtual Machine, 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. 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 Java Virtual Machine, for example, CodePageOverride=CP950.
CollectionId OPTIONAL	The collection (group of packages) to which the package is bound. This property is ignored for DB2 UDB. The default is NULLID.
ConnectionRetryCount OPTIONAL	The number of times the driver retries connection attemptsuntil 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. The default is 0.
ConnectionRetryDelay OPTIONAL	The number of seconds the driver will wait between connection retry attempts when ConnectionRetryCount is set to a positive integer. The default is 3.

Table 3-1 DB2 Connection Properties

Property	Description
CreateDefaultPackage OPTIONAL	{true | false}. Determines whether the default package should be created. For DB2 OS/390 and DB2 iSeries, the package is created in the collection specified by the CollectionId property. This would be used if the package does not yet exist. For more information about creating DB2 packages, see "Creating a DB2 Package" on page 3-13. The default is false.
DatabaseName	The name of the database to which you want to connect (used with UDB).
DynamicSections OPTIONAL	Specifies the number of statements that the DB2 driver package can prepare for a single user. The default is 200.
Grantee OPTIONAL	Specifies the name of the schema to which you want to grant EXECUTE privileges for DB2 packages. This property is ignored if the GrantExecute property is set to false. See "Creating a DB2 Package" on page 3-13 for more information about creating DB2 packages. The default is PUBLIC.
GrantExecute OPTIONAL	{true | false}. Determines whether EXECUTE privileges for DB2 packages are granted to a schema other than the one used to create them. If set to true, EXECUTE privileges are granted to the schema specified by the Grantee property. If set to false, EXECUTE privileges are not granted to another schema. See "Creating a DB2 Package" on page 3-13 for more information about creating DB2 packages. The default is true.

Table 3-1 DB2 Connection Properties

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)
LocationName	The name of the DB2 location that you want to access (used with OS/390 and iSeries).
LoginTimeout OPTIONAL	The maximum time in seconds that attempts to create a database connection will wait. A value of `0` specifies that the timeout is the default system timeout if there is one; otherwise it specifies that there is no timeout.
PackageOwner OPTIONAL	Specifies the owner of DB2 packages. See "Creating a DB2 Package" on page 3-13 for more information about creating DB2 packages. The default is NULL.
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.

Table 3-1 DB2 Connection Properties

Property	Description
PortNumber OPTIONAL	The TCP port on which the database server listens for connections. The default is 50000.
ReplacePackage OPTIONAL	{true | false}. Specifies whether the current bind process should replace an existing DB2 package. On DB2 UDB, this property must be used in conjunction with CreateDefaultPackage. For more information about creating DB2 packages, see "Creating a DB2 Package" on page 3-13. The default is false.
SecurityMechanism OPTIONAL	{ClearText | EncryptedPassword | EncryptedUIDPassword}. Determines the security method the driver uses to authenticate the user to the DB2 server when establishing a connection. If the specified authentication method is not supported by the DB2 server, the connection fails and the driver generates an exception. If set to ClearText, the driver sends the password in clear text to the DB2 server for authentication. If set to EncryptedPassword, the driver sends an encrypted password to the DB2 server for authentication. If set to EncryptedUIDPassword, the driver sends an encrypted user ID and password to the DB2 server for authentication. The default is ClearText. Requires JDK 1.4 or higher.

Table 3-1 DB2 Connection Properties

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 stream data that is less than 32K bytes can be inserted into a Long Varchar for Bit Data column, which has a maximum length of 32K bytes, or a Blob column. 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 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 stream passed into the driver is resettable. Sending binary stream data that is less than 32K bytes in length initially as a Blob significantly improves performance if the Insert or Update column is a Blob column. 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. The default is false.
ServerName	The name or IP address of the database server.
StripNewlines OPTIONAL	{true | false}. Specifies whether new-line characters in a SQL statement are sent to the DB2 server. When StripNewlines=true, the DB2 driver removes all new-line characters from SQL statements. The default is true.

Table 3-1 DB2 Connection Properties

Property	Description
UseCurrentSchema OPTIONAL	{true | false}. Specifies whether results are restricted to the tables in the current schema if a DatabaseMetaData.getTables call is called without specifying a schema or if the schema is specified as the wildcard character %. Restricting results to the tables 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 method are restricted to tables in the current schema. If set to false, results of the getTables method are not restricted. The default is false.
User	The case-sensitive user name used to connect to your DB2 database.
WithHoldCursors OPTIONAL	{true | false}. Determines whether the cursor stays open on commit—either DB2 closes all open cursors (Delete cursors) after a commit or leaves them open (Preserve cursors). If set to true, the cursor behavior is Preserve. If set to false, the cursor behavior is Delete. Rolling back a transaction closes all cursors regardless of how this property is specified. The default is true.

Performance Considerations

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

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 v7.1 or v7.2 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 v8.1 and v8.2 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 "Using a Non-Default Schema for Catalog Methods" on page 3-18 for views for catalog tables that must exist in the specified schema.

InsensitiveResultSetBufferSize

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

MaxPooledStatements

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

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 3-24 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. In this case, the `StripNewlines` property should be set to false.

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").newInstance();
conn =
myDriver.connect("jdbc:bea:db2://10.1.4.1:446;LocationName=S10B757B",
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.

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

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 3-14.

- Automatically create a package by setting specific connection properties in the connection URL. See "Creating a DB2 Package Using Connection Properties" on page 3-14.

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 at
`http://e-docs.bea.com/wls/docs91/admin_ref/utils.html#db2package`.

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

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 3-21).
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 3-21).
4. Supported only for DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries.

See "GetTypeInfo" on page B-1 for more information about data types.

Using a Non-Default Schema for Catalog Methods

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

Table 3-4 Catalog Tables for DB2

Database	Catalog Tables
DB2 for Linux/UNIX/Windows	SYSCAT.TABLES SYSCAT.COLUMNS SYSCAT.PROCEDURES SYSCAT.PROCPARAMS SYSCAT.COLAUTH SYSCAT.TABAUTH SYSCAT.KEYCOLUSE SYSCAT.INDEXES SYSCAT.INDEXCOLUSE SYSCAT.REFERENCES SYSCAT.SYSSCHEMATA SYSCAT.TYPEMAPPINGS SYSCAT.DBAUTH

Table 3-4 Catalog Tables for DB2 (Continued)

Database	Catalog Tables
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 3-5. JDBC isolation levels are mapped to the appropriate DB2 transaction isolation levels as shown. The default isolation level is `Read Committed`.

Table 3-5 Supported Isolation Levels

JDBC Isolation Level	DB2 Isolation Level
None	No Commit[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, DB2 v8.1 or v8.2 for Linux/UNIX/Windows is required.

Large Object (LOB) Support

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

- DB2 v8.1 and v8.2 for Linux/UNIX/Windows
- DB2 for z/OS
- DB2 V5R2 and V5R3 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 v8.1 and v8.2 for Linux/UNIX/Windows

- DB2 for z/OS
- DB2 V5R2 and V5R3 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 v8.1 and v8.2 for Linux/UNIX/Windows
- DB2 7.*x* for z/OS
- DB2 V5R2 and V5R3 for iSeries

Performance Workaround for Batch Inserts and Updates

For DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 for iSeries, 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.

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 v8.1 and v8.2 for Linux/UNIX/Windows
- DB2 for z/OS
- DB2 V5R2 and V5R3 for iSeries

For all other supported DB2 databases, 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 v8.1 and v8.2 for Linux/UNIX/Windows
- DB2 for z/OS
- DB2 V5R2 and V5R3 for iSeries

For all other supported DB2 databases, 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 = ?"
```

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

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.

How you return those 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 inform the driver to return the values of auto-generated keys:
  - `Statement.execute (String sql, int autoGeneratedKeys)`
  - `Statement.executeUpdate (String sql, int autoGeneratedKeys)`
- 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)`

The application fetches the values of generated keys from the driver using the `Statement.getGeneratedKeys` method.

CHAPTER 4

The Informix Driver

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

Informix Database Version Support

The BEA WebLogic Type 4 JDBC Informix driver (the "Informix driver") supports Informix Dynamic Server 9.4 and higher.

Informix Driver Classes

The driver classes for the BEA 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://dbserver1:1543;informixServer=dbserver1;databaseNam
e=dbname
```

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. 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 Java Virtual Machine, for example, CodePageOverride=CP950.
ConnectionRetryCount OPTIONAL	The number of times the driver retries connections to a database servers (primary and alternate) until a successful connection is established. Valid values are 0 and any positive integer. If set to 0, the driver does not retry a connection to the list of database servers if a connection is not established on the driver's first pass through the list. The default is 0.
ConnectionRetryDelay OPTIONAL	The number of seconds the driver will wait between connection retry attempts when ConnectionRetryCount is set to a positive integer. The default is 3.
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.
InformixServer	The name of the Informix database server to which you want to connect.

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)
LoginTimeout OPTIONAL	The maximum time in seconds that attempts to create a database connection will wait. A value of 0 specifies that the timeout is the default system timeout if there is one; otherwise it specifies that there is no timeout.
Password	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 get your password.
PortNumber	The TCP port on which the database server listens for connections. The default varies depending on operating system.
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 InformixServer.
User	The case-insensitive default user name used to connect to your Informix 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.

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:

InsensitiveResultSetBufferSize

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

MaxPooledStatements

To improve performance, the driver's own internal prepared statement pooling should be enabled when the driver does not run from within an application server or from within another application that does not provide its own prepared statement pooling. When the driver's internal prepared statement pooling is enabled, the driver caches a certain number of prepared statements created

by an application. For example, if the `MaxPooledStatements` property is set to 20, the driver caches the last 20 prepared statements created by the application. If the value set for this property is greater than the number of prepared statements used by the application, all prepared statements are cached.

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 4-9 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
clob	CLOB
char	CHAR
date	DATE
datetime hour to second	TIME
datetime year to day	DATE
datetime year to fraction(5)	TIMESTAMP
datetime year to second	TIMESTAMP

Table 4-2 Informix Data Types (Continued)

Informix Data Type	JDBC Data Type
decimal	DECIMAL
float	FLOAT
int8	BIGINT
integer	INTEGER
lvarchar	VARCHAR
money	DECIMAL
nchar	CHAR
nvarchar	VARCHAR
serial	INTEGER
serial8	BIGINT
smallfloat	REAL
smallint	SMALLINT
text	LONGVARCHAR
varchar	VARCHAR

See "GetTypeInfo" on page B-1 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 does not support returning parameter metadata for stored procedure arguments.

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 = ?"
```

ResultSet MetaData Support

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

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

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

```
SELECT id, name FROM Employee

SELECT E.id, E.name FROM Employee E
```

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

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

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

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

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

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

Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

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

Blob and Clob Searches

When searching a Clob value for a string pattern using the Clob.position method, the search pattern must be less than or equal to a maximum value of 4096 bytes. Similarly, when searching a Blob value for a byte pattern using the Blob.position method, the search pattern must be less than or equal to a maximum value of 4096 bytes.

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.

- 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 inform the driver to return the values of auto-generated keys:

  ```
  Statement.execute (String sql, int autoGeneratedKeys)
  ```

  ```
  Statement.executeUpdate (String sql, int autoGeneratedKeys).
  ```

- When using a `Insert` statement that contains parameters, the Informix 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)
  ```

The application fetches the values of generated keys from the driver using the `Statement.getGeneratedKeys()` method.

CHAPTER 5

The MS SQL Server Driver

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

Note: The BEA 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. BEA recommends that you use the new BEA WebLogic Type 4 JDBC MS SQL Server driver in place of the WebLogic jDriver for Microsoft SQL Server.

SQL Server Database Version Support

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

- Microsoft SQL Server 7.0
- Microsoft SQL Server 2000 (including SP1, SP2, and SP3a)
- Microsoft SQL Server 2000 Desktop Engine (MSDE 2000)
- Microsoft SQL Server 2000 Enterprise Edition (64-bit)

To use JDBC distributed transactions through JTA, you must install stored procedures for SQL Server. See "Installing Stored Procedures for JTA" on page 5-13 for details.

Driver Class

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

XA: `weblogic.jdbcx.sqlserver.SQLServerDataSource`

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

URL

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

```
jdbc:bea:sqlserver://dbserver:port
```

Connecting to Named Instances

Microsoft SQL Server supports 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). 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 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.
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 Java Virtual Machine, for example, CodePageOverride=CP950. 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 retry a connection to the list of database servers if a connection is not established on the driver's first pass through the list. The default is 0.

Table 5-1 SQL Server Connection Properties

Property	Description
ConnectionRetryDelay OPTIONAL	The number of seconds the driver waits before retrying connection attempts when ConnectionRetryCount is set to a positive integer. The default is 3.
DatabaseName OPTIONAL	The name of the database to which you want to connect.
HostProcess OPTIONAL	The process ID of the application connecting to Microsoft SQL Server. The value of this property appears in the hostprocess column of the master.dbo.sysprocesses table and may be useful for database administration purposes. The default is 0.
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).

Table 5-1 SQL Server Connection Properties

Property	Description
NetAddress OPTIONAL	The Media Access Control (MAC) address of the network interface card of the application connecting to Microsoft SQL Server. The value of this property appears in the net_address column of the master.dbo.sysprocesses table and may be useful for database administration purposes. The default is 000000000000.
Password	A case-insensitive password used to connect to your Microsoft SQL Server database.
PortNumber OPTIONAL	The TCP port of the primary database server that is listening for connections to the Microsoft SQL Server database. The default is 1433.
ProgramName OPTIONAL	The name of the application connecting to Microsoft SQL Server. The value of this property appears in the program_name column of the master.dbo.sysprocesses table and may be useful for database administration purposes. The default is an empty string.

Table 5-1 SQL Server Connection Properties

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. • Direct—When the driver uses the Direct method, 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 all the 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. • Cursor—When the driver uses the Cursor method, a server-side cursor is requested. The rows are retrieved from the server in blocks when returning forward-only result sets. The JDBC Statement method setFetchSize can be used to control the number of rows that are retrieved for each request. Performance tests show that the value of setFetchSize significantly impacts performance when the Cursor method is used. There is no simple rule for determining the setFetchSize value that you should use. BEA 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. The default is Direct.

Table 5-1 SQL Server Connection Properties

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, string parameters are sent to Microsoft SQL Server in Unicode. If set to false, string parameters are sent in the default encoding, which can improve performance because the server does not need to convert Unicode characters to the default encoding. You should, however, use default encoding only if the parameter string data you specify is the same as the default encoding of the database. The default is true. 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.
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 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.
User	The case-insensitive user name used to connect to your Microsoft SQL Server database.
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. The default is false. See "Server-Side Updatable Cursors" on page 5-13 for more information about using server-side updatable cursors.

Table 5-1 SQL Server Connection Properties

Property	Description
WSID OPTIONAL	The workstation ID, which typically is the network name of the computer on which the application resides. If specified, this value is stored in the hostname column of the master.dbo.sysprocesses table and can be returned by sp_who and the Transact-SQL HOST_NAME function. The value can be useful for database administration purposes. The default is an empty string.
XATransactionGroup OPTIONAL	The transaction group ID that identifies any transactions initiated by the connection. This ID can be used for distributed transaction cleanup purposes.

Performance Considerations

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

InsensitiveResultSetBufferSize

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

MaxPooledStatements

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

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 5-17 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 retrieve large amounts of data.
- Your application executes a SQL statement before processing or closing a previous large result set and does this multiple times.
- Large result sets 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.

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 5-13 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 Data Types Supported by SQL Server 7 and SQL Server 2000

SQL Server Data Type	JDBC Data Type
binary	BINARY
bit	BIT
char	CHAR
datetime	TIMESTAMP
decimal	DECIMAL
decimal() identity	DECIMAL
float	FLOAT
image	LONGVARBINARY
int	INTEGER
int identity	INTEGER
money	DECIMAL
nchar	CHAR
ntext	LONGVARCHAR
numeric	NUMERIC
numeric() identity	NUMERIC
nvarchar	VARCHAR
real	REAL
smalldatetime	TIMESTAMP
smallint	SMALLINT

Table 5-2 Data Types Supported by SQL Server 7 and SQL Server 2000

SQL Server Data Type	JDBC Data Type
smallint identity	SMALLINT
smallmoney	DECIMAL
sysname	VARCHAR
text	LONGVARCHAR
timestamp	BINARY
tinyint	TINYINT
tinyint identity	TINYINT
uniqueidentifier	CHAR
varbinary	VARBINARY
varchar	VARCHAR

Table 5-3 lists additional data types supported by SQL Server 2000 only.

Table 5-3 Addition Data Types Supported by SQL Server 2000

SQL Server Data Type	JDBC Data Type
bigint	BIGINT
bigint identity	BIGINT
sql_variant	VARCHAR

See "GetTypeInfo" on page B-1 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 SQL Server driver.

Isolation Levels

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

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, any changes you make to the result set 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.

Note: If you install a patch on your Microsoft SQL Server DBMS installation, you must reinstall the stored procedures for JTA (as described below). Also, some WebLogic Server service packs include driver updates and may require that you reinstall the stored procedures for JTA (as described below).

To install stored procedures for JTA:

1. Copy the `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\weblogic81`.

   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. The system administrator should back up the master database before running `instjdbc.sql`. At a command prompt, use the following syntax to run `instjdbc.sql`:

   `ISQL -Usa -P`*sa_password* `-S`*server_name* `-i`*location*`\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.

Large Object (LOB) Support

Although Microsoft SQL Server does not define a Blob or Clob data type, the SQL Server driver allows you to retrieve and update long data, specifically LONGVARBINARY and LONGVARCHAR data, using JDBC methods designed for Blobs and Clobs. When using these

methods to update long data as Blobs or Clobs, the updates are made to the local copy of the data contained in the Blob or Clob object.

Retrieving and updating long data using JDBC methods designed for Blobs and Clobs provides some of the same 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 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 = ?"
```

ResultSet MetaData Support

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

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

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

```
SELECT id, name FROM Employee

SELECT E.id, E.name FROM Employee E

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

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

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

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

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

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

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

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

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

Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

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

Auto-Generated Keys Support

The SQL Server driver supports retrieving the values of auto-generated keys. An auto-generated key returned by the SQL Server driver is the value of an identity column.

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 inform the driver to return the values of auto-generated keys:

  - `Statement.execute (String sql, int autoGeneratedKeys)`
  - `Statement.executeUpdate (String sql, int autoGeneratedKeys)`

- 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 the values of auto-generated keys:
  - `Connection.prepareStatement (String sql, int autoGeneratedKeys)`

The application fetches the values of generated keys from the driver using the `Statement.getGeneratedKeys` method.

CHAPTER 6

The Oracle Driver

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

Oracle Database Version Support

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

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

Oracle Driver Classes

The driver classes for the BEA 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

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

```
jdbc:bea:oracle://dbserver:port
```

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
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 set to false, the JDBC 3.0-compliant batch mechanism is used. If an application can accept not receiving update count information, setting this property to true can significantly improve performance. The default is false. See "Batch Inserts and Updates" on page 6-26 for details.
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 default DatabaseMetaData results. If set to 1, result sets contain Remarks information returned from the DatabaseMetaData methods: getTables and getColumns. If set to 2, result sets contain synonyms returned from the DatabaseMetaData methods: getColumns, 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.
ConnectionRetryCount OPTIONAL	The number of times the driver retries connections to database serveruntil a successful connection is established. Valid values are 0 and any positive integer. If set to 0, the driver does not retry a connection attempt. The default is 0.

Table 6-1 Oracle Connection String Properties

Property	Description
ConnectionRetryDelay OPTIONAL	The number of seconds the driver waits before retrying connections to a database serverwhen ConnectionRetryCount is set to a positive integer. The default is 3.
FetchTSWTZasTimestamp OPTIONAL	{true | false}. If set to true, allows column values with the TIMESTAMP WITH TIME ZONE data type (Oracle9i or higher) to be retrieved 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 6-20 for more information.
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).
LoginTimeout OPTIONAL	The maximum time in seconds that attempts to create a database connection will wait. A value of 0 specifies that the timeout is the default system timeout if there is one; otherwise it specifies that there is no timeout.

Table 6-1 Oracle Connection String Properties

Property	Description
Password	A case-insensitive password used to connect to your Oracle database. A password is required only if security is enabled on your database. If so, contact your system administrator to obtain your password.
PortNumber OPTIONAL	The TCP port of the Oracle listener running on the Oracle database server. The default is 1521, which is the Oracle default port number when installing the Oracle database software. If using a tnsnames.ora file to provide connection information, do not specify this property. See "Performance Considerations" on page 6-9 for information about specifying a port number for the Oracle listener using a tnsnames.ora file.
ServerName OPTIONAL	Specifies either the IP address 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. See "Performance Considerations" on page 6-9 for information about specifying a server name using a tnsnames.ora file.

Table 6-1 Oracle Connection String Properties

Property	Description
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 "Performance Considerations" on page 6-9 for information about specifying the server type using a `tnsnames.ora` file.
ServiceName OPTIONAL	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` 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 6-9 for information about specifying the database service name using a tnsnames.ora file.

Table 6-1 Oracle Connection String Properties

Property	Description
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 6-9 for information about specifying an Oracle SID using a tnsnames.ora file.
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 generates an exception. See "Performance Considerations" on page 6-9 for information about using tnsnames.ora files to connect.

Table 6-1 Oracle Connection String Properties

Property	Description
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 generates an exception. See "Performance Considerations" on page 6-9 for information about using tnsnames.ora files to connect.
User	The case-insensitive default user name used to connect to your Oracle 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. Operating System authentication is not currently supported by the Oracle driver.

Table 6-1 Oracle Connection String Properties

Property	Description
WireProtocolMode	This driver can improve performance if you typically return data that is repeated in consecutive rows. For example, the data in column1/row1 is the same as the data in column1/row2, and so on. If this is the case, set `WireProtocolMode=2` and the driver optimizes network traffic to the Oracle server for result sets containing multiple rows that have repeating data in some or all of the columns. • If you return single row result sets or result sets that do not contain repeating data, set this `WireProtocolMode=1` or performance may be degraded. • Oracle 10g database users may need to set `WireProtocolMode=2` to prevent performance degradation when using parameterized queries (prepared and callable statements) involving big tables.

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 6-10
- "CatalogOptions" on page 6-10
- "InsensitiveResultSetBufferSize" on page 6-10
- "MaxPooledStatements" on page 6-10
- "ResultSetMetaDataOptions" on page 6-11
- "ServerType" on page 6-11
- "WireProtocolMode" on page 6-11

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()`.

InsensitiveResultSetBufferSize

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

MaxPooledStatements

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

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 6-28 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

This driver can improve performance if you typically return data that is repeated in consecutive rows, for example, the data in column1/row1 is the same as the data in column1/row2, and so on. See WireProtocolMode.

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

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/loca
l/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 2-13 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:

ServerName	PortNumber
ServiceName	ServerType
	SID

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

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.

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

Oracle Driver Property	tnsnames.ora Attribute

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).

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 and is useful to specify connections to an Oracle Real Application Clusters (RAC) system rather than a specific Oracle instance.

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

Oracle Driver Property	tnsnames.ora 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 Database	Oracle Data Type	JDBC Data Type
Oracle8i and higher	BFILE	BLOB
	BLOB	BLOB
	CHAR	CHAR
	CLOB	CLOB
	DATE	TIMESTAMP
	FLOAT(n)	DOUBLE
	LONG	LONGVARCHAR
	long raw	LONGVARBINARY
	NCHAR	CHAR
	NCLOB	CLOB
	NUMBER (p, s)	DECIMAL
	NUMBER	DOUBLE
	NVARCHAR2	VARCHAR
	RAW	VARBINARY
Oracle9i and higher	TIMESTAMP	TIMESTAMP
	TIMESTAMP WITH LOCAL TIME ZONE	TIMESTAMP
	TIMESTAMP WITH TIME ZONE	TIMESTAMP
	VARCHAR2	VARCHAR
	XMLType	CLOB
Oracle10*g* only	BINARY_FLOAT	REAL
	BINARY_DOUBLE	DOUBLE

See "GetTypeInfo" on page B-1 for more information about data types.

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 Java Virtual Machine.
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 in the format specified by the Oracle `NLS_TIMESTAMP_TZ_FORMAT` session parameter.

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 retrieved 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.

XMLType Data Type

The Oracle driver supports tables containing columns specified as `XMLType` for Oracle9i and higher. The driver maps the Oracle `XMLType` data type to the JDBC CLOB data type. `XMLType` columns can be used in queries just like any other column type. The data from `XMLType` columns can be retrieved as a `String`, `Clob`, `CharacterStream`, or `AsciiStream`. When inserting or

updating XMLType columns, the data to be inserted or updated must be in the form of an 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 a parameter marker is used, the parameter value may be set using the setString, setClob, setCharacterStream, or setAsciiStream methods.

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 XMLTypeTbl 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 XMLTypeTbl values (?, xmltype(?))";
PreparedStatement prepStmt = con.prepareStatement(sql);
prepStmt.setInt(1, 2);
prepStmt.setString(2, xmlStr);
prepStmt.executeUpdate();
```

When the data from an XMLType column is retrieved as a Clob, the XMLType data cannot be updated using the Clob object. Calling the setString, setCharacterStream, or setAsciiStream methods of a Clob object returned from an XMLType column generates a SQLException.

REF CURSOR Data Type Support

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 retrieved 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;
```

Listing 6-2 REF Cursor 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();
}
```

Listing 6-3 REF Cursor 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();
   }
```

Character Set Conversion

To control which code page the driver uses to communicate with the Oracle server, use the `CodePageOverride` property. The code page specified by this property overrides the code page used by the driver to convert character data to the database character set. This option has no effect on how the driver converts character data to the national character set.

See Table 6-4 for a details about the different options available for character set conversions.

Table 6-4 Character Set Conversions for the Oracle Driver

Code PageOverride Value	Description
UTF8	The driver uses the UTF8 character set to communicate with the Oracle server. Using this value forces the driver to use UTF8 to communicate with the Oracle server. This can negatively affect performance.
SJIS	The driver uses the JA16SJIS character set to communicate with the Oracle server. The driver uses the SHIFT-JIS code page to convert character data to the JA16SJIS character set.

Table 6-4 Character Set Conversions for the Oracle Driver

Code PageOverride Value	Description
ENHANCED_SJIS	The driver uses a modified version of the JA16SJIS character set to communicate with the Oracle server. The ENHANCED_SJIS character set provides all the mappings of the SHIFT-JIS character set. 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 The driver uses the ENHANCED_SJIS code page to convert character data to the JA16SJIS character set.
ENHANCED_SJIS_ORACLE	The driver uses a modified version of the JA16SJIS character set to communicate with the Oracle server. The ENHANCED_SJIS_ORACLE character set provides all the mappings of the SHIFT-JIS character set. In addition, it maps the following MS-932 characters to the corresponding characters to which Oracle maps them: • \UFF5E Wave dash • \U2225 Double vertical line • \UFFE0 Cent sign • \UFF0D Minus sign • \UFFE1 Pound sign • \UFFE2 Not sign • \U301C Tilde The driver uses the ENHANCED_SJIS_ORACLE code page to convert character data to the JA16SJIS character set.
MS932	The driver uses the JA16SJIS character set to communicate with the Oracle server. The driver uses the MS932 code page to convert character data to the JA16SJIS character set. This value is provided for backward compatibility. Earlier versions of the driver used the MS932 code page when converting character data to JA16SJIS.

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 3.0 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 3.0 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 3.0 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 3.0-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 = ?"
```

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, mployeeInfo 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.

Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

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

Auto-Generated Keys Support

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

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 inform the driver to return the values of auto-generated keys:
  - `Statement.execute (String sql, int autoGeneratedKeys)`
  - `Statement.executeUpdate (String sql, int autoGeneratedKeys)`
- When using an `Insert` statement that contains parameters, the Oracle 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)
  ```

The application fetches the values of generated keys from the driver using the `Statement.getGeneratedKeys` method.

CHAPTER 7

The Sybase Driver

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

Database Version Support

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

- Sybase Adaptive Server 11.5 and 11.9
- Sybase Adaptive Server Enterprise 12.0, 12.5, and 12.5.1
- Sybase Adaptive Server Enterprise 15

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 BEA 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

To connect to a Sybase database, use the following URL format:

```
jdbc:bea:sybase://dbserver:port
```

Sybase Connection Properties

Table 6-1 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
BatchPerformanceWorkaround OPTIONAL	{true | false}. Determines the method used to execute batch operations. If set to true, the native Sybase batch mechanism is used. If set to false, the JDBC 3.0-compliant batch mechanism is used. 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. The default is false. See "Batch Inserts and Updates" on page 7-11.
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 Java Virtual Machine, for example, CodePageOverride=CP950.
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. The default is 0.
ConnectionRetryDelay OPTIONAL	The number of seconds the driver waits before retrying connections to a database serverwhen ConnectionRetryCount is set to a positive integer. The default is 3.
DatabaseName OPTIONAL	The name of the database to which you want to connect.

Table 7-1 Sybase Connection Properties

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).
LoginTimeout OPTIONAL	The maximum time in seconds that attempts to create a database connection will wait. A value of 0 specifies that the timeout is the default system timeout if there is one; otherwise it specifies that there is no timeout.
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.

Table 7-1 Sybase Connection Properties

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.

Table 7-1 Sybase Connection Properties

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. Direct—When the driver uses the Direct method, 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 all the 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. Cursor—When the driver uses the Cursor method, a server-side database cursor is requested. The rows are retrieved from the server in blocks when returning forward-only result sets. The JDBC Statement method setFetchSize can be used to control the number of rows that are retrieved for each request. Performance tests show that the value of setFetchSize significantly impacts performance when the Cursor method is used. There is no simple rule for determining the setFetchSize value that you should use. We recommend that you experiment with different setFetchSize values to find out 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. 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.
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

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.

InsensitiveResultSetBufferSize

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

MaxPooledStatements

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

that does not provide its own prepared statement pooling. When the driver's internal prepared statement pooling is enabled, the driver caches a certain number of prepared statements created by an application. For example, if the `MaxPooledStatements property` is set to 20, the driver caches the last 20 prepared statements created by the application. If the value set for this property is greater than the number of prepared statements used by the application, all prepared statements are cached.

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 7-12 for more information about returning ResultSet metadata.

Data Types

Table 6-2 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 Database	Sybase Data Type	JDBC Data Type
Sybase 11.5 and higher	binary	BINARY
	bit	BIT
	char	CHAR
	datetime	TIMESTAMP
	decimal	DECIMAL
	float	FLOAT
	image	LONGVARBINARY
	int	INTEGER
	money	DECIMAL
	nchar	CHAR
	numeric	NUMERIC
	nvarchar	VARCHAR
	real	REAL
	smalldatetime	TIMESTAMP
	smallint	SMALLINT
	smallmoney	DECIMAL
	sysname	VARCHAR
	text	LONGVARCHAR
	timestamp	VARBINARY
	tinyint	TINYINT
	varbinary	VARBINARY
	varchar	VARCHAR

Table 7-2 Sybase Data Types

Sybase Database	Sybase Data Type	JDBC Data Type
Sybase 12.5 and higher	date	DATE
	time	TIME
	unichar	CHAR
	univarchar	VARCHAR

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.

SQL Escape Sequences

See Appendix C, "SQL Escape Sequences for JDBC" 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 retrieve and update long data, specifically `LONGVARBINARY` and `LONGVARCHAR` data, using JDBC methods designed for Blobs and Clobs. When using these methods to update long data as Blobs or Clobs, the updates are made to the local copy of the data contained in the `Blob` or `Clob` object.

Retrieving and updating long data using JDBC methods designed for Blobs and Clobs provides some of the same 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 3.0-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 6-2.

Parameter Metadata Support

The Sybase driver supports returning parameter metadata for all types of SQL statements.

ResultSet MetaData Support

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

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

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

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

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

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

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

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

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

Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

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

Auto-Generated Keys Support

The Sybase driver supports retrieving the values of auto-generated keys. An auto-generated key returned by the Sybase driver is the value of an identity column

How you retrieve the values of auto-generated keys depends on whether the Insert statement you are using 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 inform the driver to return the values of auto-generated keys:
  - `Statement.execute (String sql, int autoGeneratedKeys)`
  - `Statement.executeUpdate (String sql, int autoGeneratedKeys)`

- When using an `Insert` statement that contains parameters, the Sybase 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)`

The application fetches the values of generated keys from the driver using the `Statement.getGeneratedKeys method`.

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 DataDirect Connect for JDBC drivers, which simplifies developing cross-database applications.

By default, Sybase does not evaluate `NULL` values in SQL equality (=) comparisons in an ANSI SQL-compliant manner. For example, the ANSI SQL specification defines that `col1=null` always evaluates to false. Using the default database setting (`ansinull=off`), if the value of `col1` in the following statement is `NULL`, the comparison evaluates to true instead of false:

```
SELECT * FROM table WHERE col1 = NULL
```

Setting `ansinull` to on changes the default database behavior so that SQL statements must use `IS NULL` instead of `=NULL`. For example, using the Sybase driver, if the value of `col1` in the following statement is `NULL`, the comparison evaluates to true:

```
SELECT * FROM table WHERE col1 IS NULL
```

To restore the default Sybase behavior for a connection, your application can 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*.

APPENDIX A

JDBC Support

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

JDBC Compatibility

Table A-1 shows compatibility among the JDBC specification versions, Java Virtual Machines, and the BEA WebLogic Type 4 JDBC drivers.

Table A-1 JDBC Compatibility

JDBC Version	Java 2 SDK	Drivers Compatible?
3.0	5.0	Yes

Note: WebLogic Server 9.0 requires a Java 2 SDK version 5.0.

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 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
byte[] getBytes (long, int)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
long length ()	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.

Table A-3 Blob Object

Blob Object (Continued) Methods	Version Introduced	Supported	Comments
long position (Blob, long)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 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 LONGVARBINARY data types.
long position (byte[], long)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 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 LONGVARBINARY data types.

Table A-3 Blob Object

Blob Object (Continued) Methods	Version Introduced	Supported	Comments
OutputStream setBinaryStream (long)	3.0	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
int setBytes (long, byte[])	3.0	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
int setBytes (long, byte[], int, int)	3.0	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
void truncate (long)	3.0	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.

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	No	Throws “unsupported method” exception.
Blob getBlob (int)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
Blob getBlob (String)	3.0	No	The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
boolean getBoolean (int)	1.0	Yes	
boolean getBoolean (String)	3.0	No	Throws “unsupported method” exception.
byte getByte (int)	1.0	Yes	
byte getByte (String)	3.0	No	Throws “unsupported method” exception.
byte [] getBytes (int)	1.0	Yes	
byte [] getBytes (String)	3.0	No	Throws “unsupported method” exception.
Clob getClob (int)	2.0 Core	Yes	

Table A-4 CallableStatement Object

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
Clob getClob (String)	3.0	No	Throws “unsupported method” exception.
Date getDate (int)	1.0	Yes	
Date getDate (int, Calendar)	2.0 Core	Yes	
Date getDate (String)	3.0	No	Throws “unsupported method” exception.
Date getDate (String, Calendar)	3.0	No	Throws “unsupported method” exception.
double getDouble (int)	1.0	Yes	
double getDouble (String)	3.0	No	Throws “unsupported method” exception.
float getFloat (int)	1.0	Yes	
float getFloat (String)	3.0	No	Throws “unsupported method” exception.
int getInt (int)	1.0	Yes	
int getInt (String)	3.0	No	Throws “unsupported method” exception.
long getLong (int)	1.0	Yes	
long getLong (String)	3.0	No	Throws “unsupported method” exception.
Object getObject (int)	1.0	Yes	
Object getObject (int, Map)	2.0 Core	Yes	Map ignored.
Object getObject (String)	3.0	No	Throws “unsupported method” exception.

Table A-4 CallableStatement Object

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
Object getObject (String, Map)	3.0	No	Throws “unsupported method” exception.
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	No	Throws “unsupported method” exception.
String getString (int)	1.0	Yes	
String getString (String)	3.0	No	Throws “unsupported method” exception.
Time getTime (int)	1.0	Yes	
Time getTime (int, Calendar)	2.0 Core	Yes	
Time getTime (String)	3.0	No	Throws “unsupported method” exception.
Time getTime (String, Calendar)	3.0	No	Throws “unsupported method” exception.
Timestamp getTimestamp (int)	1.0	Yes	
Timestamp getTimestamp (int, Calendar)	2.0 Core	Yes	
Timestamp getTimestamp (String)	3.0	No	Throws “unsupported method” exception.
Timestamp getTimestamp (String, Calendar)	3.0	No	Throws “unsupported method” exception.

Table A-4 CallableStatement Object

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
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	No	Throws “unsupported method” exception.
void registerOutParameter (String, int, int)	3.0	No	Throws “unsupported method” exception.
void registerOutParameter (String, int, String)	3.0	No	Throws “unsupported method” exception.
void setArray (int, Array)	2.0 Core	No	Throws “unsupported method” exception.
void setAsciiStream (String, InputStream, int)	3.0	No	Throws “unsupported method” exception.
void setBigDecimal (String, BigDecimal)	3.0	No	Throws “unsupported method” exception.
void setBinaryStream (String, InputStream, int)	3.0	No	Throws “unsupported method” exception.
void setBoolean (String, boolean)	3.0	No	Throws “unsupported method” exception.
void setByte (String, byte)	3.0	No	Throws “unsupported method” exception.

Table A-4 CallableStatement Object

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
void setBytes (String, byte [])	3.0	No	Throws “unsupported method” exception.
void setCharacterStream (String, Reader, int)	3.0	No	Throws “unsupported method” exception.
void setDate (String, Date)	3.0	No	Throws “unsupported method” exception.
void setDate (String, Date, Calendar)	3.0	No	Throws “unsupported method” exception.
void setDouble (String, double)	3.0	No	Throws “unsupported method” exception.
void setFloat (String, float)	3.0	No	Throws “unsupported method” exception.
void setInt (String, int)	3.0	No	Throws “unsupported method” exception.
void setLong (String, long)	3.0	No	Throws “unsupported method” exception.
void setNull (String, int)	3.0	No	Throws “unsupported method” exception.
void setNull (String, int, String)	3.0	No	Throws “unsupported method” exception.
void setObject (String, Object)	3.0	No	Throws “unsupported method” exception.
void setObject (String, Object, int)	3.0	No	Throws “unsupported method” exception.
void setObject (String, Object, int, int)	3.0	No	Throws “unsupported method” exception.
void setShort (String, short)	3.0	No	Throws “unsupported method” exception.

Table A-4 CallableStatement Object

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
void setString (String, String)	3.0	No	Throws "unsupported method" exception.
void setTime (String, Time)	3.0	No	Throws "unsupported method" exception.
void setTime (String, Time, Calendar)	3.0	No	Throws "unsupported method" exception.
void setTimestamp (String, Timestamp)	3.0	No	Throws "unsupported method" exception.
void setTimestamp (String, Timestamp, Calendar)	3.0	No	Throws "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 LONGVARCHAR data types.
Reader getCharacterStream ()	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
String getSubString (long, int)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.

Table A-5 Clob Object

Clob Object (Continued) Methods	Version Introduced	Supported	Comments
long length ()	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
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 LONGVARCHAR data types.
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 LONGVARCHAR data types.
OutputStream setAsciiStream (long)	3.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
Writer setCharacterStream (long)	3.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
int setString (long, String)	3.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
int setString (long, String, int, int)	3.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
void truncate (long)	3.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.

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	
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.
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.

Table A-6 Connection Object

Connection Object (Continued) Methods	Version Introduced	Supported	Comments
SQLWarning getWarnings ()	1.0	Yes	
boolean isClosed ()	1.0	Yes	
boolean isReadOnly ()	1.0	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.
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.

Table A-6 Connection Object

Connection Object (Continued) Methods	Version Introduced	Supported	Comments
PreparedStatement prepareStatement (String, int[])	3.0	No	Throws "unsupported method" exception.
PreparedStatement prepareStatement (String, String [])	3.0	No	Throws "unsupported method" exception.
void releaseSavepoint (Savepoint)	3.0	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries.
void rollback ()	1.0	Yes	
void rollback (Savepoint)	3.0	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 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.
void setHoldability (int)	3.0	Yes	Holdability parameter value is ignored.
void setReadOnly (boolean)	1.0	Yes	

Table A-6 Connection Object

Connection Object (Continued) Methods	Version Introduced	Supported	Comments
Savepoint setSavepoint ()	3.0	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. In addition, the DB2 driver only supports multiple nested savepoints for DB2 8.2 for Linux/UNIX/Windows.
Savepoint setSavepoint (String)	3.0	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 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	

Table A-7 DababaseMetaData Object

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
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.
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	
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	

Table A-7 DababaseMetaData Object

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
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	
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	

Table A-7 DababaseMetaData Object

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
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	
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	

Table A-7 DababaseMetaData Object

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
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	
int getSQLStateType ()	3.0	Yes	
String getStringFunctions ()	1.0	Yes	
ResultSet getSuperTables (String, String, String)	3.0	No	Throws “unsupported method” exception.
ResultSet getSuperTypes (String, String, String)	3.0	No	Throws “unsupported method” exception.
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	

Table A-7 DababaseMetaData Object

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
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	
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	

Table A-7 DababaseMetaData Object

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
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	
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	

Table A-7 DababaseMetaData Object

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
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	
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	

Table A-7 DababaseMetaData Object

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
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	
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	

Table A-7 DababaseMetaData Object

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
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	
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 V5R3 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 V5R3 for iSeries.

Table A-9 ParameterMetaData Object

ParameterMetaData Object (Continued)	Version Introduced	Supported	Comments
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 V5R3 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 V5R3 for iSeries.
int getPrecision (int)	3.0	Yes	The DB2 driver supports parameter metadata for stored procedures for DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 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 V5R3 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 V5R3 for iSeries.

Table A-9 ParameterMetaData Object

ParameterMetaData Object (Continued)	Version Introduced	Supported	Comments
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 V5R3 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.

Table A-10 PreparedStatement Object

PreparedStatement Object (Continued) Methods	Version Introduced	Supported	Comments
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 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries.
void setBlob (int, Blob)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
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 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries.
void setCharacterStream (int, Reader, int)	2.0 Core	Yes	

Table A-10 PreparedStatement Object

PreparedStatement Object (Continued) Methods	Version Introduced	Supported	Comments
void setClob (int, Clob)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
void setDate (int, Date)	1.0	Yes	
void setDate (int, Date, Calendar)	2.0 Core	Yes	
void setDouble (int, double)	1.0	Yes	
void setFloat (int, float)	1.0	Yes	
void setInt (int, int)	1.0	Yes	
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	
void setQueryTimeout (int)	1.0	Yes	Throws "unsupported method" exception for DB2 and Informix.
void setRef (int, Ref)	2.0 Core	No	Throws "unsupported method" exception.
void setShort (int, short)	1.0	Yes	

Table A-10 PreparedStatement Object

PreparedStatement Object (Continued) Methods	Version Introduced	Supported	Comments
void setString (int, String)	1.0	Yes	
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	
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 (Continued) 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 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 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 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries when retrieving BLOB data.
Blob getBlob (int)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.

Table A-12 ResultSet Object

ResultSet Object (Continued) Methods	Version Introduced	Supported	Comments
Blob getBlob (String)	2.0 Core	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
boolean getBoolean (int)	1.0	Yes	
boolean getBoolean (String)	1.0	Yes	
byte getByte (int)	1.0	Yes	
byte getByte (String)	1.0	Yes	
byte [] getBytes (int)	1.0	Yes	The DB2 driver supports for all DB2 versions when retrieving BINARY, VARBINARY, and LONGVARBINARY data. The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries when retrieving BLOB data.

Table A-12 ResultSet Object

ResultSet Object (Continued) Methods	Version Introduced	Supported	Comments
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 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 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 LONGVARCHAR data types.
Clob getClob (String)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
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	

Table A-12 ResultSet Object

ResultSet Object (Continued) Methods	Version Introduced	Supported	Comments
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	
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.

Table A-12 ResultSet Object

ResultSet Object (Continued) Methods	Version Introduced	Supported	Comments
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	
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.

Table A-12 ResultSet Object

ResultSet Object (Continued) Methods	Version Introduced	Supported	Comments
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	
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	

Table A-12 ResultSet Object

ResultSet Object (Continued) Methods	Version Introduced	Supported	Comments
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.
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	

Table A-12 ResultSet Object

ResultSet Object (Continued) Methods	Version Introduced	Supported	Comments
void updateBlob (int, Blob)	3.0	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
void updateBlob (String, Blob)	3.0	Yes	The DB2 driver only supports with DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 for iSeries. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
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 (Continued) Methods	Version Introduced	Supported	Comments
void updateClob (int, Clob)	3.0	Yes	
void updateClob (String, Clob)	3.0	Yes	
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	
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	

Table A-12 ResultSet Object

ResultSet Object (Continued) Methods	Version Introduced	Supported	Comments
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	

Table A-13 ResultSetMetaData Object

ResultSetMetaData Object (Continued)	Version Introduced	Supported	Comments
int getColumnCount ()	1.0	Yes	
int getColumnDisplaySize (int)	1.0	Yes	
String getColumnLabel (int)	1.0	Yes	
String getColumnName (int)	1.0	Yes	
int getColumnType (int)	1.0	Yes	
String getColumnTypeName (int)	1.0	Yes	
int getPrecision (int)	1.0	Yes	
int getScale (int)	1.0	Yes	
String getSchemaName (int)	1.0	Yes	
String getTableName (int)	1.0	Yes	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	

Table A-13 ResultSetMetaData Object

ResultSetMetaData Object (Continued)	Version Introduced	Supported	Comments
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 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 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

Statement Object (Continued) 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	No	Throws "unsupported method" exception.
boolean execute (String, String [])	3.0	No	Throws "unsupported method" exception.

Table A-15 Statement Object

Statement Object (Continued) Methods	Version Introduced	Supported	Comments
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	No	Throws "unsupported method" exception.
int executeUpdate (String, String [])	3.0	No	Throws "unsupported method" exception.
Connection getConnection ()	2.0 Core	Yes	
int getFetchDirection ()	2.0 Core	Yes	
int getFetchSize ()	2.0 Core	Yes	
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.

Table A-15 Statement Object

Statement Object (Continued) Methods	Version Introduced	Supported	Comments
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 v8.1 and v8.2 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	
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.

Table A-15 Statement Object

Statement Object (Continued) Methods	Version Introduced	Supported	Comments
void setFetchDirection (int)	2.0 Core	Yes	
void setFetchSize (int)	2.0 Core	Yes	
void setMaxFieldSize (int)	1.0	Yes	
void setMaxRows (int)	1.0	Yes	
void setQueryTimeout (int)	1.0	Yes	The DB2 driver supports setting a timeout value, in seconds, for a statement with DB2 v8.1 and v8.2 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, DB2 v7.*x* and v8.1 for z/OS, and DB2 for iSeries.

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, DB2 v7.*x* and v8.1for z/OS, and DB2 for iSeries.

XAResource Object

Table A-19 XAResource Object

XAResource Object Methods	Version Introduced	Supported	Comments
(all)	2.0 Optional	Yes	Support for all drivers except for DB2 UDB 7.x, DB2 OS/390 7.x and z/OS, and DB2 iSeries.

APPENDIX B

GetTypeInfo

The following tables provide results returned from the `DataBaseMetaData.getTypeInfo` method for all of the BEA 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 v7.x, v8.1, and v8.2 for Linux/UNIX/Windows and DB2 V5R2 and V5R3 for iSeries

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	

* Supported only for DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 V5R2 and V5R3 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

Table B-1 getTypeInfo for DB2 (Continued)

TYPE_NAME = char for bit data	
AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = length DATA_TYPE = -2 (BINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = char for bit data MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 254 (DB2 for Linux/UNIX/Windows), 255 (DB2 for z/OS), 32765 (DB2 for iSeries) SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = clob	
AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = length DATA_TYPE = 2005 (CLOB) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = clob MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32700 (DB2 v7.*x* for Linux/UNIX/Windows), 2147483647 (DB2 v8.1, v8.2 for Linux/UNIX/Windows, DB2 for z/OS, DB2 for iSeries) SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = date	
AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 91 (DATE) FIXED_PREC_SCALE = false LITERAL_PREFIX = {d' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = date MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Table B-1 getTypeInfo for DB2 (Continued)

TYPE_NAME = dbclob *	
AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = length (DB2 for Linux/UNIX/Windows, DB2 for z/OS, DB2 V5R1 for iSeries) CCSID 13488 (DB2 V5R2, V5R3 for iSeries) DATA_TYPE = 2005 (DBCLOB) FIXED_PREC_SCALE = false LITERAL_PREFIX = ’ LITERAL_SUFFIX = ’ LOCAL_TYPE_NAME = dbclob MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
* Supported only for DB2 v8.1 and v8.2 for Linux/UNIX/Windows, DB2 for z/OS, and DB2 for iSeries.	
TYPE_NAME = decimal	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = precision,scale DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = decimal MAXIMUM_SCALE = 31	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 31 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = double	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 8 (DOUBLE) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = double MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 15 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

Table B-1 getTypeInfo for DB2 (Continued)

TYPE_NAME = float	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 6 (FLOAT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = float MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 15 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = integer	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = integer MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = long varchar	
AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = -1 (LONGVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ’ LITERAL_SUFFIX = ’ LOCAL_TYPE_NAME = long varchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32700 (DB2 for Linux/UNIX/Windows, DB2 for iSeries)*, 32704 (DB2 for z/OS) * SEARCHABLE = 1 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 CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -4 (LONGVARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = X' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = long varchar for bit data MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32700 (DB2 for Linux/UNIX/Windows), 32698 (DB2 for z/OS), 32740 (DB2 for iSeries) SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = numeric	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = precision, scale DATA_TYPE = 2 (NUMERIC) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = numeric MAXIMUM_SCALE = 31	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX =10 PRECISION = 31 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = real	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 7 (REAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = float(4) MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 7 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

Table B-1 getTypeInfo for DB2 (Continued)

TYPE_NAME = rowid *	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
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 v7.*x* and v8.1 for z/OS and DB2 V5R2 and V5R3 for iSeries.	
**Supported except for WHERE ... LIKE.	
YPE_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 = time	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 92 (TIME)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {t’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = time	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-1 getTypeInfo for DB2 (Continued)

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 = 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 v8.1, v8.2 for Linux/UNIX/Windows), 32698 (DB2 for z/OS), 32739 (DB2 for iSeries) SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = 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 v8.1, v8.2 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

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 CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 2004 (BLOB) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = blob MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = boolean	
AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -7 (BIT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = boolean MAXIMUM_SCALE = 0	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = byte	
AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -4 (LONGVARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = byte MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Table B-2 getTypeInfo for Informix

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 = 32766
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = char	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = clob	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = clob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = date	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {d’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table 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 CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = int8 MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = integer	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = integer MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = lvarchar	
AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL (Informix 9.2, 9.3), 'max length' (Informix 9.4, 10) DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ’ LITERAL_SUFFIX = ’ LOCAL_TYPE_NAME = lvarchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2048 (Informix 9.2, 9.3), 32739 (Informix 9.4, 10) SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

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 CASE_SENSITIVE = false CREATE_PARAMS = start DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = serial MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = serial8	
AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = serial8 MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = smallfloat	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 7 (REAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = smallfloat MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 7 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

Table B-2 getTypeInfo for Informix

TYPE_NAME = smallint	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 5 (SMALLINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = smallint MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 5 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = text	
AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = -1 (LONGVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ’ LITERAL_SUFFIX = ’ LOCAL_TYPE_NAME = text MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varchar	
AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = max length DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ’ LITERAL_SUFFIX = ’ LOCAL_TYPE_NAME = varchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 254 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Oracle Driver

Table B-3 provides getTypeInfo results for Oracle8i, Oracle9i, and Oracle10g. Table B-4 provides additional results for data types specific to Oracle 9i and Oracle 10g. See “The Oracle Driver.”

Table B-3 getTypeInfo for Oracle (Oracle 8i, 9i, and 10g)

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 = 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	

Table B-3 getTypeInfo for Oracle (Oracle 8i, 9i, and 10g)

TYPE_NAME = clob	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = clob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = date	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	
TYPE_NAME = long	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = long	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-3 getTypeInfo for Oracle (Oracle 8i, 9i, and 10g)

TYPE_NAME = long raw	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = long raw	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = nchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = length	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 2000
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 = nclob	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nclob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-3 getTypeInfo for Oracle (Oracle 8i, 9i, and 10g)

TYPE_NAME = number (p, s)	
AUTO_INCREMENT = false	MINIMUM_SCALE = -84
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = precision, scale	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = number	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 127	
TYPE_NAME = number	
AUTO_INCREMENT = false	MINIMUM_SCALE = -84
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (FLOAT)	PRECISION = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = number	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 127	
TYPE_NAME = nvarchar2	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nvarchar2	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-3 getTypeInfo for Oracle (Oracle 8i, 9i, and 10g)

TYPE_NAME = raw	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION = 2000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = raw	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = 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 = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar2	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Oracle9i and Oracle 10g Only

Table B-4 getTypeInfo for Oracle (Oracle 8i, 9i, and 10g)

TYPE_NAME = bfile	
AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 2004 (BLOB) FIXED_PREC_SCALE = false LITERAL_PREFIX = ’ LITERAL_SUFFIX = ’ LOCAL_TYPE_NAME = bfile 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 = blob	
AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 2004 (BLOB) FIXED_PREC_SCALE = false LITERAL_PREFIX = ’ LITERAL_SUFFIX = ’ LOCAL_TYPE_NAME = blob MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = 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 = 2000 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Table B-4 getTypeInfo for Oracle (Oracle 8i, 9i, and 10g)

TYPE_NAME = clob	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = clob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = date	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	
TYPE_NAME = long	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = long	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-4 getTypeInfo for Oracle (Oracle 8i, 9i, and 10g)

TYPE_NAME = long raw	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = long raw	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = nchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = length	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 2000
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 = nclob	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nclob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-4 getTypeInfo for Oracle (Oracle 8i, 9i, and 10g)

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
TYPE_NAME = number	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 3 (FLOAT) 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

Table B-4 getTypeInfo for Oracle (Oracle 8i, 9i, and 10g)

TYPE_NAME = raw	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION = 2000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = raw	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = 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 = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar2	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

SQL Server Driver

Table B-5 provides getTypeInfo results for Microsoft SQL Server 7 and Microsoft SQL Server 2000. Table B-6 provides additional results for Microsoft SQL Server 2000 only. See“The MS SQL Server Driver.”

Table B-5 getTypeInfo for SQL Server (Microsoft SQL Server 7 and 2000)

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	
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	

Table B-5 getTypeInfo for SQL Server (Microsoft SQL Server 7 and 2000)

TYPE_NAME = datetime	
AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = datetime MAXIMUM_SCALE = 3	MINIMUM_SCALE = 3 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 23 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = decimal	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = precision,scale DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = decimal MAXIMUM_SCALE = 28 (SQL Server 7), 38 (SQL Server 2000) *	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 38 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
* Configurable server option for Microsoft SQL Server 2000.	
TYPE_NAME = decimal() identity	
AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = precision DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = decimal() identity MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 28 (SQL Server 7.0), 38 (SQL Server 2000) SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

Table B-5 getTypeInfo for SQL Server (Microsoft SQL Server 7 and 2000)

TYPE_NAME = float	
AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 2
DATA_TYPE = 6 (FLOAT)	PRECISION = 53
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	
TYPE_NAME = image	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = image	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = int	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = int	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

Table B-5 getTypeInfo for SQL Server (Microsoft SQL Server 7 and 2000)

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

Table B-5 getTypeInfo for SQL Server (Microsoft SQL Server 7 and 2000)

TYPE_NAME = ntext	
AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -1 (LONGVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = N' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = ntext MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 1073741823 SEARCHABLE = 1 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = numeric	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = precision,scale DATA_TYPE = 2 (NUMERIC) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = numeric MAXIMUM_SCALE = 28 (SQL Server 7), 38 (SQL Server 2000) *	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 28 (SQL Server 7), 38 (SQL Server 2000) * SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
* Configurable server option for Microsoft SQL Server 2000.	
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) SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

Table B-5 getTypeInfo for SQL Server (Microsoft SQL Server 7 and 2000)

TYPE_NAME = nvarchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nvarchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
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-5 getTypeInfo for SQL Server (Microsoft SQL Server 7 and 2000)

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-5 getTypeInfo for SQL Server (Microsoft SQL Server 7 and 2000)

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	
TYPE_NAME = timestamp	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-5 getTypeInfo for SQL Server (Microsoft SQL Server 7 and 2000)

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
TYPE_NAME = uniqueidentifier	
AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 1(CHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ’ LITERAL_SUFFIX = ’ LOCAL_TYPE_NAME = uniqueidentifier MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 36 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Table B-5 getTypeInfo for SQL Server (Microsoft SQL Server 7 and 2000)

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 = 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	

Microsoft SQL Server 2000 Only

Table B-6 getTypeInfo for SQL Server (Microsoft SQL Server 2000 Only)

TYPE_NAME = bigint	
AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = bigint MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = bigint identity	
AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = bigint identity MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 0 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = sql_variant	
AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = sql_variant MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 8000 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Sybase Driver

Table B-7 provides getTypeInfo results for all Sybase databases supported by the Sybase driver (see Chapter 7, "The Sybase Driver").

Table B-7 getTypeInfo for Sybase

TYPE_NAME = binary	
AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = length DATA_TYPE = -2 (BINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = 0x LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = binary MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 255 (Sybase 11.*x*, 12.0) 2048 (Sybase 12.5 and higher) * SEARCHABLE = 2 SQL_DATA_TYPE = 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 CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -7 (BIT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = bit MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 0 NUM_PREC_RADIX = NULL PRECISION = 1 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = char	
AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = length DATA_TYPE = 1 (CHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ’ LITERAL_SUFFIX = ’ LOCAL_TYPE_NAME = char MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 255 (Sybase 11.*x*, 12.0) 2048 (Sybase 12.5 and higher) * SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

Table B-7 getTypeInfo for Sybase

TYPE_NAME = date *

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 91 (DATE)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = date
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 10
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

* Supported only for Sybase 12.5.1 and higher.

TYPE_NAME = datetime

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 93 (TIMESTAMP)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = datetime
MAXIMUM_SCALE = 6
MINIMUM_SCALE = 6
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 23
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = decimal

AUTO_INCREMENT = false
CASE_SENSITIVE = false
CREATE_PARAMS = precision,scale
DATA_TYPE = 3 (DECIMAL)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = decimal
MAXIMUM_SCALE = 38
MINIMUM_SCALE = 0
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 38
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = false

Table B-7 getTypeInfo for Sybase

TYPE_NAME = float

AUTO_INCREMENT = false
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 6 (FLOAT)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = float
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = 10
PRECISION = 15
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = false

TYPE_NAME = image

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = -4 (LONGVARBINARY)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = 0x
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = image
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 2147483647
SEARCHABLE = 1
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = int

AUTO_INCREMENT = false
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 4 (INTEGER)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = int
MAXIMUM_SCALE = 0
MINIMUM_SCALE = 0
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 10
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = false

Table B-7 getTypeInfo for Sybase

TYPE_NAME = money

AUTO_INCREMENT = false
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 3 (DECIMAL)
FIXED_PREC_SCALE = true
LITERAL_PREFIX = $
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = money
MAXIMUM_SCALE = 4
MINIMUM_SCALE = 4
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 19
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = false

TYPE_NAME = nchar

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = NULL
DATA_TYPE = 1 (CHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = nchar
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION =
255 (Sybase 11.*x*, 12.0)
2048 (Sybase 12.5 and higher) *
SEARCHABLE = 3
SQL_DATA_TYPE = 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
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 7 (REAL)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = real
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = 10
PRECISION = 7
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = false

Table B-7 getTypeInfo for Sybase

TYPE_NAME = smalldatetime

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 93 (TIMESTAMP)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = smalldatetime
MAXIMUM_SCALE = 3
MINIMUM_SCALE = 3
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 16
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = smallint

AUTO_INCREMENT = false
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 5 (SMALLINT)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = smallint
MAXIMUM_SCALE = 0
MINIMUM_SCALE = 0
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 5
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = false

TYPE_NAME = smallmoney

AUTO_INCREMENT = false
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 3 (DECIMAL)
FIXED_PREC_SCALE = true
LITERAL_PREFIX = $
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = smallmoney
MAXIMUM_SCALE = 4
MINIMUM_SCALE = 4
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 10
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = false

Table B-7 getTypeInfo for Sybase

TYPE_NAME = sysname

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = max length
DATA_TYPE = 12 (VARCHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = sysname
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 30
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = text

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = NULL
DATA_TYPE = -1 (LONGVARCHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = text
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 2147483647
SEARCHABLE = 1
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = time *

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 92 (TIME)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = time
MAXIMUM_SCALE = 3
MINIMUM_SCALE = 0
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 12
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

* Supported only for Sybase 12.5.1 and higher.

Table B-7 getTypeInfo for Sybase

TYPE_NAME = timestamp

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = -3 (VARBINARY)
FIXED_PREC_SCALE = false
LITERAL_PREFIX =0x
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = timestamp
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 8
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = tinyint

AUTO_INCREMENT = false
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = -6 (TINTYINT)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = tinyint
MAXIMUM_SCALE = 0
MINIMUM_SCALE = 0
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 3
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = true

TYPE_NAME = unichar *

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = length
DATA_TYPE = 1 (CHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = unichar
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION =2048
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

* Supported only for Sybase 12.5 and higher.

Table B-7 getTypeInfo for Sybase

TYPE_NAME = univarchar *

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = max length
DATA_TYPE = 12 (VARCHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = univarchar
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 2048
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

* Supported only for Sybase 12.5 and higher.

TYPE_NAME = varbinary

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = max length
DATA_TYPE = -3 (VARBINARY)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = 0x
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = varbinary
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION =
255 (Sybase 11.*x*, 12.0)
2048 (Sybase 12.5 and higher) *
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

* For Sybase 12.5.1 and higher, precision is determined by the server page size.

TYPE_NAME = varchar

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = max length
DATA_TYPE = 12 (VARCHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = varchar
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION =
255 (Sybase 11.*x*, 12.0)
2048 (Sybase 12.5 and higher) *
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

* For Sybase 12.5.1 and higher, precision is determined by the server page size.

APPENDIX C

SQL Escape Sequences for JDBC

Language features, such as outer joins and scalar function calls, are commonly implemented by database systems. The syntax for these features is often database-specific, even when a standard syntax has been defined. JDBC defines escape sequences that contain standard syntaxes for the following language features:

- Date, time, and timestamp literals
- Scalar functions such as numeric, string, and data type conversion functions
- Outer joins
- Procedure calls

The escape sequence used by JDBC is:

```
{extension}
```

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

Table C-2 Scalar Functions Supported

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

Table C-2 Scalar Functions Supported

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

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

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 BEA 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
SQL Server	Left outer joins Right outer joins Full outer joins Nested outer joins
Sybase	Left outer joins Right outer joins Nested outer joins

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, a schema 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

BEA WebLogic JDBC Spy is a wrapper that wraps a BEA 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 1.22-, JDBC 2.0-, and JDBC 3.0-compliant, including support for the JDBC 2.0 Optional Package.
- Logging works with all BEA WebLogic Type 4 JDBC drivers.
- Logging is consistent, regardless of which BEA 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, typically C:\bea\weblogic81.

2. In the WebLogic Server Administration Console or in the config.xml 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 JDBCConnectionPool →Configuration →General tab, add the spyAttributes to the end of the existing URL. For example:

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

   Alternatively, in the config.xml file, update the URL in the JDBC data source entry. For example:

   ```
   <JDBCConnectionPool Name="datasource"
   Password="{3DES}0zvizFP1" Targets="myserver"
   InitialCapacity="10" MaxCapacity="10"
   DriverName="weblogic.jdbcx.db2.DB2DataSource"
   Properties="user=john;DatabaseName=wls;PortNumber=50000;
   ServerName=db2host;batchPerformanceWorkaround=true"
   URL="jdbc:bea:DB2://db2host:50000;user=john;
   spyAttributes=(log=(file)d:\spy.log;timestamp=yes)"
   KeepXAConnTillTxComplete="true"
   />
   ```

3. Stop and restart WebLogic Server.

BEA 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.
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.

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