Oracle® Complex Event Processing

Type 4 JDBC Drivers

Release 10*g*R3 (10.3)

September 2008

ORACLE®

Oracle Complex Event Processing Type 4 JDBC Drivers, Release 10*g*R3 (10.3)

Contents

Introduction and Roadmap

This section describes the contents and organization of this guide—*Oracle Complex Event Processing Type 4 JDBC Drivers*.

Note: In this section, *Oracle Complex Event Processing* is also referred to as *Oracle CEP*, for simplicity.

- "Document Scope and Audience" on page 1-1
- "Oracle CEP Documentation Set" on page 1-2
- "Guide to This Document" on page 1-2
- "Samples for the Oracle CEP Application Developer" on page 1-3

Document Scope and Audience

This document is a resource for software developers who develop event driven real-time applications. It also contains information that is useful for business analysts and system architects who are evaluating Oracle CEP or considering the use of Oracle CEP for a particular application.

The topics in this document are relevant during the design, development, configuration, deployment, and performance tuning phases of event driven applications. The document also includes topics that are useful in solving application problems that are discovered during test and pre-production phases of a project.

It is assumed that the reader is familiar with the Java programming language and Spring.

Oracle CEP Documentation Set

This document is part of a larger Oracle CEP documentation set that covers a comprehensive list of topics. The full documentation set includes the following documents:

- Oracle CEP Getting Started
- Oracle CEP Application Development Guide
- Oracle CEP Administration and Configuration Guide
- Oracle CEP EPL Reference Guide
- Oracle CEP Reference Guide
- Oracle CEP Release Notes
- Oracle CEP Visualizer Help
- Oracle CEP Type 4 JDBC Drivers

See the main Oracle CEP documentation page for further details.

Guide to This Document

This document is organized as follows:

- This chapter, Chapter 1, "Introduction and Roadmap," introduces the organization of this guide and the Oracle CEP documentation set and samples.
- Chapter 2, "Using The Type 4 JDBC Drivers,"provides information about connecting to a database with Oracle CEP Type 4 JDBC drivers.
- Chapter 3, "The MS SQL Server Type 4 JDBC Driver," provides detailed information about the Microsoft SQL Server 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 Type 4 JDBC drivers.
- Appendix C, "SQL Escape Sequences for JDBC," describes the scalar functions supported for the Type 4 JDBC drivers. Your data store may not support all of these functions.

Samples for the Oracle CEP Application Developer

In addition to this document, Oracle provides a variety of code samples for Oracle CEP application developers. The examples illustrate Oracle CEP in action, and provide practical instructions on how to perform key development tasks.

Oracle recommends that you run some or all of the examples before programming and configuring your own event driven application.

Note: When you initially install Oracle CEP, you must chose the `Custom` option to also install the examples. The `Typical` option does *not* include the examples.

If you previously installed Oracle CEP using the `Typical` option, and you now want to also install the examples, re-run the Oracle CEP installation process and specify the same Oracle CEP home directory; a later step in the installation process allows you to then install just the examples.

The examples are distributed in two ways:

- Pre-packaged and compiled in their own domain so you can immediately run them after you install the product.
- Separately in a Java source directory so you can see a typical development environment setup.

The following four examples are provided in both their own domain and as Java source in this release of Oracle CEP:

- HelloWorld—Example that shows the basic elements of an Oracle CEP application. See Hello World Example for additional information.

  The HelloWorld domain is located in `*ORACLE_CEP_HOME*\ocep_10.3\samples\domains\helloworld_domain`, where `*ORACLE_CEP_HOME*` refers to the Oracle CEP installation directory, such as `c:\oracle_cep`.

  The HelloWorld Java source code and configuration files are located in `*ORACLE_CEP_HOME*\ocep_10.3\samples\source\applications\helloworld`.

- ForeignExchange (FX)—Example that includes multiple adapters, streams, and complex event processor with a variety of EPL rules, all packaged in the same Oracle CEP application. See Foreign Exchange (FX) Example for additional information.

  The ForeignExchange domain is located in `*ORACLE_CEP_HOME*\ocep_10.3\samples\domains\fx_domain`, where

`ORACLE_CEP_HOME` refers to the Oracle CEP installation directory, such as `c:\oracle_cep`.

The ForeignExchange Java source code and configuration files are located in *`ORACLE_CEP_HOME`*`\ocep_10.3\samples\source\applications\fx`.

- Signal Generation—Example that receives simulated market data and verifies if the price of a security has fluctuated more than two percent, and then detects if there is a *trend* occurring by keeping track of successive stock prices for a particular symbol.See Signal Generation Example for additional information.

  The Signal Generation domain is located in *`ORACLE_CEP_HOME`*`\ocep_10.3\samples\domains\signalgeneration_domain`, where *`ORACLE_CEP_HOME`* refers to the Oracle CEP installation directory, such as `c:\oracle_cep`.

  The Signal Generation Java source code and configuration files are located in *`ORACLE_CEP_HOME`*`\ocep_10.3\samples\source\applications\signalgeneration`.

- Record and Playback—Example that shows how to configure the recording and playback of events to a persistent event store, as well as how to use the built-in HTTP pub-sub adapter to publish messages to a channel. See Event Record and Playback Example for additional information.

  The Record and Playback domain is located in *`ORACLE_CEP_HOME`*`\ocep_10.3\samples\domains\recplay_domain`, where *`ORACLE_CEP_HOME`* refers to the Oracle CEP installation directory, such as `c:\oracle_cep`.

  The Record and Playback Java source code and configuration files are located in *`ORACLE_CEP_HOME`*`\ocep_10.3\samples\source\applications\recplay`.

Using The Type 4 JDBC Drivers

The Type 4 JDBC drivers from DataDirect provide JDBC high-performance access through Oracle CEP to industry-leading data stores across the Internet and intranets. The 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 Oracle CEP Type 4 JDBC drivers from DataDirect are proven drivers that:

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

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

- "Required Permissions for the Java Security Manager" on page 2-10
- "Unicode Support" on page 2-13
- "Unicode Support" on page 2-13
- "Error Handling" on page 2-13

JDBC Specification Compliance

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

- Connection validation
- Client information storage and retrieval
- Auto-load driver classes (when using Java SE 6)

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

Installation

Type 4 JDBC drivers are automatically installed with Oracle CEP and are automatically added to your classpath on the server.

Supported Databases

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

Table 2-1 Supported Databases

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

Connecting Through JDBC Data Sources

To use the Type 4 JDBC drivers, you create a JDBC data source in your Oracle CEP 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.

For information about JDBC and data sources in Oracle CEP, see Configuring Access to a Relational Database.

Specifying Connection Properties

You specify connection properties for connections in a data source Oracle CEP `config.xml` file. Connection properties vary by DBMS. For the list of the connection properties specific to each Type 4 JDBC driver, see the appropriate driver chapter:

- For the MS SQL Server driver, see "SQL Server Connection Properties" on page 3-4.

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 Type 4 JDBC Drivers support the `LoginTimeout` connection property. When you specify a `LoginTimeout` connection property and the connection is not created immediately, the request waits for the time you specify. If the connection cannot be created within the time specified, the driver throws an SQL exception.

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

- "SQL Server Connection Properties" on page 3-4

Using IP Addresses

The Type 4 JDBC drivers support Internet Protocol (IP) addresses in IPv4 and IPv6 format. IPv6 addresses are only supported when connecting to certain database versions (as shown in Table 2-2). In addition, to connect to IPv6 addresses, the driver machine requires J2SE 5.0 or higher on Windows and J2SE 1.4 on UNIX/Linux.

Table 2-2 IP Address Formats Supported by the Type 4 JDBC Drivers

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

If your network supports named servers, the server name specified in the connection URL or data source can resolve to an IPv4 or IPv6 address. Alternatively, you can specify addresses using IPv4 or IPv6 format in the server name portion of the connection URL. You also can specify addresses in either format using the `ServerName` data source property.

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

In addition to the normal IPv6 format, the Type 4 JDBC drivers support IPv6 alternative formats for compressed and IPv4/IPv6 combination addresses.

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

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

Using Security

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

Authentication

On most computer systems, a password is used to prove a user's identity. This password often is transmitted over the network and can possibly be intercepted by malicious hackers. Because this password is the one secret piece of information that identifies a user, anyone knowing a user's password can effectively be that user. Authentication methods protect the identity of the user. Type 4 JDBC drivers support the following authentication methods:

- User ID/password authentication authenticates the user to the database using a database user name and password.
- Kerberos is a trusted third-party authentication service. The drivers support both Windows Active Directory Kerberos and MIT Kerberos implementations for Oracle. For SQL Server, the driver supports Windows Active Directory Kerberos only.

- Client authentication uses the user ID of the user logged onto the system on which the driver is running to authenticate the user to the database. The database server relies on the client to authenticate the user and does not provide additional authentication.
- NTLM authentication is a single sign-on authentication method for Windows environments. This method provides authentication from Windows clients only.

Table 2-3 shows the authentication methods supported by the Type 4 JDBC drivers.

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

Driver	UserID/ Password	Kerberos	Client	NTLM
Microsoft SQL Server	X	X[1]		X

1. Supported for Microsoft SQL Server 2000 and higher.

Kerberos Authentication Requirements

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

Table 2-4 Kerberos Authentication Requirements for the Drivers

Component	Requirements
Database server	The database server must be running one of the following databases: **Microsoft SQL Server:** • Microsoft SQL Server 2005 • Microsoft SQL Server 2000 • Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher
Kerberos server	The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos Key Distribution Center (KDC). If using Windows Active Directory, this machine is also the domain controller. **Microsoft SQL Server:** Network authentication must be provided by Windows Active Directory on one of the following operating systems: • Windows Server 2003 • Windows 2000 Server Service Pack 3 or higher
Client	J2SE 1.4.2 or higher must be installed.

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

NTLM Authentication Requirements

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

Table 2-5 NTLM Authentication Requirements for the Drivers

Component	Requirements
Database server	The database server must be administered by the same domain controller that administers the client and must be running one of the following databases: **Microsoft SQL Server:** • Microsoft SQL Server 2005 • Microsoft SQL Server 2000 Service Pack 3 or higher • Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher
Domain controller	The domain controller must administer both the database server and the client. Network authentication must be provided by NTLM on one of the following operating systems: • Windows Server 2003 • Windows 2000 Server Service Pack 3 or higher
Client	The client must be administered by the same domain controller that administers the database server and must be running on one of the following operating systems: • Windows Vista • Windows Server 2003 • Windows XP Service Pack 1 or higher • Windows 2000 Service Pack 4 or higher • Windows NT 4.0 In addition, J2SE 1.3 or higher must be installed.

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

Data Encryption Across the Network

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

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

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

Type 4 JDBC drivers support the following encryption method:

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

Table 2-6 shows the data encryption methods supported by the Type 4 JDBC drivers.

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

Driver	Database-Specific	SSL
Microsoft SQL Server		X[1]

1. Supported for Microsoft SQL Server 2000 and higher.

SSL Encryption

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

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

See the individual driver chapters for details about configuring SSL.

SSL Server Authentication

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

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

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

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

  and

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

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

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

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

  and

  ```
  TrustStorePassword=MyTruststorePassword
  ```

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

Alternatively, you can configure the Type 4 JDBC drivers to trust any certificate sent by the server, even if the issuer is not a trusted CA. Allowing a driver to trust any certificate sent from the server is useful in test environments because it eliminates the need to specify truststore information on each client in the test environment. If the driver is configured to trust any certificate sent from the server, the issuer information in the certificate is ignored.

Required Permissions for the Java Security Manager

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

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

Permissions for Establishing Connections

To establish a connection to the database server, the 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 MS SQL Server:
grant codeBase "file:WL_HOME${/}server${/}lib${/}wlsqlserver.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 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 2-10.

Granting Access to Temporary Files

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

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

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

You can also grant these permissions to individual files as shown in "Permissions for Establishing Connections" on page 2-10.

Permissions for Kerberos Authentication

To use Kerberos authentication with the Type 4 JDBC drivers that support it, the application and driver code bases must be granted security permissions in the security policy file of the Java 2 Platform as shown in the following examples.

For more information about using Kerberos authentication with the Type 4 JDBC drivers, see the appropriate driver chapters.

Microsoft SQL Server

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

where:

- `WL_HOME` is the directory in which you installed WebLogic Server.
- `your_realm` is the Kerberos realm (or Windows Domain) to which the database host machine belongs.
- `db_hostname` is the host name of the machine running the database.
- `SQLServer_port` is the TCP/IP port on which the Microsoft SQL Server instance is listening.

Unicode Support

Multi-lingual applications can be developed on any operating system platform with JDBC using the 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 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 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 Type 4 JDBC driver has the following format:

```
[BEA][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][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 MS SQL Server Type 4 JDBC Driver

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

SQL Server Database Version Support

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

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

To use JDBC distributed transactions through JTA, you must install stored procedures for Microsoft SQL Server. See "Installing Stored Procedures for JTA" on page 3-47 for details.

Driver Class

The driver class for the Type 4 JDBC MS SQL Server driver is `weblogic.jdbc.sqlserver.SQLServerDriver`. This is a non-XA driver.

Microsoft SQL Server URL

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

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

where:

- *hostname* is the TCP/IP address or TCP/IP host name of the server to which you are connecting. See "Using IP Addresses" on page 2-3 for details on using IP addresses.

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

- *port* is the number of the TCP/IP port.
- *property=value* specifies connection properties. For a list of connection properties and their valid values, see "SQL Server Connection Properties" on page 3-4.

For example:

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

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

Connecting to Named Instances

Microsoft SQL Server and Microsoft SQL Server 2005 support multiple instances of a SQL Server database running concurrently on the same server. An instance is identified by an instance name.

To connect to a named instance using a connection URL, use the following URL format:

```
jdbc:bea:sqlserver://server_name\\instance_name
```

Note: The first back slash character (\) in \\instance_name is an escape character.

where:

server_name is the IP address or hostname of the server.

instance_name is the name of the instance to which you want to connect on the server.

For example, the following connection URL connects to an instance named instance1 on server1:

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

SQL Server Connection Properties

Table 3-1 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 Oracle CEP 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 3-1 SQL Server Connection Properties

Property	Description
AlwaysReportTriggerResults OPTIONAL	{true | false}. Determines how the driver reports results generated by database triggers (procedures that are stored in the database and executed, or fired, when a table is modified). For Microsoft SQL Server 2005, this includes triggers fired by Data Definition Language (DDL) events. If set to true, the driver returns all results, including results generated by triggers. Multiple trigger results are returned one at a time. Use the `Statement.getMoreResults()` method to retrieve individual trigger results. Warnings and errors are reported in the results as they are encountered. If set to false (the default): • For Microsoft SQL Server 2005, the driver does not report trigger results if the statement is a single INSERT, UPDATE, DELETE, CREATE, ALTER, DROP, GRANT, REVOKE, or DENY statement. • For other Microsoft SQL Server databases, the driver does not report trigger results if the statement is a single INSERT, UPDATE, or DELETE statement. In this case, the only result that is returned is the update count generated by the statement that was executed (if errors do not occur). Although trigger results are ignored, any errors generated by the trigger are reported. Any warnings generated by the trigger are enqueued. If errors are reported, the update count is not reported. The default is false.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
AuthenticationMethod	{auto | kerberos | ntlm | userIdPassword}. Determines which authentication method the driver uses when establishing a connection. If set to auto (the default), the driver uses SQL Server authentication, Kerberos authentication, or NTLM authentication when establishing a connection. The driver selects an authentication method based on a combination of criteria, such as whether the application provides a user ID, the driver is running on a Windows platform, and the driver can load the DLL required for NTLM authentication. See "Using the AuthenticationMethod Property" on page 3-33 for more information about using the default value. If set to kerberos, the driver uses Kerberos authentication. The driver ignores any user ID or password specified. This value is supported only when connecting to Microsoft SQL Server 2000 or higher. If set to ntlm, the driver uses NTLM authentication if the DLL required for NTLM authentication can be loaded. If the driver cannot load the DLL, the driver throws an exception. The driver ignores any user ID or password specified. If set to userIdPassword, the driver uses SQL Server authentication when establishing a connection. If a user ID is not specified, the driver throws an exception. The `User` property provides the user ID. The `Password` property provides the password. NOTE: The values type4, type2, and none are deprecated, but are recognized for backward compatibility. We recommend that you use the kerberos, ntlm, and userIdPassword value, respectively, instead. See "Authentication" on page 3-32 for more information about using authentication with the SQL Server driver.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
CodePageOverride OPTIONAL	Specifies the code page the driver uses when converting character data. The specified code page overrides the default database code page. All character data retrieved from or written to the database is converted using the specified code page. The value must be a string containing the name of a valid code page supported by your JVM, for example, `CodePageOverride=CP950`. By default, the driver automatically determines which code page to use to convert Character data. Use this property only if you need to change the driver's default behavior. If a value is set for the `CodePageOverride` property and the `SendStringParametersAsUnicode` property is set to true, the driver ignores the SendStringParametersAsUnicode property and generates a warning. The driver always sends parameters using the code page specified by `CodePageOverride` if this property is specified.
ConnectionRetryCount OPTIONAL	The number of times the driver retries connections to a database server until a successful connection is established. Valid values are 0 and any positive integer. If set to 0, the driver does not try to reconnect after the initial unsuccessful attempt. The `ConnectionRetryDelay` property specifies the wait interval, in seconds, used between attempts. The default is 5.
ConnectionRetryDelay OPTIONAL	The number of seconds the driver waits before retrying connection attempts when ConnectionRetryCount is set to a positive integer. The default is 1.
ConvertNull	{1 | 0}. Controls how data conversions are handled for null values. If set to 1 (the default), the driver checks the data type being requested against the data type of the table column storing the data. If a conversion between the requested type and column type is not defined, the driver generates an "unsupported data conversion" exception regardless of the data type of the column value. If set to 0, the driver does not perform the data type check if the value of the column is null. This allows null values to be returned even though a conversion between the requested type and the column type is undefined. The default is 1.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
DatabaseName OPTIONAL	The name of the database to which you want to connect. See also "Database Connection Property" on page 3-55.
DescribeParameters	{noDescribe | describeIfString}. Controls whether the driver attempts to determine, at execute time, how to send String parameters to the server based on the database data type. Sending String parameters as the type the database expects improves performance and prevents unexpected locking issues caused by data type mismatches. The `SendStringParametersAsUnicode` property controls whether the driver sends String parameter values to the server as Unicode (for example, nvarchar) or non-Unicode (for example, varchar). This property helps applications in which character columns are all Unicode or all non-Unicode. For applications that access both Unicode and non-Unicode columns, a data type mismatch still occurs for some columns if the driver always sends String parameter values to the server in only one format. If set to noDescribe, the driver does not attempt to describe SQL parameters to determine the database data type. The driver sends String parameter values to the server based on the setting of the `SendStringParametersAsUnicode` property. If set to describeIfString, the driver attempts to describe SQL parameters to determine the database data type if one or multiple parameters has been bound as a String (using the PreparedStatement methods setString(), setCharacterStream(), and setAsciiStream()). If the driver can determine the database data type, the driver sends the String parameter data to the server as Unicode if the database type is an n-type (for example, nvarchar). If the database type is not an n-type, the driver converts the data to the character encoding defined by the parameter's collation and sends the data to the server in that character encoding. If the driver cannot determine the data type of the parameters, it sends String parameter values to the server based on the setting of the `SendStringParametersAsUnicode` property. The default is noDescribe.

Table 3-1 SQL Server Connection Properties (Continued)

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

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
EncryptionMethod	{noEncryption | SSL | requestSSL | loginSSL}. Determines whether SSL encryption is used to encrypt data and login requests transmitted over the network between the driver and database server. See "Data Encryption" on page 3-41 for information about choosing between encrypting data, including login requests, and only encrypting login requests. If set to SSL, the login request and data is encrypted using SSL. If the database server does not support SSL, the connection fails and the driver throws an exception. When SSL is enabled, the following properties also apply: HostNameInCertificate TrustStore TrustStorePassword ValidateServerCertificate If set to requestSSL, the login request and data is encrypted using SSL. If the database server does not support SSL, the driver establishes an unencrypted connection. If set to loginSSL, the login request is encrypted using SSL. Data is encrypted using SSL If the database server is configured to require SSL. If the database server does not require SSL, data is not encrypted and only the login request is encrypted. NOTE: If SSL is enabled, the driver communicates with database protocol packets set by the server's default packet size. Any value set by the `PacketSize` property is ignored. See "Data Encryption" on page 3-41 for more information about configuring data encryption. See "Performance Considerations" on page 3-24 for information about configuring this property for optimal performance. The default is noEncryption.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
HostNameInCertificate OPTIONAL	{host_name | #SERVERNAME#}. Specifies a host name for certificate validation when SSL encryption is enabled (`EncryptionMethod=SSL`) and validation is enabled (`ValidateServerCertificate=true`). This property is optional and provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested. If a host name is specified, the driver compares the specified host name to the DNSName value of the SubjectAlternativeName in the certificate. If a DNSName value does not exist, the driver compares the host name with the Common Name (CN) part of the certificate's Subject name. If the values do not match, the connection fails and the driver throws an exception. If #SERVERNAME# is specified, the driver compares the server name specified in the connection URL or data source of the connection to the DNSName value of the SubjectAlternativeName in the certificate. If a DNSName value does not exist, the driver compares the host name to the CN parts of the certificate's Subject name. If the values do not match, the connection fails and the driver throws an exception. NOTE: If multiple CN parts are present, the driver validates the host name against each CN part. If any one validation succeeds, a connection is established. If unspecified, the driver does not validate the host name in the certificate. If SSL encryption or certificate validation is not enabled, any value specified for this property is ignored. See "Data Encryption" on page 3-41 for information about configuring for authentication. The default is an empty string.
HostProcess OPTIONAL	The process ID of the application connecting to Microsoft SQL Server. The value is a string up to a maximum of 128 characters. The value of this property may be useful for database administration purposes. This value is stored in the hostprocess column of the: • sys.sysprocesses table (Microsoft SQL Server 2005) • master.dbo.sysprocesses table (Microsoft SQL Server 2000) Microsoft SQL Server 7 does not store this value. The default is 0.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description

InitializationString	Specifies one or multiple SQL commands to be executed by the driver after it has established the connection to the database and has performed all initialization for the connection. The following connection URL sets the handling of null values to the Microsoft SQL Server default:
	`jdbc:bea:sqlserver://server1:1433;`
	`InitializationString=set ANSI_NULLS off;`
	`DatabaseName=test`
	Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified. The following connection URL sets the handling of null values to the Microsoft SQL Server default and allows delimited identifiers:
	`jdbc:bea:sqlserver://server1:1433;`
	`InitializationString=(set ANSI_NULLS off;`
	`set QUOTED_IDENTIFIER on);DatabaseName=test`
	If the execution of a SQL command fails, the connection attempt also fails and the driver throws an exception indicating which SQL command or commands failed.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
InsensitiveResultSetBufferSize OPTIONAL	{-1 | 0 | x}. Determines the amount of memory used by the driver to cache insensitive result set data. It must have one of the following values: If set to -1, the driver caches all insensitive result set data in memory. If the size of the result set exceeds available memory, an OutOfMemoryException is generated. Because the need to write result set data to disk is eliminated, the driver processes the data more efficiently. If set to 0, the driver caches all insensitive result set data in memory, up to a maximum of 2 GB. If the size of the result set data exceeds available memory, the driver pages the result set data to disk. Because result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk. If set to x, where x is a positive integer, the driver caches all insensitive result set data in memory, using this value to set the size (in KB) of the memory buffer for caching insensitive result set data. If the size of the result set data exceeds the buffer size, the driver pages the result set data to disk. Because the result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk. Specifying a buffer size that is a power of 2 results in more efficient memory use. See “Performance Considerations” on page 3-24 for information about configuring this property for optimal performance. The default is 2048 (KB).
JavaDoubleToString	{true | false}. Determines whether the driver uses its internal conversion algorithm or the JVM conversion algorithm when converting double or float values to string values. If set to true, the driver uses the JVM algorithm when converting double or float values to string values. If set to false (the default), the driver uses its internal algorithm when converting double or float values to string. Setting the property to false improves performance; however, slight rounding differences can occur when compared to the same conversion using the JVM algorithm. These differences are within the allowable error of the double and float data types. The default is false.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description

LoadLibraryPath	Specifies the directory the driver looks in for the DLL used for NTLM authentication. The value is the fully qualified path of the directory that contains the DLL. When you install the driver, the NTLM DLLs are placed in the `WL_HOME/server/lib` subdirectory, where `WL_HOME` is the directory in which you installed WebLogic Server.
	
	By default, the driver looks for the NTLM authentication DLLs in a directory on the Windows system path defined by the PATH environment variable.
	
	If you install the driver in a directory that is not on the Windows system path, you can set this property to specify the location of the NTLM authentication DLLs. For example, if you install the driver in a directory named "DataDirect" that is not on the Windows system path, you can use this property to specify the directory containing the NTLM authentication DLLs.
	
	`jdbc:bea:sqlserver://server3:1433;`
	`DatabaseName=test;LoadLibraryPath=C:\DataDirect\li`
	`b;`
	`User=test;Password=secret`
	
	See "Configuring NTLM Authentication" on page 3-39 for more information about NTLM authentication.

LoginTimeout	The amount of time, in seconds, the driver waits for a connection to be established before returning control to the application and throwing a timeout exception. If set to 0 (the default), the driver does not time out a connection request.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
LongDataCacheSize	{-1 | 0 | x}. Determines whether the driver caches long data (images, pictures, long text, or binary data) in result sets. To improve performance, you can disable long data caching if your application retrieves columns in the order in which they are defined in the result set. If set to -1, the driver does not cache long data in result sets. It is cached on the server. Use this value only if your application retrieves columns in the order in which they are defined in the result set. If set to 0, the driver caches long data in result sets in memory. If the size of the result set data exceeds available memory, the driver pages the result set data to disk. If set to x, where x is a positive integer, the driver caches long data in result sets in memory and uses this value to set the size (in KB) of the memory buffer for caching result set data. If the size of the result set data exceeds available memory, the driver pages the result set data to disk. See "Performance Considerations" on page 3-24 for information about configuring this property for optimal performance. The default is 2048.
NetAddress OPTIONAL	The Media Access Control (MAC) address of the network interface card of the application connecting to Microsoft SQL Server. This value is a string up to a maximum of 12 characters. The value of this property may be useful for database administration purposes. This value is stored in the net_address column of the: • sys.sysprocesses table (Microsoft SQL Server 2005) • master.dbo.sysprocesses table (Microsoft SQL Server 2000) The default is 000000000000.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
PacketSize	PacketSize={-1 | 0 | *x*}. Determines the number of bytes for each database protocol packet transferred from the database server to the client machine (Microsoft SQL Server refers to this packet as a network packet). Adjusting the packet size can improve performance. The optimal value depends on the typical size of data inserted, updated, or returned by the application and the environment in which it is running. Typically, larger packet sizes work better for large amounts of data. For example, if an application regularly returns character values that are 10,000 characters in length, using a value of 32 (16 KB) typically results in improved performance. If set to -1, the driver uses the default maximum packet size used by the database server. If set to 0 (the default), the driver uses a packet size of 64 KB. If set to *x*, an integer from 1 to 128, the driver uses a packet size that is a multiple of 512 bytes. For example, `PacketSize=8` means to set the packet size to 8 * 512 bytes (4096 bytes). See "Performance Considerations" on page 3-24 for information about configuring this property for optimal performance. The default is 0.
Password	A case-insensitive password used to connect to your Microsoft SQL Server database. A password is required only if SQL Server authentication is enabled on your database. If so, contact your system administrator to obtain your password. See "Authentication" on page 3-32 for more information about configuring authentication.
PortNumber OPTIONAL	The TCP port of the primary database server that is listening for connections to the Microsoft SQL Server database. This property is supported only for data source connections. The default is 1433.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
ProgramName OPTIONAL	The name of the application connecting to Microsoft SQL Server. This value is a string up to a maximum of 128 characters. The value of this property may be useful for database administration purposes. This value is stored in the program_name column of the: • sys.sysprocesses table (Microsoft SQL Server 2005) • master.dbo.sysprocesses table (Microsoft SQL Server 2000) Microsoft SQL Server 7 does not store this value The default is an empty string.
QueryTimeout	{*positive integer* | -1 | 0}. Sets the default query timeout (in seconds) for all statements created by a connection. If set to a positive integer, the driver uses the value as the default timeout for any statement created by the connection. To override the default timeout value set by this connection option, call the Statement.setQueryTimeout() method to set a timeout value for a particular statement. If set to -1, the query timeout functionality is disabled. The driver silently ignores calls to the Statement.setQueryTimeout() method. If set to 0 (the default), the default query timeout is infinite (the query does not time out).

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
ReceiveStringParameterType	{NVARCHAR | VARCHAR | DESCRIBE}. Specifies how the driver describes String stored procedure output parameters to the database. If set to NVARCHAR (the default), the driver describes String stored procedure output parameters as nvarchar (4000). Use this value if all output parameters returned by the connection are nchar or nvarchar. If the output parameter is char or varchar, the driver returns the output parameter value, but the returned value is limited to 4000 characters. If set to VARCHAR, the driver describes String stored procedure output parameters as varchar (8000). Use this value if all output parameters returned by the connection are char or varchar. If the output parameter is nchar or nvarchar, data may not be returned correctly. This can occur when the returned data uses a code page other than the database default code page. If set to DESCRIBE, the driver submits a request to the database to describe the parameters of the stored procedure. The driver uses the parameter data types returned by the driver to determine whether to describe the String output parameters as nvarchar or varchar. Use this value if there is a combination of nvarchar and varchar output parameters and if the varchar output parameters can return values that are greater than 4000 characters. This method always works, but it incurs the expense of having to describe the output parameters. The default is NVARCHAR

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
ResultSetMetaDataOptions	{0 | 1}. The SQL Server driver can return table name information in the ResultSet metadata for Select statements if your application requires that information. If set to 0 (the default) and the ResultSetMetaData.getTableName() method is called, the driver does not perform additional processing to determine the correct table name for each column in the result set. In this case, the getTableName() method may return an empty string for each column in the result set. If set to 1 and the ResultSetMetaData.getTableName() method is called, the driver performs additional processing to determine the correct table name for each column in the result set. The driver also can return schema name and catalog name information when the ResultSetMetaData.getSchemaName() and ResultSetMetaData.getCatalogName() methods are called if the driver can determine that information. See “ResultSet MetaData Support” on page 3-52 for more information about returning ResultSet metadata. The default is 0.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
SelectMethod OPTIONAL	{direct | cursor}. A hint to the driver that determines whether the driver requests a database cursor for Select statements. Performance and behavior of the driver are affected by this property, which is defined as a hint because the driver may not always be able to satisfy the requested method. • If set to direct (the default), the database server sends the complete result set in a single response to the driver when responding to a query. A server-side database cursor is not created if the requested result set type is a forward-only result set.Typically, responses are not cached by the driver. Using this method, the driver must process the entire response to a query before another query is submitted. If another query is submitted (using a different statement on the same connection, for example), the driver caches the response to the first query before submitting the second query. Typically, the Direct method performs better than the Cursor method. • If set to cursor, a server-side cursor is requested. When returning forward-only result sets, the rows are retrieved from the server in blocks. The `setFetchSize()` method can be used to control the number of rows that are retrieved for each request when forward-only result sets are returned. Performance tests show that, when returning forward-only result sets, the value of `Statement.setFetchSize()` significantly impacts performance. There is no simple rule for determining the `setFetchSize()` value that you should use. Oracle recommends that you experiment with different setFetchSize() values to determine which value gives the best performance for your application. The cursor method is useful for queries that produce a large amount of data, particularly if multiple open result sets are used. See "Performance Considerations" on page 3-24 for information about configuring this property for optimal performance. The default is Direct.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
SendStringParametersAsUnicode OPTIONAL	{true | false}. Determines whether string parameters are sent to the Microsoft SQL Server database in Unicode or in the default character encoding of the database. If set to true (the default), string parameters are sent to Microsoft SQL Server in Unicode. If set to false, the driver sends string parameters to the database in the default character encoding of the database, which can improve performance because the server does not need to convert Unicode characters to the default encoding. If a value is specified for the CodePageOverride property and this property is set to true, this property is ignored and a warning is generated. See "Performance Considerations" on page 3-24 for information about configuring this property for optimal performance. The default is true.
ServerName REQUIRED	Specifies either the IP address in IPv4 or IPv6 format, or the server name (if your network supports named servers) of the primary database server or named instance. For example, 122.23.15.12 or SQLServerServer. To connect to a named instance, specify *server_name*\\instance_name for this property, where *server_name* is the IP address and *instance_name* is the name of the instance to which you want to connect on the specified server. This property is supported only for data source connections. See "Connecting to Named Instances" on page 3-3 for more information about connecting to named instances.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
SnapshotSerializable	{true | false}. For Microsoft SQL Server 2005 only. Allows your application to use Snapshot Isolation for connections. To configure Snapshot Isolation for connections, you must have your Microsoft SQL Server 2005 database configured for Snapshot Isolation, your application must have the transaction isolation level set to Serializable, and this property must be set to true. If set to false (the default) and your application has the transaction isolation level set to Serializable, the application uses the Serializable isolation level. This property is useful for applications that have the Serializable isolation level set. Using the `SnapshotSerializable` property in this case allows you to use Snapshot Isolation with no or minimum code changes. If you are developing a new application, you may find that using the constant TRANSACTION_SNAPSHOT is a better choice. See "Isolation Levels" on page 3-45 for details. See "Performance Considerations" on page 3-24 for information about configuring this property for optimal performance. The default is false.
TransactionMode	{implicit | explicit}. Controls how the driver delimits the start of a local transaction. If set to implicit, the driver uses implicit transaction mode. This means that Microsoft SQL Server, not the driver, automatically starts a transaction when a transactionable statement is executed. Typically, implicit transaction mode is more efficient than explicit transaction mode because the driver does not have to send commands to start a transaction and a transaction is not started until it is needed. When TRUNCATE TABLE statements are used with implicit transaction mode, Microsoft SQL Server may roll back the transaction if an error occurs. If this occurs, use the explicit value for this property. If set to explicit, the driver uses explicit transaction mode. This means that the driver, not Microsoft SQL Server, starts a new transaction if the previous transaction was committed or rolled back. The default is implicit.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
TrustStore	Specifies the directory of the truststore file to be used when SSL server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts. This value overrides the directory of the truststore file specified by the javax.net.ssl.trustStore Java system property. If this property is not specified, the truststore directory is specified by the javax.net.ssl.trustStore Java system property. This property is ignored if `ValidateServerCertificate=false`.
TrustStorePassword	Specifies the password of the truststore file to be used when SSL server authentication is used. The truststore file contains a list of the Certificate Authorities (CAs) that the client trusts. This value overrides the password of the truststore file specified by the javax.net.ssl.trustStorePassword Java system property. If this property is not specified, the truststore password is specified by the javax.net.ssl.trustStorePassword Java system property. This property is ignored if `ValidateServerCertificate=false`.
User	The case-insensitive user name used to connect to your Microsoft SQL Server database. A user name is required only if SQL Server authentication is enabled on your database. If so, contact your system administrator to obtain your user name.
UseServerSideUpdatableCursors	{true | false}. Determines whether the driver uses server-side cursors when an updatable result set is requested. If set to true, server-side updatable cursors are created when an updatable result set is requested. If set to false, the default updatable result set functionality is used. See "Server-Side Updatable Cursors" on page 3-46 for more information about using server-side updatable cursors. See "Performance Considerations" on page 3-24 for information about configuring this property for optimal performance. The default is false.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
ValidateServerCertificate	{true | false}. Determines whether the driver validates the certificate sent by the database server when SSL encryption is enabled (`EncryptionMethod=SSL`). When using SSL server authentication, any certificate sent by the server must be issued by a trusted Certificate Authority (CA). Allowing the driver to trust any certificate returned from the server even if the issuer is not a trusted CA is useful in test environments because it eliminates the need to specify truststore information on each client in the test environment. If set to false (the default), the driver does not validate the certificate sent by the database server. The driver ignores any truststore information specified by the TrustStore and TrustStorePassword properties or Java system properties. If set to true, the driver validates the certificate sent by the database server. Any certificate from the server must be issued by a trusted CA in the truststore file. if the `HostNameInCertificate` property is specified, the driver also validates the certificate using a host name. The `HostNameInCertificate` property is optional and provides additional security against man-in-the-middle (MITM) attacks by ensuring that the server the driver is connecting to is the server that was requested. Truststore information is specified using the TrustStore and TrustStorePassword properties or by using Java system properties. See "Data Encryption" on page 3-41 for information about configuring for authentication.
WSID OPTIONAL	The workstation ID, which typically is the network name of the computer on which the application resides. The value is a string up to a maximum of 128 characters. The value of this property may be useful for database administration purposes and can be returned by sp_who and the Transact-SQL HOST_NAME function. This value is stored in the hostname column of the: • sys.sysprocesses table (Microsoft SQL Server 2005) • master.dbo.sysprocesses table (Microsoft SQL Server 2000) Microsoft SQL Server 7 does not store this value. The default is an empty string.

Table 3-1 SQL Server Connection Properties (Continued)

Property	Description
XATransactionGroup OPTIONAL	The transaction group ID that identifies any transactions initiated by the connection. This ID can be used for distributed transaction cleanup purposes. You can use the XAResource.recover method to roll back any transactions left in an unprepared state. When you call XAResource.recover, any unprepared transactions that match the ID on the connection used to call XAResource.recover are rolled back. For example, if you specify `XATransactionGroup=ACCT200` and call XAResource.recover on the same connection, any transactions left in an unprepared state identified by the transaction group ID of ACCT200 are rolled back. See "Distributed Transaction Cleanup" on page 3-48 for more information about distributed transaction cleanup.
XMLDescribeType	{longvarchar | longvarbinary}. Determines whether the driver maps XML data to the LONGVARCHAR or LONGVARBINARY data type. If set to longvarchar (the default), the driver maps XML data to the LONGVARCHAR data type. If set to longvarbinary, the driver maps XML data to the LONGVARBINARY data type. See "Returning and Inserting/Updating XML Data" on page 3-29 for more information. The default is longvarchar.

Performance Considerations

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

- "EncryptionMethod" on page 3-25
- "InsensitiveResultSetBufferSize" on page 3-25
- "LongDataCacheSize" on page 3-25
- "PacketSize" on page 3-25
- "ResultSetMetaDataOptions" on page 3-26

- “SelectMethod” on page 3-26
- “SendStringParametersAsUnicode” on page 3-26
- “SnapshotSerializable” on page 3-26
- “UseServerSideUpdatableCursors” on page 3-27

EncryptionMethod

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

InsensitiveResultSetBufferSize

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

LongDataCacheSize

To improve performance when your application retrieves images, pictures, long text, or binary data, you can disable caching for long data on the client if your application retrieves long data column values in the order they are defined in the result set. If your application retrieves long data column values out of order, long data values must be cached on the client. In this case, performance can be improved by increasing the amount of memory used by the driver before writing data to disk.

PacketSize

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

ResultSetMetaDataOptions

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

See "ResultSet MetaData Support" on page 3-52 for more information about returning ResultSet metadata.

SelectMethod

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

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

SendStringParametersAsUnicode

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

SnapshotSerializable

You must have your Microsoft SQL Server 2005 database configured for Snapshot Isolation for this connection property to work. See "Using the Snapshot Isolation Level (Microsoft SQL Server 2005 Only)" on page 3-45 for details.

Snapshot Isolation provides transaction-level read consistency and an optimistic approach to data modifications by not acquiring locks on data until data is to be modified. This Microsoft SQL Server 2005 feature can be useful if you want to consistently return the same result set even if another transaction has changed the data and 1) your application executes many read operations or 2) your application has long running transactions that could potentially block users from reading data. This feature has the potential to eliminate data contention between read operations and update operations. When this connection property is set to true (thereby, you are using Snapshot Isolation), performance is improved due to increased concurrency.

UseServerSideUpdatableCursors

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

See "Server-Side Updatable Cursors" on page 3-46 for more information about using server-side updatable cursors.

Data Types

Table 3-2 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 3-2 Microsoft SQL Server Data Types

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

Table 3-2 Microsoft SQL Server Data Types (Continued)

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

Table 3-2 Microsoft SQL Server Data Types (Continued)

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

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

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

Returning and Inserting/Updating XML Data

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

Returning XML Data

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

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

and the following code:

```
String sql="SELECT xmlCol FROM xmlTable";
```

```
ResultSet rs=stmt.executeQuery(sql);
```

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

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

Character Data

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

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

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

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

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

```
ResultSet.getAsciiStream()
```

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

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

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

Binary Data

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

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

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

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

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

Inserting/Updating XML Data

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

Character Data

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

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

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

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

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

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

Binary Data

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

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

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

Authentication

Authentication protects the identity of the user so that user credentials cannot be intercepted by malicious hackers when transmitted over the network. See “Authentication” on page 2-4 for an overview.

The SQL Server driver supports the following methods of authentication:

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

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

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

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

The `AuthenticationMethod` connection property controls which authentication mechanism the driver uses when establishing connections. See "Using the AuthenticationMethod Property" on page 3-33 for information about setting the value for this property.

Using the AuthenticationMethod Property

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

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

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

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

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

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

Configuring SQL Server Authentication

1. Set the `AuthenticationMethod` property to auto or userIdPassword (the default). See "Using the AuthenticationMethod Property" on page 3-33 for more information about setting a value for this property.
2. Set the `User` property to provide the user ID.
3. Set the `Password` property to provide the password.

Configuring Kerberos Authentication

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

Product Requirements

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

Table 3-3 Kerberos Authentication Requirements for the SQL Server Driver

Component	Requirements
Microsoft SQL Server database server	The database server must be administered by the same domain controller that administers the client and must be running one of the following databases: • Microsoft SQL Server 2005 • Microsoft SQL Server 2000 • Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher
Kerberos server	The Kerberos server is the machine where the user IDs for authentication are administered. The Kerberos server is also the location of the Kerberos KDC. Network authentication must be provided by Windows Active Directory on one of the following operating systems: • Windows Server 2003 • Windows 2000 Server Service Pack 3 or higher
Client	The client must be administered by the same domain controller that administers the database server. In addition, J2SE 1.4.2 or higher must be installed.

Configuring the Driver

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

- krb5.conf is a Kerberos configuration file containing values for the Kerberos realm and the KDC name for that realm. WebLogic Server installs a generic file that you must modify for your environment.
- JDBCDriverLogin.conf file is a configuration file that specifies which Java Authentication and Authorization Service (JAAS) login module to use for Kerberos authentication. This file is configured to load automatically unless the java.security.auth.login.config system property is set to load another configuration file. You can modify this file, but the driver must be able to find the JDBC_DRIVER_01 entry in this file or another specified login

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

To configure the driver:

1. Set the driver's `AuthenticationMethod` property to auto (the default) or kerberos. See "Using the AuthenticationMethod Property" on page 3-33 for more information about setting a value for this property.

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

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

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

   ```
   [libdefaults]
      default_realm = XYZ.COM

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

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

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

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

3. If using Kerberos authentication with a Security Manager on a Java 2 Platform, you must grant security permissions to the application and driver. See "Permissions for Kerberos Authentication" on page 2-12 for an example.

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

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

Specifying User Credentials for Kerberos Authentication (Delegation of Credentials)

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

There may be times when you want the driver to use a set of user credentials other than the operating system user name and password. For example, many application servers or Web servers act on behalf of the client user logged on the machine on which the application is running, rather than the server user.

If you want the driver to use a set of user credentials other than the operating system user name and password, include code in your application to obtain and pass a javax.security.auth.Subject used for authentication as shown in the following example.

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

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

LoginContext lc = null;
Subject subject = null;

try {

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

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

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

    public Object run() {

        Connection con = null;
    try {

         Class.forName("com.ddtek.jdbc.sqlserver.SQLServerDriver");
         String url = "jdbc:bea:sqlserver://myServer:1433";
         con = DriverManager.getConnection(url);
        }
     catch (Exception except) {

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

        return con;
    }
});

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

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

Obtaining a Kerberos Ticket Granting Ticket

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

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

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

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

where user is the application *user*.

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

Configuring NTLM Authentication

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

Product Requirements

Verify that your environment meets the requirements listed in Table 3-4 before you configure your environment for NTLM authentication.

Table 3-4 NTLM Authentication Requirements for the SQL Server Driver

Component	Requirements
Database server	The database server must be administered by the same domain controller that administers the client and must be running on one of the following databases: • Microsoft SQL Server 2005 • Microsoft SQL Server 2000 Service Pack 3 or higher • Microsoft SQL Server 2000 Enterprise Edition (64-bit) Service Pack 2 or higher
Domain controller	The domain controller must administer both the database server and the client. Network authentication must be provided by NTLM on one of the following operating systems: • Windows Server 2003 • Windows 2000 Server Service Pack 3 or higher
Client	The client must be administered by the same domain controller that administers the database server and must be running on one of the following operating systems: • Windows Vista • Windows Server 2003 • Windows XP Service Pack 2 or higher • Windows 2000 Service Pack 4 or higher • Windows NT 4.0 In addition, J2SE 1.3 or higher must be installed.

Configuring the Driver

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

- DDJDBCAuth*xx*.dll (32-bit)
- DDJDBC64Auth*xx*.dll (Itanium 64-bit)
- DDJDBCx64Auth*xx*.dll (AMD64 and Intel EM64T 64-bit)

where *xx* is a two-digit number.

The DLLs are located in the `WL_HOME/server/lib` directory (where `WL_HOME` is the directory in which you installed WebLogic Server). If the application using NTLM authentication is running in a 32-bit JVM, the driver automatically uses DDJDBCAuthxx.dll. Similarly, if the application is running in a 64-bit JVM, the driver uses DDJDBC64Authxx.dll or DDJDBCx64Authxx.dll.

To configure the driver:

1. Set the `AuthenticationMethod` property to auto (the default) or ntlm. See "Using the AuthenticationMethod Property" on page 3-33 for more information about setting a value for this property.
2. By default, the driver looks for the NTLM authentication DLLs in a directory on the Windows system path defined by the PATH environment variable. If you install the driver in a directory that is not on the Windows system path, perform one of the following actions to ensure the driver can load the DLLs:
   - Add the `WL_HOME/server/lib` directory to the Windows system path, where `WL_HOME` is the directory in which you installed WebLogic Server.
   - Copy the NTLM authentication DLLs from `WL_HOME/server/lib` to a directory that is on the Windows system path, where `WL_HOME` is the directory in which you installed WebLogic Server.
   - Set the `LoadLibraryPath` property to specify the location of the NTLM authentication DLLs. For example, if you install the driver in a directory named "DataDirect" that is not on the Windows system path, you can use the `LoadLibraryPath` property to specify the directory containing the NTLM authentication DLLs:

   ```
   jdbc:bea:sqlserver://server3:1521;
   DatabaseName=test;LoadLibraryPath=C:\DataDirect\lib;User=test;Password=
   secret
   ```
3. If using NTLM authentication with a Security Manager on a Java 2 Platform, security permissions must be granted to allow the driver to establish connections. See "Permissions for Establishing Connections" on page 2-10 for an example.

Data Encryption

The SQL Server driver supports SSL for data encryption. SSL secures the integrity of your data by encrypting information and providing authentication. See "Data Encryption Across the Network" on page 2-7 for an overview.

Depending on your Microsoft SQL Server configuration, you can choose to encrypt all data, including the login request, or encrypt the login request only. Encrypting login requests, but not data, is useful for the following scenarios:

- When your application needs security, but cannot afford to pay the performance penalty for encrypting data transferred between the driver and server.
- Microsoft SQL Server 2005 only. When the server is not configured for SSL, but your application still requires a minimum degree of security.

Note: When SSL is enabled, the driver communicates with database protocol packets set by the server's default packet size. Any value set by the `PacketSize` property is ignored.

Using SSL with Microsoft SQL Server

If your Microsoft SQL Server database server has been configured with an SSL certificate signed by a trusted CA, the server can be configured so that SSL encryption is either optional or required. When required, connections from clients that do support SSL encryption fail.

Although a signed trusted SSL certificate is recommended for the best degree of security, Microsoft SQL Server 2005 can provide limited security protection even if an SSL certificate has not been configured on the server. If a trusted certificate is not installed, the server will use a self-signed certificate to encrypt the login request, but not the data.

Table 3-5 shows how the different `EncryptionMethod` property values behave with different Microsoft SQL Server configurations.

Table 3-5 EncryptionMethod Property and Microsoft SQL Server Configurations

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

Table 3-5 EncryptionMethod Property and Microsoft SQL Server Configurations (Continued)

Value	No SSL Certificate	SSL Certificate	
		SSL Optional	SSL Required
requestSSL	Login request and data are not encrypted	Login request and data are encrypted	Login request and data are encrypted.
loginSSL	Microsoft SQL Server 2005: Login request is encrypted, but data is not encrypted Microsoft SQL Server 2000: Connection attempt fails.	Login request is encrypted, but data is not encrypted.	Login request and data are encrypted.

Configuring SSL Encryption

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

DML with Results (Microsoft SQL Server 2005)

The SQL Server driver supports the Microsoft SQL Server 2005 Output clause for Insert, Update, and Delete statements. For example, suppose you created a table with the following statement:

```
CREATE TABLE table1(id int, name varchar(30))
```

The following Update statement updates the values in the id column of table1 and returns a result set that includes the old ID (replaced by the new ID), the new ID, and the name associated with these IDs:

```
UPDATE table1 SET id=id*10 OUTPUT deleted.id as oldId, inserted.id as newId,
inserted.name
```

The driver returns the results of Insert, Update, or Delete statements and the update count in separate result sets. The output result set is returned first, followed by the update count for the Insert, Update, or Delete statement. To execute DML with Results statements in an application, use the Statement.execute() or PreparedStatement.execute() method. Then, use Statement.getMoreResults () to obtain the output result set and the update count. For example:

```
String sql = "UPDATE table1 SET id=id*10 OUTPUT deleted.id as oldId,
   inserted.id as newId, inserted.name";
boolean isResultSet = stmt.execute(sql);

int   updateCount = 0;
while (true) {

   if (isResultSet) {
         resultSet = stmt.getResultSet();
         while (resultSet.next()) {

            System.out.println("oldId: " + resultSet.getInt(1) +
                               "newId: " + resultSet.getInt(2) +
                               "name: " + resultSet.getString(3));
         }
         resultSet.close();
   }
   else {
         updateCount = stmt.getUpdateCount();
         if (updateCount == -1) {
            break;
         }

         System.out.println("Update Count: " + updateCount);
   }
```

```
      isResultSet = stmt.getMoreResults();
   }
```

SQL Escape Sequences

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

Isolation Levels

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

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

* Supported for Microsoft SQL Server 2005 only.

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

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

You can use the Snapshot isolation level in either of the following ways:

- Setting the `SnapshotSerializable` property changes the behavior of the Serializable isolation level to use the Snapshot isolation level. This allows an application to use the Snapshot isolation level with no or minimum code changes. See the description of this property in Table 3-1 for more information.
- Importing the ExtConstants class allows you to specify the TRANSACTION_SNAPSHOT or TRANSACTION_SERIALIZABLE isolation levels for an individual statement in the same application. The ExtConstants class in the com.ddtek.jdbc.extensions package defines

the TRANSACTION_SNAPSHOT constant. For example, the following code imports the ExtConstants class and sets the TRANSACTION_SNAPSHOT isolation level:

```
    import com.ddtek.jdbc.extensions.ExtConstants;

Connection.setTransactionIsolation(ExtConstants.TRANSACTION_SNAPSHOT);
```

Using Scrollable Cursors

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

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

Server-Side Updatable Cursors

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

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

When server-side updatable cursors are used with sensitive result sets that were generated from a database table that contains a primary key, the following changes you make to the result set are visible:

- Own Inserts are visible. Others Inserts are not visible.
- Own and Others Updates are visible.
- Own and Others Deletes are visible.

Using the default behavior of the driver (`UseServerSideUpdatableCursors=false`), those changes would not be visible.

Installing Stored Procedures for JTA

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

To install stored procedures for JTA:

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

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

2. From the database server, use the ISQL utility to run the `instjdbc.sql` script. As a precaution, have your system administrator back up the master database before running `instjdbc.sql`. At a command prompt, use the following syntax to run `instjdbc.sql`:

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

   where:

   sa_password is the password of the system administrator.

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

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

   The `instjdbc.sql` script generates many messages. In general, these messages can be ignored; however, the system administrator should scan the output for any messages that may indicate an execution error. The last message should indicate that `instjdbc.sql` ran successfully. The script fails when there is insufficient space available in the master database to store the JDBC XA procedures or to log changes to existing procedures.

Distributed Transaction Cleanup

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

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

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

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

Transaction Timeout

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

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

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

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

Explicit Transaction Cleanup

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

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

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

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

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

Large Object (LOB) Support

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

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

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

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

Batch Inserts and Updates

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

Parameter Metadata Support

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

Insert and Update Statements

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

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

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

Select Statements

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

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

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

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

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

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

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

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

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

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

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

Joins are supported. For example:

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

Fully qualified names and aliases are supported. For example:

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

Stored Procedures

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

ResultSet MetaData Support

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

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

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

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

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

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

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

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

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

Rowset Support

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

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

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

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

Auto-Generated Keys Support

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

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

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

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

Null Values

When the Microsoft SQL Server driver establishes a connection, the driver sets the Microsoft SQL Server database option ansi_nulls to on. This action ensures that the driver is compliant with the ANSI SQL standard, which makes developing cross-database applications easier.

By default, Microsoft SQL Server does not evaluate null values in SQL equality (=) or inequality (<>) comparisons or aggregate functions in an ANSI SQL-compliant manner. For example, the ANSI SQL specification defines that `col1=null` as shown in the following Select statement always evaluates to false:

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

Using the default database setting (ansi_nulls=off), the same comparison evaluates to true instead of false.

Setting ansi_nulls to on changes how the database handles null values and forces the use of `IS NULL` instead of `=NULL`. For example, if the value of col1 in the following Select statement is null, the comparison evaluates to true:

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

In your application, you can restore the default Microsoft SQL Server behavior for a connection in the following ways:

- Use the `InitializationString` property to specify the SQL command `set ANSI_NULLS off`. For example, the following URL ensures that the handling of null values is restored to the Microsoft SQL Server default for the current connection:

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

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

  ```
  SET ANSI_NULLS OFF
  ```

Database Connection Property

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

If both the Database and DatabaseName connection properties are specified in a connection URL, the last of either property positioned in the connection URL is used. For example, if your application specifies the following connection URL, the value of the Database connection property would be used instead of the value of the DatabaseName connection property.

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

JDBC Support

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

JDBC Compatibility

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

Table A-1 JDBC Compatibility

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

Supported Functionality

The following tables list functionality supported for each JDBC object.

Array Object

Table A-2 Array Object

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

Blob Object

Table A-3 Blob Object

Blob Object Methods	Version Introduced	Supported	Comments
InputStream getBinaryStream ()	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the JDBCLONGVARBINARY data type.
byte[] getBytes (long, int)	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARBINARY data type.
long length ()	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARBINARY data type.
long position (Blob, long)	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARBINARY data type.
long position (byte[], long)	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARBINARY data type.
OutputStream setBinaryStream (long)	3.0	Yes	The SQL Server driver supports using with data types that map to the LONGVARBINARY data type.
int setBytes (long, byte[])	3.0	Yes	The SQL Server driver supports using with data types that map to the LONGVARBINARY data type.
int setBytes (long, byte[], int, int)	3.0	Yes	The SQL Server driver supports using with data types that map to the LONGVARBINARY data type.

Table A-3 Blob Object (Continued)

Blob Object Methods	Version Introduced	Supported	Comments
void truncate (long)	3.0	Yes	The SQL Server driver supports using with data types that map to the LONGVARBINARY data type.

CallableStatement Object

Table A-4 CallableStatement Object

CallableStatement Object Methods	Version Introduced	Supported	Comments
Array getArray (int)	2.0 Core	No	Throws “unsupported method” exception.
Array getArray (String)	3.0	No	Throws “unsupported method” exception.
BigDecimal getBigDecimal (int)	2.0 Core	Yes	
BigDecimal getBigDecimal (int, int)	1.0	Yes	
BigDecimal getBigDecimal (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Blob getBlob (int)	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the JDBC LONGVARBINARY data type.
Blob getBlob (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
boolean getBoolean (int)	1.0	Yes	

Table A-4 CallableStatement Object (Continued)

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

Table A-4 CallableStatement Object (Continued)

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
double getDouble (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
float getFloat (int)	1.0	Yes	
float getFloat (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
int getInt (int)	1.0	Yes	
int getInt (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
long getLong (int)	1.0	Yes	
long getLong (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Object getObject (int)	1.0	Yes	
Object getObject (int, Map)	2.0 Core	Yes	Map ignored.
Object getObject (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Object getObject (String, Map)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception. Map ignored.
Ref getRef (int)	2.0 Core	No	Throws “unsupported method” exception.
Ref getRef (String)	3.0	No	Throws “unsupported method” exception.

Table A-4 CallableStatement Object (Continued)

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
short getShort (int)	1.0	Yes	
short getShort (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
String getString (int)	1.0	Yes	
String getString (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Time getTime (int)	1.0	Yes	
Time getTime (int, Calendar)	2.0 Core	Yes	
Time getTime (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Time getTime (String, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Timestamp getTimestamp (int)	1.0	Yes	
Timestamp getTimestamp (int, Calendar)	2.0 Core	Yes	
Timestamp getTimestamp (String)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
Timestamp getTimestamp (String, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
URL getURL (int)	3.0	No	Throws “unsupported method” exception.

Table A-4 CallableStatement Object (Continued)

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

Table A-4 CallableStatement Object (Continued)

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
void setBoolean (String, boolean)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setByte (String, byte)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setBytes (String, byte [])	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setCharacterStream (String, Reader, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setDate (String, Date)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setDate (String, Date, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setDouble (String, double)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setFloat (String, float)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setInt (String, int)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setLong (String, long)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw “unsupported method” exception.

Table A-4 CallableStatement Object (Continued)

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

Table A-4 CallableStatement Object (Continued)

CallableStatement Object (Continued) Methods	Version Introduced	Supported	Comments
void setTimestamp (String, Timestamp, Calendar)	3.0	Yes	Supported for the SQL Server driver only. All other drivers throw "unsupported method" exception.
void setURL (String, URL)	3.0	No	Throws "unsupported method" exception.
boolean wasNull ()	1.0	Yes	

Clob Object

Table A-5 Clob Object

Clob Object Methods	Version Introduced	Supported	Comments
InputStream getAsciiStream ()	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARCHAR data type.
Reader getCharacterStream ()	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARCHAR data type.
String getSubString (long, int)	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARCHAR data type.
long length ()	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARCHAR data type.
long position (Clob, long)	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARCHAR data type.

Table A-5 Clob Object (Continued)

Clob Object (Continued) Methods	Version Introduced	Supported	Comments
long position (String, long)	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARCHAR data type.
OutputStream setAsciiStream (long)	3.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARCHAR data type.
Writer setCharacterStream (long)	3.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARCHAR data type.
int setString (long, String)	3.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARCHAR data type.
int setString (long, String, int, int)	3.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARCHAR data type.
void truncate (long)	3.0 Core	Yes	The SQL Server driver supports using with data types that map to the LONGVARCHAR data type.

Connection Object

Table A-6 Connection Object

Connection Object Methods	Version Introduced	Supported	Comments
void clearWarnings ()	1.0	Yes	
void close ()	1.0	Yes	When a connection is closed while a transaction is still active, that transaction is rolled back.

Table A-6 Connection Object (Continued)

Connection Object Methods	Version Introduced	Supported	Comments
void commit ()	1.0	Yes	
Statement createStatement ()	1.0	Yes	
Statement createStatement (int, int)	2.0 Core	Yes	
Statement createStatement (int, int, int)	3.0	No	Throws "unsupported method" exception.
boolean getAutoCommit ()	1.0	Yes	
String getCatalog ()	1.0	Yes	
String getClientInfo ()	4.0	Yes	
String getClientInfo (String)	4.0	Yes	
int getHoldability ()	3.0	Yes	
DatabaseMetaData getMetaData ()	1.0	Yes	
int getTransactionIsolation ()	1.0	Yes	
Map getTypeMap ()	2.0 Core	Yes	Always returns empty java.util.HashMap.
SQLWarning getWarnings ()	1.0	Yes	
boolean isClosed ()	1.0	Yes	
boolean isReadOnly ()	1.0	Yes	
boolean isValid ()	4.0	Yes	
String nativeSQL (String)	1.0	Yes	Always returns same String as passed in.

Table A-6 Connection Object (Continued)

Connection Object Methods	Version Introduced	Supported	Comments
CallableStatement prepareCall (String)	1.0	Yes	
CallableStatement prepareCall (String, int, int)	2.0 Core	Yes	
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	
PreparedStatement prepareStatement (String, int, int, int)	3.0	No	Throws "unsupported method" exception.
PreparedStatement prepareStatement (String, int[])	3.0	Yes	
PreparedStatement prepareStatement (String, String [])	3.0	Yes	
void releaseSavepoint (Savepoint)	3.0	Yes	
void rollback ()	1.0	Yes	
void rollback (Savepoint)	3.0	Yes	
void setAutoCommit (boolean)	1.0	Yes	

Table A-6 Connection Object (Continued)

Connection Object Methods	Version Introduced	Supported	Comments
void setCatalog (String)	1.0	Yes	
String setClientInfo (Properties)	4.0	Yes	
String setClientInfo (String, String)	4.0	Yes	
void setHoldability (int)	3.0	Yes	Holdability parameter value is ignored.
void setReadOnly (boolean)	1.0	Yes	
Savepoint setSavepoint ()	3.0	Yes	
Savepoint setSavepoint (String)	3.0	Yes	
void setTransactionIsolation (int)	1.0	Yes	
void setTypeMap (Map)	2.0 Core	Yes	Ignored.

DatabaseMetaData Object

Table A-7 DababaseMetaData Object

DatabaseMetaData Object Methods	Version Introduced	Supported	Comments
boolean allProceduresAreCallable ()	1.0	Yes	
boolean allTablesAreSelectable ()	1.0	Yes	
boolean dataDefinitionCausesTransaction Commit ()	1.0	Yes	
boolean dataDefinitionIgnoredInTransactions ()	1.0	Yes	
boolean deletesAreDetected (int)	2.0 Core	Yes	

Table A-7 DababaseMetaData Object (Continued)

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
boolean doesMaxRowSizeIncludeBlobs ()	1.0	Yes	Not supported by the SQL Server driver.
getAttributes (String, String, String, String)	3.0	Yes	Empty result set is returned.
ResultSet getAttributes (String, String, String, String)	3.0	No	Throws “unsupported method” exception.
ResultSet getBestRowIdentifier (String, String, String, int, boolean)	1.0	Yes	
ResultSet getCatalogs ()	1.0	Yes	
String getCatalogSeparator ()	1.0	Yes	
String getCatalogTerm ()	1.0	Yes	
String getClientInfoProperties ()	4.0	Yes	
ResultSet getColumnPrivileges (String, String, String, String)	1.0	Yes	
ResultSet getColumns (String, String, String, String)	1.0	Yes	
Connection getConnection ()	2.0 Core	Yes	
ResultSet getCrossReference (String, String, String, String, String, String)	1.0	Yes	
int getDatabaseMajorVersion ()	3.0	Yes	
int getDatabaseMinorVersion ()	3.0	Yes	
String getDatabaseProductName ()	1.0	Yes	
String getDatabaseProductVersion ()	1.0	Yes	
int getDefaultTransactionIsolation ()	1.0	Yes	
int getDriverMajorVersion ()	1.0	Yes	
int getDriverMinorVersion ()	1.0	Yes	

Table A-7 DababaseMetaData Object (Continued)

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

Table A-7 DabarbaseMetaData Object (Continued)

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
int getMaxProcedureNameLength ()	1.0	Yes	
int getMaxRowSize ()	1.0	Yes	
int getMaxSchemaNameLength ()	1.0	Yes	
int getMaxStatementLength ()	1.0	Yes	
int getMaxStatements ()	1.0	Yes	
int getMaxTableNameLength ()	1.0	Yes	
int getMaxTablesInSelect ()	1.0	Yes	
int getMaxUserNameLength ()	1.0	Yes	
String getNumericFunctions ()	1.0	Yes	
ResultSet getPrimaryKeys (String, String, String)	1.0	Yes	
ResultSet getProcedureColumns (String, String, String, String)	1.0	Yes	
ResultSet getProcedures (String, String, String)	1.0	Yes	
String getProcedureTerm ()	1.0	Yes	
int getResultSetHoldability ()	3.0	Yes	
ResultSet getSchemas ()	1.0	Yes	
String getSchemaTerm ()	1.0	Yes	
String getSearchStringEscape ()	1.0	Yes	
String getSQLKeywords ()	1.0	Yes	
int getSQLStateType ()	3.0	Yes	
String getStringFunctions ()	1.0	Yes	
ResultSet getSuperTables (String, String, String)	3.0	Yes	Empty result set is returned.

Table A-7 DababaseMetaData Object (Continued)

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

Table A-7 DababaseMetaData Object (Continued)

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
boolean othersInsertsAreVisible (int)	2.0 Core	Yes	
boolean othersUpdatesAreVisible (int)	2.0 Core	Yes	
boolean ownDeletesAreVisible (int)	2.0 Core	Yes	
boolean ownInsertsAreVisible (int)	2.0 Core	Yes	
boolean ownUpdatesAreVisible (int)	2.0 Core	Yes	
boolean storesLowerCaseIdentifiers ()	1.0	Yes	
boolean storesLowerCaseQuoted Identifiers ()	1.0	Yes	
boolean storesMixedCaseIdentifiers ()	1.0	Yes	
boolean storesMixedCaseQuoted Identifiers ()	1.0	Yes	
boolean storesUpperCaseIdentifiers ()	1.0	Yes	
boolean storesUpperCaseQuoted Identifiers ()	1.0	Yes	
boolean supportsAlterTableWith AddColumn ()	1.0	Yes	
boolean supportsAlterTableWith DropColumn ()	1.0	Yes	
boolean supportsANSI92EntryLevelSQL ()	1.0	Yes	
boolean supportsANSI92FullSQL ()	1.0	Yes	
boolean supportsANSI92Intermediate SQL ()	1.0	Yes	
boolean supportsBatchUpdates ()	2.0 Core	Yes	
boolean supportsCatalogsInData Manipulation ()	1.0	Yes	

Table A-7 DababaseMetaData Object (Continued)

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

Table A-7 DababaseMetaData Object (Continued)

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

Table A-7 DabababseMetaData Object (Continued)

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

Table A-7 DababaseMetaData Object (Continued)

DatabaseMetaData Object (Continued) Methods	Version Introduced	Supported	Comments
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	
int getParameterCount ()	3.0	Yes	
int getParameterMode (int)	3.0	Yes	
int getParameterType (int)	3.0	Yes	
String getParameterTypeName (int)	3.0	Yes	
int getPrecision (int)	3.0	Yes	
int getScale (int)	3.0	Yes	
int isNullable (int)	3.0	Yes	
boolean isSigned (int)	3.0	Yes	
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	

Table A-10 PreparedStatement Object (Continued)

PreparedStatement Object Methods	Version Introduced	Supported	Comments
void clearParameters ()	1.0	Yes	
boolean execute ()	1.0	Yes	
ResultSet executeQuery ()	1.0	Yes	
int executeUpdate ()	1.0	Yes	
ResultSetMetaData getMetaData ()	2.0 Core	Yes	
ParameterMetaData getParameterMetaData ()	3.0	Yes	
void setArray (int, Array)	2.0 Core	No	Throws "unsupported method" exception.
void setAsciiStream (int, InputStream, int)	1.0	Yes	
void setBigDecimal (int, BigDecimal)	1.0	Yes	
void setBinaryStream (int, InputStream, int)	1.0	Yes	
void setBlob (int, Blob)	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the JDBC LONGVARBINARY data type.
void setBoolean (int, boolean)	1.0	Yes	
void setByte (int, byte)	1.0	Yes	
void setBytes (int, byte [])	1.0	Yes	

Table A-10 PreparedStatement Object (Continued)

PreparedStatement Object Methods	Version Introduced	Supported	Comments
void setCharacterStream (int, Reader, int)	2.0 Core	Yes	
void setClob (int, Clob)	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the JDBC LONGVARCHAR data type.
void setDate (int, Date)	1.0	Yes	
void setDate (int, Date, Calendar)	2.0 Core	Yes	
void setDouble (int, double)	1.0	Yes	
void setFloat (int, float)	1.0	Yes	
void setInt (int, int)	1.0	Yes	
void setLong (int, long)	1.0	Yes	
void setNull (int, int)	1.0	Yes	
void setNull (int, int, String)	2.0 Core	Yes	
void setObject (int, Object)	1.0	Yes	
void setObject (int, Object, int)	1.0	Yes	
void setObject (int, Object, int, int)	1.0	Yes	

Table A-10 PreparedStatement Object (Continued)

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

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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	
InputStream getBinaryStream (int)	1.0	Yes	
InputStream getBinaryStream (String)	1.0	Yes	
Blob getBlob (int)	2.0 Core	Yes	
Blob getBlob (String)	2.0 Core	Yes	
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	
byte [] getBytes (String)	1.0	Yes	
Reader getCharacterStream (int)	2.0 Core	Yes	
Reader getCharacterStream (String)	2.0 Core	Yes	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
Clob getClob (int)	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the JDBC LONGVARCHAR data type.
Clob getClob (String)	2.0 Core	Yes	The SQL Server driver supports using with data types that map to the JDBC LONGVARCHAR data type.
int getConcurrency ()	2.0 Core	Yes	
String getCursorName ()	1.0	No	Throws "unsupported method" exception.
Date getDate (int)	1.0	Yes	
Date getDate (int, Calendar)	2.0 Core	Yes	
Date getDate (String)	1.0	Yes	
Date getDate (String, Calendar)	2.0 Core	Yes	
double getDouble (int)	1.0	Yes	
double getDouble (String)	1.0	Yes	
int getFetchDirection ()	2.0 Core	Yes	
int getFetchSize ()	2.0 Core	Yes	
float getFloat (int)	1.0	Yes	
float getFloat (String)	1.0	Yes	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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	
Object getObject (int, Map)	2.0 Core	Yes	
Object getObject (String)	1.0	Yes	
Object getObject (String, Map)	2.0 Core	Yes	Map ignored.
Ref getRef (int)	2.0 Core	No	Throws "unsupported method" exception.
Ref getRef (String)	2.0 Core	No	Throws "unsupported method" exception.
int getRow ()	2.0 Core	Yes	
short getShort (int)	1.0	Yes	
short getShort (String)	1.0	Yes	
Statement getStatement ()	2.0 Core	Yes	
String getString (int)	1.0	Yes	
String getString (String)	1.0	Yes	

Table A-12 ResultSet Object

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

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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	
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	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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	
void updateBlob (int, Blob)	3.0	Yes	The SQL Server driver supports using with data types that map to the JDBC LONGVARBINARY data type.
void updateBlob (String, Blob)	3.0	Yes	The SQL Server driver supports using with data types that map to the JDBC LONGVARBINARY data type.

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
void 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	
void updateClob (int, Clob)	3.0	Yes	The SQL Server driver supports using with data types that map to the JDBC LONGVARCHAR data type
void updateClob (String, Clob)	3.0	Yes	The SQL Server driver supports using with data types that map to the JDBC LONGVARCHAR data type
void updateDate (int, Date)	2.0 Core	Yes	
void updateDate (String, Date)	2.0 Core	Yes	
void updateDouble (int, double)	2.0 Core	Yes	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
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	
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	

Table A-12 ResultSet Object

ResultSet Object Methods	Version Introduced	Supported	Comments
void updateShort (String, short)	2.0 Core	Yes	
void updateString (int, String)	2.0 Core	Yes	
void updateString (String, String)	2.0 Core	Yes	
void updateTime (int, Time)	2.0 Core	Yes	
void updateTime (String, Time)	2.0 Core	Yes	
void updateTimestamp (int, Timestamp)	2.0 Core	Yes	
void updateTimestamp (String, Timestamp)	2.0 Core	Yes	
boolean wasNull ()	1.0	Yes	

ResultSetMetaData Object

Table A-13 ResultSetMetaData Object

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

Table A-13 ResultSetMetaData Object (Continued)

ResultSetMetaData Object (Continued)	Version Introduced	Supported	Comments
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 SQL Server Type 4 driver. • To return a table name for the SQL Server Type 4 driver, add the following property to the connection pool Properties field: `ResultsetMetaDataOptions=1`
boolean isAutoIncrement (int)	1.0	Yes	
boolean isCaseSensitive (int)	1.0	Yes	
boolean isCurrency (int)	1.0	Yes	
boolean isDefinitelyWritable (int)	1.0	Yes	
int isNullable (int)	1.0	Yes	
boolean isReadOnly (int)	1.0	Yes	
boolean isSearchable (int)	1.0	Yes	
boolean isSigned (int)	1.0	Yes	
boolean isWritable (int)	1.0	Yes	

SavePoint Object

Table A-14 SavePoint Object

SavePoint Object Methods	Version Introduced	Supported	Comments
(all)	3.0	Yes	

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.
void cancel ()	1.0	Yes	The SQL Server driver cancels the execution of the statement. If the statement is cancelled by the database server, the driver throws an exception indicating that it was cancelled.
void clearBatch ()	2.0 Core	Yes	
void clearWarnings ()	1.0	Yes	
void close ()	1.0	Yes	
boolean execute (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.
boolean execute (String, int)	3.0	Yes	
boolean execute (String, int [])	3.0	Yes	

Table A-15 Statement Object (Continued)

Statement Object Methods	Version Introduced	Supported	Comments
boolean execute (String, String [])	3.0	Yes	
int [] executeBatch ()	2.0 Core	Yes	
ResultSet executeQuery (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.
int executeUpdate (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.
int executeUpdate (String, int)	3.0	Yes	
int executeUpdate (String, int [])	3.0	Yes	
int executeUpdate (String, String [])	3.0	Yes	
Connection getConnection ()	2.0 Core	Yes	
int getFetchDirection ()	2.0 Core	Yes	
int getFetchSize ()	2.0 Core	Yes	
ResultSet getGeneratedKeys ()	3.0	Yes	The SQL server driver returns 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.
int getMaxFieldSize ()	1.0	Yes	
int getMaxRows ()	1.0	Yes	
boolean getMoreResults ()	1.0	Yes	

Table A-15 Statement Object (Continued)

Statement Object Methods	Version Introduced	Supported	Comments
boolean getMoreResults (int)	3.0	Yes	
int getQueryTimeout ()	1.0	Yes	The SQL Server driver returns 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.
void setFetchDirection (int)	2.0 Core	Yes	
void setFetchSize (int)	2.0 Core	Yes	
void setMaxFieldSize (int)	1.0	Yes	
void setMaxRows (int)	1.0	Yes	

Table A-15 Statement Object (Continued)

Statement Object Methods	Version Introduced	Supported	Comments
void setQueryTimeout (int)	1.0	Yes	The SQL Server 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	

GetTypeInfo

The following tables provide results returned from the `DataBaseMetaData.getTypeInfo` method for all of the 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.

- "SQL Server Driver" on page B-1

SQL Server Driver

Table B-1 provides getTypeInfo results for for all Microsoft SQL Server databases supported by the SQL Server driver. See Chapter 3, "The MS SQL Server Type 4 JDBC Driver."

Table B-1 getTypeInfo for SQL Server

TYPE_NAME = bigint *	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bigint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
* Supported only for Microsoft SQL Server 2000 and higher.	

Table B-1 getTypeInfo for SQL Server (Continued)

TYPE_NAME = bigint identity *	
AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = bigint identity	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
* Supported only for Microsoft SQL Server 2000 and higher.	
TYPE_NAME = binary	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (BINARY)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = binary	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
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	

Table B-1 getTypeInfo for SQL Server (Continued)

TYPE_NAME = char	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = char	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = datetime	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 3
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 23
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 3	
TYPE_NAME = decimal	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *precision,scale*	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION =
FIXED_PREC_SCALE = false	28 (SQL Server 7) *,
LITERAL_PREFIX = NULL	38 (SQL Server 2000 and SQL Server 2005) *
LITERAL_SUFFIX = NULL	SEARCHABLE = 2
LOCAL_TYPE_NAME = decimal	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE =	SQL_DATETIME_SUB = NULL
28 (SQL Server 7), *	UNSIGNED_ATTRIBUTE = false
38 (SQL Server 2000 and SQL Server 2005) *	
* Configurable server option for Microsoft SQL Server 2000 and higher.	

Table B-1 getTypeInfo for SQL Server (Continued)

TYPE_NAME = decimal() identity	

AUTO_INCREMENT = true
CASE_SENSITIVE = false
CREATE_PARAMS = *precision*
DATA_TYPE = 3 (DECIMAL)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = decimal() identity
MAXIMUM_SCALE = 0
	MINIMUM_SCALE = 0
	NULLABLE = 0
	NUM_PREC_RADIX = 10
	PRECISION =
	28 (SQL Server 7),
	38 (SQL Server 2000 and SQL Server 2005)
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

TYPE_NAME = float	

AUTO_INCREMENT = false
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 6 (FLOAT)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = float
MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = 2
	PRECISION = 53
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = false

TYPE_NAME = image	

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

Table B-1 getTypeInfo for SQL Server (Continued)

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	
TYPE_NAME = int identity	
AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 4 (INTEGER)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = int identity	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
TYPE_NAME = money	
AUTO_INCREMENT = false	MINIMUM_SCALE = 4
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 19
FIXED_PREC_SCALE = true	SEARCHABLE = 2
LITERAL_PREFIX = $	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = money	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 4	

Table B-1 getTypeInfo for SQL Server (Continued)

TYPE_NAME = nchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *length*	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = ntext	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 1073741823
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = ntext	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = numeric	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = *precision,scale*	NUM_PREC_RADIX = 10
DATA_TYPE = 2 (NUMERIC)	PRECISION =
FIXED_PREC_SCALE = false	28 (SQL Server 7),*
LITERAL_PREFIX = NULL	38 (SQL Server 2000 and SQL Server 2005) *
LITERAL_SUFFIX = NULL	SEARCHABLE = 2
LOCAL_TYPE_NAME = numeric	SQL_DATA_TYPE = NULL
MAXIMUM_SCALE =	SQL_DATETIME_SUB = NULL
28 (SQL Server 7),*	UNSIGNED_ATTRIBUTE = false
38 (SQL Server 2000 and SQL Server 2005) *	

* Configurable server option for Microsoft SQL Server 2000 and higher.

Table B-1 getTypeInfo for SQL Server (Continued)

TYPE_NAME = numeric() identity	
AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = *precision*	NUM_PREC_RADIX = 10
DATA_TYPE = 2 (NUMERIC)	PRECISION = 28 (SQL Server 7.0), 38 (SQL Server 2000 and SQL Server 2005)
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = numeric() identity	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	
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 = nvarchar(max) *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 1073741823
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nvarchar(max)	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
* Supported only for Microsoft SQL Server 2005.	

Table B-1 getTypeInfo for SQL Server (Continued)

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	
TYPE_NAME = smallint	
AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 5 (SMALLINT)	PRECISION = 5
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallint	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

Table B-1 getTypeInfo for SQL Server (Continued)

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	
TYPE_NAME = sql_variant *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 12 (VARCHAR)	PRECISION = 8000
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = sql_variant	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	
* Supported only for Microsoft SQL Server 2000 and higher.	

Table B-1 getTypeInfo for SQL Server (Continued)

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-1 getTypeInfo for SQL Server (Continued)

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

Table B-1 getTypeInfo for SQL Server (Continued)

TYPE_NAME = varbinary	

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = *max length*
DATA_TYPE = -3 (VARBINARY)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = 0x
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = varbinary
MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 8000
	SEARCHABLE = 2
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = varbinary(max) *	

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

* Supported only for Microsoft SQL Server 2005.	
TYPE_NAME = varchar	

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = max length
DATA_TYPE = 12 (VARCHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = ’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = varchar
MAXIMUM_SCALE = NULL
	MINIMUM_SCALE = NULL
	NULLABLE = 1
	NUM_PREC_RADIX = NULL
	PRECISION = 8000
	SEARCHABLE = 3
	SQL_DATA_TYPE = NULL
	SQL_DATETIME_SUB = NULL
	UNSIGNED_ATTRIBUTE = NULL

Table B-1 getTypeInfo for SQL Server (Continued)

TYPE_NAME = varchar(max) *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar(max)	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
* Supported only for Microsoft SQL Server 2005.	
TYPE_NAME = xml *	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 1073741823
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = xml	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
* Supported only for Microsoft SQL Server 2005.	

SQL Escape Sequences for JDBC

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

- Date, time, and timestamp literals
- Scalar functions such as numeric, string, and data type conversion functions
- Outer joins
- Escape characters for wildcards used in LIKE clauses
- Procedure calls

The escape sequence used by JDBC is:

```
{extension}
```

The escape sequence is recognized and parsed by the 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 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
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		

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 Type 4 JDBC drivers for each data store.

Table C-3 Outer Join Escape Sequences Supported

Data Store	Outer Join Escape Sequences
SQL Server	Left outer joins Right outer joins Full outer joins Nested outer joins

LIKE Escape Character Sequence for Wildcards

You can specify the character to be used to escape wildcard characters (% and _, for example) in LIKE clauses. The escape sequence for escape characters is:

```
{escape 'escape-character'}
```

where *`escape-character`* is the character used to escape the wildcard character.

For example. the following SQL statement specifies that an asterisk (*) be used as the escape character in the LIKE clause for the wildcard character %:

```
SELECT col1 FROM table1 WHERE col1 LIKE '*%%' {escape '*'}
```

Procedure Call Escape Sequences

A procedure is an executable object stored in the data store. Generally, it is one or more SQL statements that have been precompiled. The escape sequence for calling a procedure is:

```
{[?=]call procedure-name[([parameter][,parameter]...)]}
```

where:

procedure-name specifies the name of a stored procedure.

parameter specifies a stored procedure parameter.