DataDirect TECHNOLOGIES

DataDirect Connect® for JDBC®

User's Guide and Reference

January 2003

DataDirect Technologies
3202 Tower Oaks Blvd. Suite 300
Rockville, Maryland 20852

Table of Contents

Preface

This book is your guide to DataDirect Connect® *for* JDBC® from DataDirect Technologies. The DataDirect Connect *for* JDBC product consists of a number of Type 4 JDBC drivers that are compliant with the JDBC specification. It also includes the following tools to test and enhance connectivity: DataDirect Test™ *for* JDBC (DataDirect Test), DataDirect Spy™ *for* JDBC (DataDirect Spy), and the DataDirect Connection Pool Manager.

Using This Book

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

This book contains the following information:

- Chapter 1 "Quick Start Connect" on page 15 provides information about connecting with and testing your DataDirect Connect *for* JDBC drivers.
- Chapter 2 "Using DataDirect Connect for JDBC Drivers" on page 21 provides information about using JDBC applications with the DataDirect Connect *for* JDBC drivers.
- Subsequent chapters for each database driver provide detailed information specific to the driver.
- Appendix A "JDBC Support" on page 87 provides information about developing JDBC applications for DataDirect Connect *for* JDBC environments.

- Appendix B "GetTypeInfo" on page 133 provides results returned from the method DataBaseMetaData.getTypeinfo for all of the DataDirect Connect *for* JDBC drivers.
- Appendix C "Designing JDBC Applications for Performance Optimization" on page 175 provides information about enhancing the performance of your application by optimizing its code.
- Appendix D "SQL Escape Sequences for JDBC" on page 191 describes the scalar functions supported for the DataDirect Connect *for* JDBC drivers. Your data store may not support all of these functions.
- Appendix E "Using DataDirect Test" on page 201 introduces DataDirect Test, a development component that allows you to test and learn the JDBC API, and contains a tutorial that takes you through a working example of its use.
- Appendix F "Tracking JDBC Calls with DataDirect Spy" on page 255 introduces DataDirect Spy, a development component that allows you to track JDBC calls, and describes how to use it.
- Appendix G "Connection Pool Manager" on page 263 describes how to use the DataDirect Connection Pool Manager.

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

Typographical Conventions

This book uses the following typographical conventions:

Convention	Explanation
italics	Introduces new terms that you may not be familiar with, and is used occasionally for emphasis.
bold	Emphasizes important information. Also indicates button, menu, and icon names on which you can act. For example, click **Next**.
UPPERCASE	Indicates the name of a file. For operating environments that use case-sensitive file names, the correct capitalization is used in information specific to those environments. Also indicates keys or key combinations that you can use. For example, press the ENTER key.
`monospace`	Indicates syntax examples, values that you specify, or results that you receive.
`monospaced italics`	Indicates names that are placeholders for values you specify; for example, *`filename`*.
forward slash /	Separates menus and their associated commands. For example, Select File / Copy means to select Copy from the File menu.
vertical rule |	Indicates an OR separator to delineate items.
brackets [ ]	Indicates optional items. For example, in the following statement: SELECT [DISTINCT], DISTINCT is an optional keyword.
braces { }	Indicates that you must select one item. For example, {yes | no} means you must specify either yes or no.
ellipsis . . .	Indicates that the immediately preceding item can be repeated any number of times in succession. An ellipsis following a closing bracket indicates that all information in that unit can be repeated.

Contacting Technical Support

DataDirect Technologies provides technical support for registered users of this product, including limited installation support, for the first 30 days. Register online for your SupportLink user ID and password for access to the password-protected areas of the SupportLink web site at http://www.datadirect-technologies.com/support/support_index.asp. Your user ID and password are issued to you by email upon registration.

For post-installation support, contact us using one of the methods listed below or purchase further support by enrolling in the SupportLink program. For more information about SupportLink, contact your sales representative.

The DataDirect Technologies web site provides the latest support information through SupportLink Online, our global service network providing access to support contact details, tools, and valuable information. Our SupportLink users access information using the web and automatic email notification. SupportLink Online includes a knowledge base so you can search on keywords for technical bulletins and other information.

World Wide Web

http://www.datadirect-technologies.com/support/support_index.asp

E-Mail

USA, Canada, and Mexico	supportlink@datadirect-technologies.com
Europe, Middle East, and Africa	int.supportlink@datadirect-technologies.com
Japan	jpn.answerline@datadirect.co.jp
All other countries	http://www.datadirect-technologies.com/contactus/distributor.asp provides a list of the correct e-mail contacts.

Local Telephone Support

Local phone numbers can be found at:

http://www.datadirect-technologies.com/support/support_contact_aline.asp

SupportLink support is available 24 hours a day, seven days a week.

Fax Information

Fax US, Mexico, and Canada	1 919 461 4527
Fax EMEA	+32 (0) 15 32 09 19

When you contact us, please provide the following information:

- The **product serial number** or a case number. If you do not have a SupportLink contract, we will ask you to speak with a sales representative.
- Your **name and organization.** For a first-time call, you may be asked for full customer information, including location and contact details.
- The **version number** of your DataDirect product.
- The type and version of your **operating system**.
- Any **third-party software or other environment information** required to understand the problem.
- A **brief description of the problem,** including any error messages you have received, **and the steps preceding the occurrence of the problem.** Depending on the complexity of the problem, you may be asked to submit an example so that we can recreate the problem.
- An assessment of the **severity level** of the problem.

1 Quick Start Connect

This chapter provides basic information for connecting with and testing your DataDirect Connect *for* JDBC drivers immediately after installation. To take full advantage of the features of DataDirect Connect *for* JDBC, we recommend that you read Chapter 2 "Using DataDirect Connect for JDBC Drivers" and the chapters specific to the drivers you are using.

Connecting to a Database

Once the DataDirect Connect *for* JDBC drivers are installed, you can connect from your application to your database in either of the following ways: with a connection URL through the JDBC Driver Manager or with a JNDI data source. This quick start explains how to test your database connection using a connection URL. See Chapter 2 "Using DataDirect Connect for JDBC Drivers" for details on using data sources.

You can connect through the JDBC Driver Manager with the DriverManager.getConnection method. This method uses a string containing a URL. Use the following steps to load the DataDirect Connect *for* JDBC driver from your JDBC application.

1. Setting the Classpath

The DataDirect Connect *for* JDBC drivers must be defined in your CLASSPATH variable. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate the JDBC drivers on your computer. If the drivers are not defined on your CLASSPATH, you will receive the error "class not found" when trying to load the driver. Set your system CLASSPATH to include the following entries, where `driver.jar` is the driver jar file (for example, sqlserver.jar) and `install_dir` is the path to your DataDirect Connect *for* JDBC installation directory:

```
install_dir/lib/base.jar
install_dir/lib/util.jar
install_dir/lib/driver.jar
```

Windows Example

```
CLASSPATH=.;c:\connectjdbc\lib\base.jar;c:\connectjdbc\lib\
util.jar;c:\connectjdbc\lib\sqlserver.jar
```

UNIX Example

```
CLASSPATH=.;/home/user1/connectjdbc/lib/base.jar;/home/
user1/connectjdbc/lib/util.jar;/home/user1/connectjdbc/lib/
sqlserver.jar
```

2. Registering the Driver

Registering the driver tells the JDBC Driver Manager which driver to load. When loading a driver using class.forName(), you must specify the name of the driver. The names of each DataDirect Connect *for* JDBC driver are:

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

For example:

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

3. Passing the Connection URL

After registering the driver, you must pass your database connection information in the form of a connection URL. See the following URL formats for each DataDirect Connect *for* JDBC driver. Use them as templates to create your own connection URLs, substituting the appropriate values specific to your database.

DB2 UDB[1]

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

DB2 OS/390 and iSeries[1]

```
jdbc:datadirect:db2://server_name:50000;Location=db2_location;
CollectionId=your_collectionname;PackageName=your_packagename
```

Informix

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

Oracle

```
jdbc:datadirect:oracle://server_name:1521
```

SQL Server [2]

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

Sybase

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

[1] See the section "Creating a DB2 Package" on page 44 in the DB2 driver chapter before configuring your initial connection.

[2] For instructions on connecting to named instances, see "Connecting to Named Instances" on page 73 in the SQL Server driver chapter.

For example, to specify a connection URL for SQL Server that includes the user ID and password:

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

NOTES:

- The `server_name` is an IP address or a host name, assuming that your network resolves host names to IP addresses. You can test this by using the ping command to access the host name and verifying that you receive a reply with the correct IP address.
- The numeric value after the server name is the port number on which the database is listening. The values listed here are sample defaults. You should determine the port number that your database is using and substitute that value.

You can find the complete list of Connection URL parameters in the section "Connection String Properties" in each driver chapter of this book.

Testing the Connection

DataDirect Test is a pure Java software component developed by DataDirect Technologies for JDBC testing. DataDirect Test is a graphical, menu-driven program that allows developers to test JDBC applications. It can also be used as a tool to help developers learn the JDBC API. Its menu items either correspond to specific JDBC functions (for example, connecting to a database or passing a SQL statement) or encapsulate multiple JDBC function calls as a shortcut to perform common tasks (for example, returning a record result). DataDirect Test displays the results of all JDBC function calls and provides fully-commented sample Java JDBC code in the same window.

See the tutorial in Appendix E "Using DataDirect Test" on page 201 for complete information about testing your connection using DataDirect Test.

2 Using DataDirect Connect *for* JDBC Drivers

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

About the Product

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

- DataDirect Test
- DataDirect Spy
- DataDirect Connection Pool Manager
- J2EE Connector Architecture resource adapters

The Drivers

The DataDirect Connect *for* JDBC drivers are compliant with the JDBC 3.0 specification. The DataDirect Connect *for* JDBC drivers also support the following JDBC 2.0 Standard Extension Features:

- Data Sources
- Connection Pooling
- Distributed Transactions

DataDirect Test

DataDirect Test is a menu-driven software component that allows you to test the JDBC API and learn how to use the DataDirect Connect *for* JDBC drivers. DataDirect Test contains menu items that:

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

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

See Appendix E "Using DataDirect Test" on page 201 for more information about DataDirect Test.

DataDirect Spy

DataDirect Spy is a software component for tracking JDBC calls. It passes calls issued by an application to an underlying JDBC driver and logs detailed information about the calls. DataDirect Spy provides the following advantages:

- Logs all JDBC calls (3.0), including support for the JDBC 2.0 Optional Package.
- Logging is consistent, regardless of the JDBC driver used.
- All parameters and function results for JDBC calls can be logged.
- Logging works with all DataDirect Connect *for* JDBC drivers.
- Logging can be enabled without changing the application by using the appropriate DataDirect Connect *for* JDBC data source object.

See Appendix F "Tracking JDBC Calls with DataDirect Spy" on page 255 for more information about DataDirect Spy.

DataDirect Connection Pool Manager

Database access performance can be improved significantly when connection pooling is used. Connection pooling means that connections are reused rather than created each time a connection is requested. Your application can use connection pooling through the DataDirect Connection Pool Manager.

See Appendix G "Connection Pool Manager" on page 263 for more information about the DataDirect Connection Pool Manager.

J2EE Connector Architecture Resource Adapters

The J2EE Connector architecture defines a standard structure for connecting the J2EE platform to Enterprise Information Systems (EISs). Examples of EISs include mainframe transaction processing, database systems, and legacy applications not written in the Java programming language. The J2EE Connector Architecture enables the integration of EISs with application servers and enterprise applications.

The J2EE Connector Architecture defines a standard set of system-level contracts between an application server and EISs to ensure compatibility between them. The resource adapter implements the EIS portion of these system-level contracts.

A resource adapter is a system-level software driver used by an application server to connect to an EIS. The resource adapter communicates with the server to provide the underlying transaction, security, and connection pooling mechanisms.

The J2EE Connector Architecture also defines a standard Service Provider Interface (SPI) for integrating the transaction, security and connection management facilities of an application server with those of a transactional resource manager. The JDBC 3.0 specification describes the relationship of JDBC to the SPI specified in the J2EE Connector Architecture.

DataDirect Connect *for* JDBC supports appropriate JDBC functionality through the J2EE Connector Architecture SPI by providing resource adapters. The DataDirect resource adapters are provided in resource archive (RAR) files, and are named like the DataDirect Connect *for* JDBC driver files, for example, oracle.rar. See the Installed File list in the DataDirect Connect *for* JDBC readme file for the names and locations of the RAR files. See the *DataDirect Connect for JDBC Installation Guide* for information about creating the resource adapters.

Using Resource Adapters with an Application Server

In an application server environment, resource adapters are deployed using a deployment tool. Each RAR file includes a deployment descriptor, which instructs the application server on how to use the resource adapter in an application server environment. The deployment descriptor contains information about the resource adapter, including security and transactional capabilities, and the ManagedConnectionFactory class name. See your application server documentation for details about how to deploy components using the deployment tool.

Using Resource Adapters from an Application

The J2EE Connector Architecture resource adapters may also be used directly from an application, rather than through a container-managed, application server environment. The following code example shows how you might access an Oracle database using the resource adapter.

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

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

public class RAExample {

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

          // Create a connection factory instance
      OracleManagedConnectionFactory managedFactory =
          new OracleManagedConnectionFactory();
      managedFactory.setServerName("MyOracleServer");
      managedFactory.setPortNumber("1521");
       managedFactory.setSID("TEST");

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

       // Get an InitialContext. Using File System JNDI Service
       // Provider as an example
       Hashtable   env = new Hashtable();

       env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.fscontext.RefFSContextFactory");
          env.put(Context.PROVIDER_URL,
"file:c:/ConnectionFactories");

       Context connectorContext = new InitialContext(env);

       // Bind the connection factory
       try {
            connectorContext.bind("OracleConnectionFactory",
factory);
       }
       catch (NameAlreadyBoundException except) {
            connectorContext.rebind("OracleConnectionFactory",
                      factory);
         }
     }
     catch (Exception except) {
         System.out.println("Error creating DataSource");
      System.exit(0);
     }

     // Connect via the DataSource

     try {

         // Get an InitialContext. Using File System JNDI Service
      // Provider as an example
      Hashtable   env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY,
  "com.sun.jndi.fscontext.RefFSContextFactory");
      env.put(Context.PROVIDER_URL,
  "file:c:/ConnectionFactories");

      Context connectorContext = new InitialContext(env);

      // Lookup the connection factory
      DataSource dataSource = (DataSource)
    connectorContext.lookup("OracleConnectionFactory");
      Connection connection =
          dataSource.getConnection("scott", "tiger");
     }
     catch (Exception except) {
         System.out.println("Looking up connection factory");
     }
    }
}
```

Connecting Through the JDBC Driver Manager

One way of connecting to a database is through the JDBC Driver Manager, using the DriverManager.getConnection method. This method uses a string containing a URL. The following code shows an example of using the JDBC Driver Manager to connect to Microsoft SQL Server while passing the user name and password:

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

URL Examples

The complete connection URL format used with the Driver Manager is:

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

where:

drivername	is the name of the DataDirect Connect *for* JDBC driver, for example, `sybase`.
hostname	is the TCP/IP address or TCP/IP host name of the server to which you are connecting. 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 the appropriate DataDirect Connect *for* JDBC driver chapter.

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

DB2[1]

```
jdbc:datadirect:db2://server1:50000;DatabaseName=jdbc;PackageName=pkg1;
User=test;Password=secret
```

DB2 OS/390 and iSeries[1]

```
jdbc:datadirect:db2://server_name:50000;Location=Sample;
CollectionId=default;PackageName=pkg1
```

Informix

```
jdbc:datadirect:informix://server4:1526;informixServer=ol_test;
DatabaseName=jdbc
```

Oracle

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

SQL Server [2]

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

Sybase

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

[1] See the section “Creating a DB2 Package” on page 44 in the DB2 driver chapter before configuring your initial connection.

[2] For instructions on connecting to named instances, see “Connecting to Named Instances” on page 73 in the SQL Server driver chapter.

Connecting Through Data Sources

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

How Data Sources Are Implemented

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

Each DataDirect Connect *for* JDBC data source class provided implements the following interfaces defined in the JDBC 2.0 Optional Package:

- javax.sql.DataSource
- javax.sql.ConnectionPoolDataSource, which enables you to implement connection pooling
- javax.sql.XADataSource and javax.sql.XADataConnection, which enable you to implement distributed transactions through JTA

Creating Data Sources

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

- JNDI_LDAP_Example.java. Use this example to create a JDBC data source and save it in your LDAP directory, using the JNDI Provider for LDAP.
- JNDI_FILESYSTEM_Example.java. Use this example to create a JDBC data source and save it in your local file system, using the File System JNDI Provider.

You can use these examples as templates for creating a DataDirect Connect *for* JDBC data source that meets your needs.

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

Calling a Data Source in an Application

Applications can call a DataDirect Connect *for* JDBC data source using a logical name to retrieve the javax.sql.DataSource object. This object loads the DataDirect Connect *for* JDBC driver and can be used to establish a connection to the underlying database.

Once a DataDirect Connect *for* JDBC data source has been registered with JNDI, it can be used by your JDBC application as shown in the following example:

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

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

For instructions on creating JDBC data sources, see "Creating Data Sources" on page 31.

Using Connection Pooling

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

Example A: Without Connection Pooling

1. The client application creates a connection.
2. The client application sends a data access query.
3. The client application obtains the result set of the query.
4. The client application displays the result set to the end user.
5. The client application ends the connection.

Example B: With Connection Pooling

1. The client checks the connection pool for an unused connection.
2. If an unused connection exists, it is returned by the pool implementation; otherwise, it creates a new connection.

3 The client application sends a data access query.

4 The client application obtains the result set of the query.

5 The client application displays the result set to the end user.

6 The client application returns the connection to the pool.

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

The pool implementation creates "real" database connections using the getPooledConnection() method of ConnectionPoolDataSource. Then, the pool implementation registers itself as a listener to the PooledConnection. When a client application requests a connection, the pool implementation (DataDirect Connection Pool Manager) assigns one of its available connections. If there is no connection available, the DataDirect Connection Pool Manager establishes a new connection and assigns it to that application. When the client application closes the connection, the DataDirect Connection Pool Manager is notified by the driver through the ConnectionEventListener interface that the connection is free and available for reuse. The pool implementation is also notified by the ConnectionEventListener interface when the client somehow corrupts the database connection, so that the pool implementation can remove that connection from the pool.

Connection pool implementations like the DataDirect Connection Pool Manager use objects that implement the javax.sql.ConnectionPooledDataSource interface to create the pooled connections managed in the pool. All the Connect *for* JDBC DataSource objects implement the ConnectionPooledDataSource interface. Once a DataDirect Connect *for* JDBC data source has been created and registered with JNDI, it can be used by your JDBC application as shown in the following example, typically through a third-party connection pool tool:

```
Context ctx = new InitialContext();
ConnectionPoolDataSource ds =
(ConnectionPoolDataSource)ctx.lookup("jdbc/EmployeeDB");
pooledConnection pcon = ds.getPooledConnection("scott", "tiger");
```

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

The DataDirect Connection Pool Manager is part of DataDirect Connect *for* JDBC. See Appendix G "Connection Pool Manager" on page 263 for details.

Specifying Connection Properties

You can specify connection properties using the JDBC Driver Manager or DataDirect Connect *for* JDBC data sources. See "URL Examples" on page 28 for information about specifying properties through the Driver Manager. See "Creating Data Sources" on page 31 for information about example data sources that specify connection properties.

For the list of the connection properties specific to each DataDirect Connect *for* JDBC driver, see the appropriate driver chapter.

Testing Connections

See Appendix E "Using DataDirect Test" on page 201 for instructions on testing JDBC driver connections using DataDirect Test.

Using the Drivers on a Java 2 Platform

When using the DataDirect Connect *for* JDBC drivers on a Java 2 Platform with the standard security manager enabled, you must grant the driver some additional permissions. Refer to your Java 2 Platform documentation for more information about Java 2 Platform security model and permissions.

You can run an application on a Java 2 Platform with the standard security manager using:

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

where *application_class_name* is the class name of the application.

Web browser applets running in the Java 2 plug-in are always running in a Java Virtual Machine with the standard security manager enabled. To enable the necessary permissions, you must add them to the security policy file of the Java 2 Platform. This security policy file can be found in the jre/lib/security subdirectory of the Java 2 Platform installation directory.

To use JDBC data sources, all code bases must have the following permissions:

```
// permissions granted to all domains
grant {
// DataSource access
permission java.util.PropertyPermission "java.naming.*", "read,write";
// Adjust the server host specification for your environment
permission java.net.SocketPermission "*.datadirect-technologies.com:0-65535",
"connect";
};
```

To use insensitive scrollable cursors, and perform client-side sorting of some DatabaseMetaData ResultSets, all code bases must have access to temporary files.

For JDK 1.1 environments, access to "current working directory" must be granted.

For Java 2 Platform environments, access to the temporary directory specified by the VM configuration must be granted.

The following shows an example of permissions being granted for the C:\TEMP directory:

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

Unicode Support

Multi-lingual applications can be developed on any operating system platform with JDBC using the DataDirect Connect *for* JDBC drivers to access both Unicode and non-Unicode enabled databases. Internally, Java applications use UTF-16 Unicode encoding for string data. When fetching data, the DataDirect Connect *for* 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 DataDirect Connect *for* 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 the DataDirect Connect *for* JDBC drivers has the following format:

`[DataDirect][`*`Connect for JDBC Driver`*`]`*`message`*

For example:

```
[DataDirect][Sybase JDBC Driver]Timeout expired.
```

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

Database Errors

An error generated by the database has the following format:

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

For example:

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

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

3 The DB2 Driver

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

- DB2 Universal Database (UDB) 7.1, 7.2, and 8.1 running on Windows NT, Windows 2000, UNIX, Linux, and Linux/s390 via DRDA
- DB2 6.1 and DB2 UDB 7.1 running on OS/390 and z/OS via DRDA
- DB2 UDB V4R5, V5R1, and V5R2 running on iSeries and AS/400

NOTE: IBM currently uses the term "UDB" in connection with DB2 mainframe platforms. This documentation uses the following terms as described:

- "DB2 UDB" refers to all versions of DB2 running on Windows, UNIX, and Linux/s390 platforms
- "DB2 OS/390" refers to all versions of DB2 on OS/390 and z/OS platforms
- "DB2 iSeries" refers to all versions of DB2 on iSeries and AS/400

Data Source and Driver Classes

The data source class for the DB2 driver is:

com.ddtek.jdbcx.db2.DB2DataSource

See "Connecting Through Data Sources" on page 30 for information on DataDirect Connect *for* JDBC data sources.

The driver class for the DB2 driver is:

com.ddtek.jdbc.db2.DB2Driver

J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the DB2 resource adapter is:

com.ddtek.resource.spi.DB2ManagedConnectionFactory

See "J2EE Connector Architecture Resource Adapters" on page 24 for information about using DataDirect Connect *for* JDBC drivers as J2EE Connector Architecture resource adapters.

Connection String Properties

You can use the following connection properties with the JDBC Driver Manager or DataDirect Connect *for* JDBC data sources.

Table 3-1 lists the JDBC connection properties supported by the DB2 driver, and describes each property. The properties have the form:

property=value

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

Table 3-1. DB2 Connection String Properties

Property	Description
AddToCreateTable OPTIONAL	A string that is automatically added to all Create Table statements. This field is primarily for users who need to add an "in database" clause.
AlternateID OPTIONAL	Sets the default DB2 schema used by unqualified SQL identifiers to the specified value. The value must be a valid DB2 schema.
CollectionId OPTIONAL	The collection (group of packages) to which the package is bound. This property is ignored for DB2 UDB. The default is DEFAULT.
CreateDefaultPackage OPTIONAL	{true | false}. Determines whether the package name defined with the PackageName property should be created. For DB2 OS/390 and DB2 iSeries, the package is created in the collection specified by the CollectionId property. This would be used if the package does not yet exist. For more information about creating DB2 packages, see "Creating a DB2 Package" on page 44. The default is false.

Table 3-1. DB2 Connection String Properties (cont.)

Property	Description
DynamicSections OPTIONAL	Specifies the number of statements that the DB2 driver package can prepare for a single user. The default is 100.
DatabaseName	The name of the database to which you want to connect (used with UDB).
ForceFixRow OPTIONAL	{true | false}. Forces the driver to fetch rows using the Fix Row Protocol, even if the package specifies Limited Block Protocol. The default is false.
LocationName	The name of the DB2 location that you want to access (used with OS/390 and iSeries).
MaxPooledStatements OPTIONAL	The maximum number of pooled PreparedStatements for this connection. The default is 0.
PackageName	Specifies the name (7-character limit) of the package that the driver uses to process static and dynamic SQL. For more information about creating DB2 packages, see "Creating a DB2 Package" on page 44. The default package name is DDTEK.
Password	A case-sensitive password used to connect to your DB2 database. A password is required only if security is enabled on your database. If so, contact your system administrator to get your password.
PortNumber OPTIONAL	The TCP port (use for DataSource connections only). The default is 50000.

***Table 3-1. DB2 Connection String Properties** (cont.)*

Property	**Description**
ReplacePackage OPTIONAL	{true | false}. Specifies whether the current bind process should replace an existing PackageName. On DB2 UDB, this property must be used in conjunction with CreateDefaultPackage. For more information about creating DB2 packages, see "Creating a DB2 Package" on page 44. The default is false.
ServerName	The IP address (use for DataSource connections only).
StripNewlines OPTIONAL	{true | false}. Specifies whether new-line characters in a SQL statement are sent to the DB2 server. When StripNewlines=true, the DB2 driver removes all new-line characters from SQL statements. The default is true.
User	The case-sensitive user name used to connect to your DB2 database. A user name is required only if security is enabled on your database. If so, contact your system administrator to get your user name.
WithHoldCursors OPTIONAL	{true | false}. Determines whether the cursor stays open on commit and rollback—either DB2 closes all open cursors (Delete cursors) after a commit or rollback, or leaves them open (Preserve cursors). When set to true, the cursor behavior is Preserve. When set to false, the cursor behavior is Delete. The default is true.

Creating a DB2 Package

A DB2 package must exist on the DB2 server before you can use the DB2 driver.

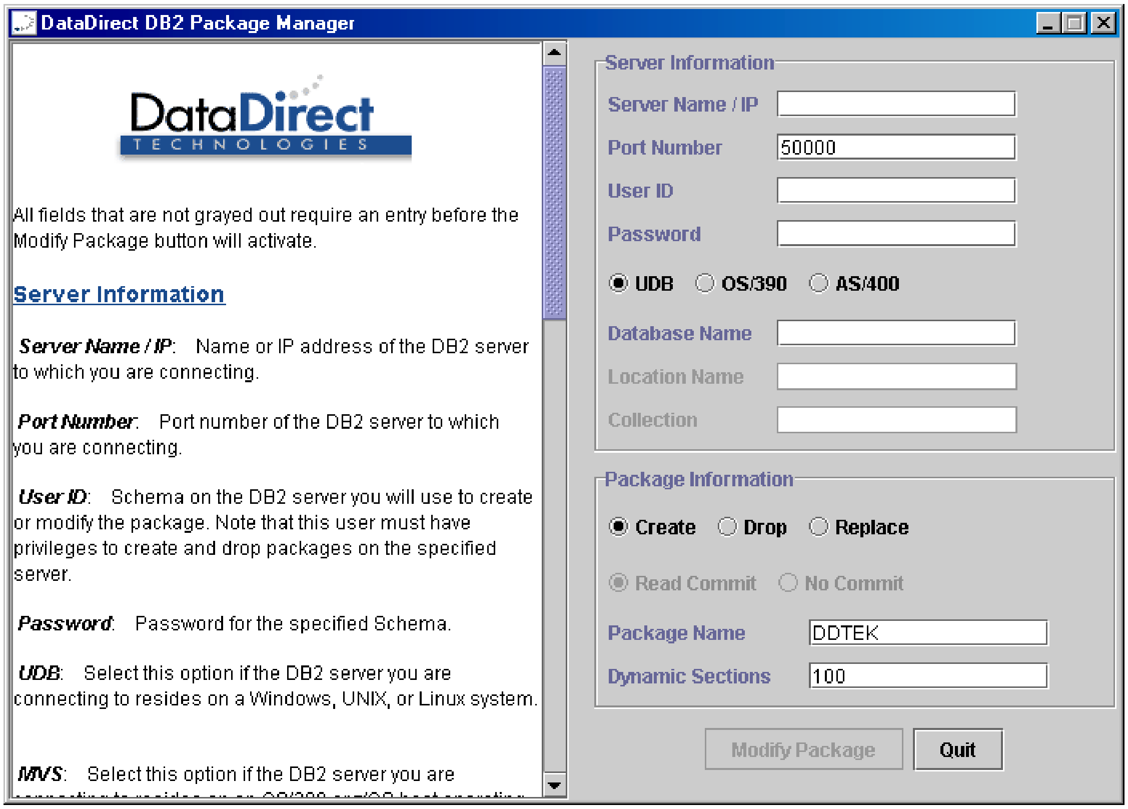
NOTE: If you have already established a CollectionId or a PackageName with a previous version of the DataDirect Connect DB2 *for* JDBC driver, this version of the DataDirect Connect DB2 *for* JDBC driver still can use that same CollectionId or PackageName. If you have not established either one, you must create them before you set up your initial connection.

You can create a DB2 package in either of the following ways:

- Using the DataDirect DB2 Package Manager, a graphical tool written in Java that makes it easy for you to create, drop, and replace DB2 packages for DB2 UDB, DB2 OS/390, and DB2 iSeries
- Specifying specific connection properties in the initial connection URL

Using the DataDirect DB2 Package Manager

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


DataDirect DB2 Package Manager

All fields that are not grayed out require an entry before the Modify Package button will activate.

Server Information

Server Name / IP: Name or IP address of the DB2 server to which you are connecting.

Port Number: Port number of the DB2 server to which you are connecting.

User ID: Schema on the DB2 server you will use to create or modify the package. Note that this user must have privileges to create and drop packages on the specified server.

Password: Password for the specified Schema.

UDB: Select this option if the DB2 server you are connecting to resides on a Windows, UNIX, or Linux system.

MVS: Select this option if the DB2 server you are

Server Information

Server Name / IP	
Port Number	50000
User ID	
Password	

[● UDB] ○ OS/390 ○ AS/400

Database Name	
Location Name	
Collection	

Package Information

[● Create] ○ Drop ○ Replace

[● Read Commit] ○ No Commit

Package Name	DDTEK
Dynamic Sections	100

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

Using Connection Properties

Table 3-2 lists the connection properties you should use in your initial connection URL when you create a DB2 package:

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

Property	Database
DatabaseName=*database_name*	DB2 UDB only
LocationName=*location_name*	DB2 OS/390 and iSeries only
CollectionId=*collection_name*	DB2 OS/390 and iSeries only
PackageName=*package_name*	DB2 UDB, OS/390, and iSeries
CreateDefaultPackage=TRUE	DB2 UDB, OS/390, and iSeries
ReplacePackage=TRUE	DB2 UDB only

Using CreateDefaultPackage=TRUE creates a package of the name that you specify. You can then use the package for your other connections. If you use CreateDefaultPackage=TRUE, and you do not specify a CollectionId, the DEFAULT CollectionId is created.

NOTE: On DB2 UDB, you must use ReplacePackage=TRUE in conjunction with CreateDefaultPackage to create a new package; however, if a package already exists, it will be replaced when using ReplacePackage=TRUE.

Example for DB2 UDB:

If your initial DB2 UDB connection uses the following URL:

```
jdbc:datadirect:db2://server1:50000;DatabaseName=SAMPLE;PackageName=JDBCPKG;
CreateDefaultPackage=TRUE;ReplacePackage=TRUE
```

Then, the next connection would use:

```
jdbc:datadirect:db2://server1:50000;DatabaseName=SAMPLE;PackageName=JDBCPKG
```

Example for DB2 OS/390 and iSeries:

If your initial DB2 UDB connection uses the following URL:

```
jdbc:datadirect:db2://server1:50000;LocationName=SAMPLE;CollectionId=DEFAULT;
PackageName=JDBCPKG;CreateDefaultPackage=TRUE
```

Then, the next connection would use:

```
jdbc:datadirect:db2://server1:50000;LocationName=SAMPLE;CollectionId=DEFAULT;
PackageName=JDBCPKG
```

For both of the previous examples, three subpackages would be created: JDBCPKGA, JDBCPKGB, and JDBCPKGC. They are used as follows:

A => 100 cursors
B => 100 cursors WITH HOLD
C => 2 cursors for Stored Procedure Calls

Your user ID must have CREATE PACKAGE privileges on the database, or your database administrator must create a package for you.

Data Types

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

Table 3-3. DB2 Data Types

DB2 Data Type	JDBC Data Type
Bigint	BIGINT[1]
Blob	BLOB [2]
Char	CHAR
Char for Bit Data	BINARY
Clob	CLOB
Date	DATE
Decimal	DECIMAL
Double	DOUBLE
Float	FLOAT
Integer	INTEGER
Long Varchar	LONGVARCHAR
Long Varchar for Bit Data	LONGVARBINARY
Numeric	NUMERIC
Real	REAL
Rowid[3]	VARBINARY
Smallint	SMALLINT
Time	TIME
Timestamp	TIMESTAMP
Varchar	VARCHAR
Varchar for Bit Data	VARBINARY

[1]BIGINT is supported only for DB2 UDB 8.1.

[2]BLOB is supported only for DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.

[3]Rowid is supported only for DB2 OS/390 and DB2 iSeries V5R2.

SQL Escape Sequences

See Appendix D "SQL Escape Sequences for JDBC" on page 191 for information about the SQL escape sequences supported by the DB2 driver.

Isolation Levels

DB2 supports the Read Committed transaction isolation level.

Using Scrollable Cursors

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

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

JTA Support

To use JDBC distributed transactions through JTA with the DB2 driver, you must have DB2 UDB 8.1 on Windows NT, Windows 2000, or UNIX.

Large Object (LOB) Support

Retrieving and updating Blobs is supported by the DB2 driver only with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.

Retrieving and updating Clobs is supported by the DB2 driver. The DB2 driver supports Clobs up to a maximum of 2 GB with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2; it supports Clobs up to a maximum of 32 KB with all other DB2 databases.

4 The Informix Driver

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

- Informix Dynamic Server with Universal Data Option 9.2 and higher running on Windows and UNIX via SQLI
- Informix Dynamic Server 2000 9.2 and 9.3 running on Windows and UNIX via SQLI

Data Source and Driver Classes

The data source class for the Informix driver is:

com.ddtek.jdbcx.informix.InformixDataSource

See “Connecting Through Data Sources” on page 30 for information on DataDirect Connect *for* JDBC data sources.

The driver class for the Informix driver is:

com.ddtek.jdbc.informix.InformixDriver

J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the Informix resource adapter is:

com.ddtek.resource.spi.InformixManagedConnectionFactory

See "J2EE Connector Architecture Resource Adapters" on page 24 for information about using DataDirect Connect *for* JDBC drivers as J2EE Connector Architecture resource adapters.

Connection String Properties

You can use the following connection properties with the JDBC Driver Manager or DataDirect Connect *for* JDBC data sources.

Table 4-1 lists the JDBC connection properties supported by the Informix driver, and describes each property. The properties have the form:

```
property=value
```

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

Table 4-1. Informix Connection String Properties

Property	Description
DatabaseName	The name of the database to be used.
InformixServer	The name of the Informix database server to which you want to connect.
MaxPooledStatements OPTIONAL	The maximum number of pooled PreparedStatements for this connection. The default is 0.
Password	A case-insensitive password used to connect to your Informix database. A password is required only if security is enabled on your database. If so, contact your system administrator to get your password.
PortNumber	The TCP port. The default varies depending on operating system.

Table 4-1. Informix Connection String Properties *(cont.)*

Property	Description
ServerName	The IP address (use for DataSource connections only).
User	The case-insensitive default user name used to connect to your Informix database. A user name is required only if security is enabled on your database. If so, contact your system administrator to get your user name.

Data Types

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

Table 4-2. Informix Data Types

Informix Data Type	JDBC Data Type
blob	BLOB
boolean	BIT
byte	LONGVARBINARY
clob	CLOB
char	CHAR
date	DATE
datetime hour to second	TIME
datetime year to day	DATE
datetime year to fraction(5)	TIMESTAMP
datetime year to second	TIMESTAMP
decimal	DECIMAL
float	FLOAT
int8	BIGINT
integer	INTEGER

Table 4-2. Informix Data Types *(cont.)*

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

SQL Escape Sequences

See Appendix D "SQL Escape Sequences for JDBC" on page 191 for information about the SQL escape sequences supported by the Informix driver.

Isolation Levels

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

Using Scrollable Cursors

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

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

5 The Oracle Driver

The DataDirect Connect *for* JDBC Oracle driver (the "Oracle driver") supports Oracle 8i R2 (8.1.6), Oracle 8i R3 (8.1.7), and Oracle 9i running on Windows and UNIX.

Data Source and Driver Classes

The data source class for the Oracle driver is:

com.ddtek.jdbcx.oracle.OracleDataSource

See "Connecting Through Data Sources" on page 30 for information on DataDirect Connect *for* JDBC data sources.

The driver class for the Oracle driver is:

com.ddtek.jdbc.oracle.OracleDriver

J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the Oracle resource adapter is:

com.ddtek.resource.spi.OracleManagedConnectionFactory

See "J2EE Connector Architecture Resource Adapters" on page 24 for information about using DataDirect Connect *for* JDBC drivers as J2EE Connector Architecture resource adapters.

Connection String Properties

You can use the following connection properties with the JDBC Driver Manager or DataDirect Connect *for* JDBC data sources.

Table 5-1 lists the JDBC connection properties supported by the Oracle driver, and describes each property. The properties have the form:

property=value

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

Table 5-1. Oracle Connection String Properties

Property	**Description**
BatchPerformanceWorkaround OPTIONAL	{true | false}. Determines the method used to execute batch operations. When set to true, the native Oracle batch mechanism is used. When set to false, the JDBC 3.0-compliant batch mechanism is used. See "Batch Inserts and Updates" on page 66 for details. The default is false.
CatalogIncludesSynonyms OPTIONAL	{true | false}. When set to true, includes synonyms in the result sets returned from the DatabaseMetaData methods getColumns, getProcedureColumns, and getIndexInfo. The default is false.
FetchTSWTSasTimestamp OPTIONAL	{true | false}. When set to true, allows column values with the Oracle9i TIMESTAMP WITH TIME ZONE data type to be retrieved as a JDBC TIMESTAMP data type. When set to false, column values with the Oracle9i TIMESTAMP WITH TIME ZONE data type must be retrieved as a string. See "TIMESTAMP WITH TIME ZONE Data Type" on page 63 for details. The default is false.
MaxPooledStatements OPTIONAL	The maximum number of pooled PreparedStatements for this connection. The default is 0.

Table 5-1. Oracle Connection String Properties *(cont.)*

Property	**Description**
Password	A case-insensitive password used to connect to your Oracle database. A password is required only if security is enabled on your database. If so, contact your system administrator to get your password.
PortNumber OPTIONAL	The TCP port of the Oracle listener running on the Oracle database server. The default is 1521, which is the Oracle default PortNumber when installing the Oracle database software.
ServerName	Either the IP address or the name (if your network supports named servers) of the server running the Oracle database software. For example, OracleAppServer or 122.23.15.12. Use for DataSource connections only.
SID OPTIONAL	The Oracle System Identifier that refers to the instance of the Oracle database software running on the server. The default is 'ORCL', which is the Oracle default SID when installing the Oracle database software.
User	The case-insensitive default user name used to connect to your Oracle database. A user name is required only if security is enabled on your database. If so, contact your system administrator to get your user name. Operating System authentication is not currently supported in the Oracle driver.

Data Types

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

Table 5-2. Oracle Data Types

Oracle Data Type	JDBC Data Type
bfile	BLOB
blob	BLOB
char	CHAR
clob	CLOB
date	TIMESTAMP
long	LONGVARCHAR
long raw	LONGVARBINARY
nchar	CHAR
nclob	CLOB
number	DECIMAL
number	FLOAT
nvarchar2	VARCHAR
raw	VARBINARY
varchar2	VARCHAR

Table 5-3 lists additional data types supported in Oracle9i only.

Table 5-3. Oracle9i Only Data Types

Oracle Data Type	JDBC Data Type
TIMESTAMP	TIMESTAMP
TIMESTAMP WITH LOCAL TIME ZONE	TIMESTAMP

Table 5-3. Oracle9i Only Data Types (cont.)

Oracle Data Type	JDBC Data Type
TIMESTAMP WITH TIME ZONE	VARCHAR
XMLType	CLOB

Oracle9i Data Type Support

Table 5-3 "Oracle9i Only Data Types" on page 60 lists some new data types supported in Oracle9i only and lists how they are mapped to JDBC data types by the Oracle driver. This section provides details about how the Oracle driver supports these new data types.

Oracle9i Date/Time Data Types

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

Oracle9i Date/Time Session Parameters

At connection, the Oracle driver sets the following date/time Oracle9i session parameters:

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

TIMESTAMP Data Type

The Oracle9i TIMESTAMP data type is mapped to the JDBC TIMESTAMP data type.

TIMESTAMP WITH LOCAL TIME ZONE Data Type

The Oracle9i TIMESTAMP WITH LOCAL TIME ZONE data type is mapped to the java.sql.Types.TIMESTAMP java type.

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

When setting TIMESTAMP WITH LOCAL TIME ZONE columns:

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

TIMESTAMP WITH TIME ZONE Data Type

By default, the Oracle9i TIMESTAMP WITH TIME ZONE data type is mapped to the java.sql.Types.VARCHAR java type.

When retrieving TIMESTAMP WITH TIME ZONE values as a string (using resultSet.getString, for example), the value is returned as the string representation of the timestamp including time zone information. The string representation is formatted in the format specified by the Oracle9i NLS_TIMESTAMP_TZ_FORMAT session parameter.

By default, retrieving TIMESTAMP WITH TIME ZONE values as a timestamp (using resultSet.getTimeStamp, for example) is not supported because the time zone information stored in the database would be lost when the data is converted to a timestamp. To provide backward compatibility with existing applications, you can use the FetchTSWTZasTimestamp connection property to allow TIMESTAMP WITH TIME ZONE values to be fetched as a timestamp. The default value of the FetchTSWTSasTimestamp connection property is false, which disables retrieving TIMESTAMP WITH TIME ZONE values as timestamps. For more information about specifying connection properties, see "Connection String Properties" on page 58.

When setting TIMESTAMP WITH TIME ZONE columns:

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

XMLType Data Type

The Oracle driver supports tables containing columns specified as XMLType. The driver maps the Oracle9i XMLType data type to the Java Clob data type. XMLType columns can be used in queries just like any other column type. The data from XMLType columns can be retrieved as a String, Clob, CharacterStream, or AsciiStream. When inserting or updating XMLType columns, the data to be inserted or updated must be in the form of an XMLType data type.

Oracle9i provides the xmltype() function to construct an XMLType data object. The xmlData argument of the xmltype() function can be specified as a string literal or a parameter marker. If a parameter marker is used, the parameter value may be set using the setString, setClob, setCharacterStream, or setAsciiStream methods.

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

```
//  Insert xml data as a literal
String sql = "insert into XMLTypeTbl values (1,
xmltype('" +
"<emp><empNo>123</empNo><empName>Mark</empName></
emp>'))";

Statement stmt = con.createStatement();
stmt.executeUpdate(sql);
```

The following code inserts data into an XMLType column using a prepared statement:

```
//  Insert xml data as a String parameter
String xmlStr = "<emp><empNo>234</
empNo><empName>Trish</empName></emp>";
String sql = "insert into XMLTypeTbl values (?,
xmltype(?))";

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

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

SQL Escape Sequences

See Appendix D "SQL Escape Sequences for JDBC" on page 191 for information about the SQL escape sequences supported by the Oracle driver.

Isolation Levels

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

Using Scrollable Cursors

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

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

JTA Support

To use JDBC distributed transactions through JTA, you must have version 8.1.7 or higher, and install the Oracle JAVA_XA package on the server.

Batch Inserts and Updates

The DataDirect Connect *for* JDBC Oracle driver provides two mechanisms for supporting batch operations. One mechanism uses the native Oracle batch functionality. This mechanism typically is the faster of the two mechanisms, but it is not compliant with the JDBC 3.0 specification because the native

Oracle functionality returns a single update count for all the operations in the batch. The JDBC 3.0 specification requires individual update counts to be returned for each operation in the batch.

The second mechanism uses code that resides in the driver to execute the batch operations. This mechanism complies with the JDBC 3.0 specification, but it is slower than using the native Oracle batch functionality.

The BatchPerformanceWorkaround connection property determines which batch mechanism is used. If the value of the BatchPerformanceWorkaround connection property is true, the native Oracle batch mechanism is used; otherwise, the JDBC 3.0-compliant mechanism is used. The default value of the BatchPerformanceWorkaround property is false.

6 The SQL Server Driver

The DataDirect Connect *for* JDBC SQL Server driver (the "SQL Server driver") supports Microsoft SQL Server 7.0 and SQL Server 2000 (including SP1 and SP2) running on Windows via TDS.

To use JDBC distributed transactions through JTA, you must install stored procedures for SQL Server. See "Installing Stored Procedures for JTA" on page 76 for details.

Data Source and Driver Classes

The data source class for the SQL Server driver is:

com.ddtek.jdbcx.sqlserver.SQLServerDataSource

See "Connecting Through Data Sources" on page 30 for information on DataDirect Connect *for* JDBC data sources.

The driver class for the SQL Server driver is:

com.ddtek.jdbc.sqlserver.SQLServerDriver

J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the SQL Server resource adapter is:

com.ddtek.resource.spi.SQLServerManagedConnectionFactory

See "J2EE Connector Architecture Resource Adapters" on page 24 for information about using DataDirect Connect *for* JDBC drivers as J2EE Connector Architecture resource adapters.

Connection String Properties

You can use the following connection properties with the JDBC Driver Manager or DataDirect Connect *for* JDBC data sources.

Table 6-1 lists the JDBC connection properties supported by the SQL Server driver, and describes each property. The properties have the form:

property=value

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

Table 6-1. SQL Server Connection String Properties

Property	Description
DatabaseName OPTIONAL	The name of the database to which you want to connect.
MaxPooledStatements OPTIONAL	The maximum number of pooled PreparedStatements for this connection. The default is 0.
Password	A case-insensitive password used to connect to your SQL Server database. A password is required only if security is enabled on your database. If so, contact your system administrator to get your password.
PortNumber OPTIONAL	The TCP port. The default is 1433.

Table 6-1. SQL Server Connection String Properties (cont.)

Property	Description
SelectMethod OPTIONAL	{direct | cursor}. A hint to the SQL Server driver that determines whether the driver should request a database cursor for Select statements. Performance and behavior of the driver are affected by this property, which is defined as a hint because the driver may not always be able to satisfy the requested method. Direct—When the driver uses the direct method, the database server sends the complete result set in a single response to the driver when responding to a query. Normally, responses are not cached by the driver. Using this method, the driver must process all the response to a query before another query is submitted. If another query is submitted (using a different statement on the same connection, for example), the driver caches the response to the first query before submitting the second query. Because the driver must cache the responses when there are multiple active queries, avoid using the direct method when using multiple open result sets that contain large amounts of data. Cursor—When the driver uses the cursor method, a server-side cursor is generated. The rows are fetched from the server in blocks. The JDBC Statement method setFetchSize can be used to control the number of rows that are retrieved for each request. Performance tests show that the value of setFetchSize significantly impacts performance when the cursor method is used. There is no simple rule for determining the setFetchSize value that you should use. We recommend that you experiment with different setFetchSize values to find out which value gives the best performance for your application. The cursor method is useful for queries that produce a large amount of data, particularly if multiple open result sets are used. The default is direct.

Table 6-1. SQL Server Connection String Properties *(cont.)*

Property	Description
SendStringParametersAsUnicode OPTIONAL	{true | false}. Determines whether string parameters are sent to the SQL Server database in Unicode or in the default character encoding of the database. True means that string parameters are sent to SQL Server in Unicode. False means that they are sent in the default encoding, which can improve performance because the server does not need to convert Unicode characters to the default encoding. You should, however, use default encoding only if the parameter string data you specify is consistent with the default encoding of the database. The default is true.
ServerName	The IP address (use for DataSource connections only). SQL Server Driver: To connect to a named instance, specify *server_name*\instance_name for this property, where *server_name* is the IP address and *instance_name* is the name of the instance to which you want to connect on the specified server.
User	The case-insensitive user name used to connect to your SQL Server database. A user name is required only if security is enabled on your database. If so, contact your system administrator to get your user name.
WSID OPTIONAL	The workstation ID, which, typically is the network name of the computer on which the application resides. If specified, this value is stored in the master.dbo.sysprocesses column, hostname, and can be returned by sp_who and the Transact-SQL HOST_NAME function.

Connecting to Named Instances

Microsoft SQL Server 2000 supports multiple instances of a SQL Server database running concurrently on the same server. An instance is identified by an instance name.

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

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

NOTE: The first backslash character (\) in \\instance_name is an escape character.

where:

server_name is the IP address or hostname of the server.

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

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

```
jdbc:datadirect:sqlserver://server1\\instance1;User=test;Password=secret
```

To connect to a named instance using a data source, specify the ServerName connection property as described in "Connection String Properties" on page 70.

Data Types

Table 6-2 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 6-2. SQL Server Data Types

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

Table 6-2. SQL Server Data Types (cont.)

SQL Server Data Type	JDBC Data Type
timestamp	BINARY
tinyint	TINYINT
tinyint identity	TINYINT
uniqueidentifier	CHAR
varbinary	VARBINARY
varchar	VARCHAR

Table 6-3 lists additional data types supported by SQL Server 2000 only.

Table 6-3. SQL Server 2000 Only Data Types

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

SQL Escape Sequences

See Appendix D "SQL Escape Sequences for JDBC" on page 191 for information about the SQL escape sequences supported by the SQL Server driver.

Isolation Levels

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

Using Scrollable Cursors

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

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

Installing Stored Procedures for JTA

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

To install stored procedures for JTA:

1 Copy the sqljdbc.dll file from the DataDirect Connect *for* JDBC *install_dir*/SQLServer JTA directory to the *SQL_Server_Root*/binn directory of the SQL Server database server.

2 From the database server, use the ISQL utility to run the instjdbc.sql script. The system administrator should back up the master database before running instjdbc.sql.

At a command prompt, use the following syntax to run instjdbc.sql:

```
ISQL -Usa -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. This script is located in the DataDirect Connect for JDBC *install_dir*/SQLServer JTA directory.

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

Large Object (LOB) Support

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

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

- Provides random access to data
- Allows searching for patterns in the data, such as retrieving long data that begins with a specific character string
- Allows determining the length of the data before the data is actually retrieved

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

When the SQL Server detects an error in a statement or parameter set in a batch Insert or Update, the SQL Server driver 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 will contain one entry for each statement or parameter set. Any entries for statements or parameter sets that failed will contain the value Statement.EXECUTE_FAILED.

7 The Sybase Driver

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

- Sybase Adaptive Server 11.5 and 11.9 running on Windows and UNIX via TDS
- Sybase Adaptive Server Enterprise 12.0 and 12.5 running on Windows and UNIX via TDS

Data Source and Driver Classes

The data source class for the Sybase driver is:

com.ddtek.jdbcx.sybase.SybaseDataSource

See "Connecting Through Data Sources" on page 30 for information on DataDirect Connect *for* JDBC data sources.

The driver class for the Sybase driver is:

com.ddtek.jdbc.sybase.SybaseDriver

J2EE Connector Architecture Resource Adapter Class

The ManagedConnectionFactory class for the Sybase resource adapter is:

com.ddtek.resource.spi.SybaseManagedConnectionFactory

See "J2EE Connector Architecture Resource Adapters" on page 24 for information about using DataDirect Connect *for* JDBC drivers as J2EE Connector Architecture resource adapters.

Connection String Properties

You can use the following connection properties with the JDBC Driver Manager or DataDirect Connect *for* JDBC data sources.

Table 7-1 lists the JDBC connection properties supported by the Sybase driver, and describes each property. The properties have the form:

```
property=value
```

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

Table 7-1. Sybase Connection String Properties

Property	Description
BatchPerformanceWorkaround OPTIONAL	{true | false}. Determines the method used to execute batch operations. When set to true, the native Sybase batch mechanism is used. When set to false, the JDBC 3.0-compliant batch mechanism is used. See "Batch Inserts and Updates" on page 86. The default is false.
DatabaseName OPTIONAL	The name of the database to which you want to connect.

Table 7-1. Sybase Connection String Properties *(cont.)*

Property	Description
MaxPooledStatements OPTIONAL	The maximum number of pooled PreparedStatements for this connection. The default is 0.
Password	The case-sensitive password used to connect to your Sybase database. A password is required only if security is enabled on your database. If so, contact your system administrator to get your password.
PortNumber	The TCP port. The default varies depending on operating system.

***Table 7-1. Sybase Connection String Properties** (cont.)*

Property	Description
SelectMethod OPTIONAL	{direct | cursor}. A hint to the Sybase driver that determines whether the driver should request a database cursor for Select statements. Performance and behavior of the driver are affected by this property, which is defined as a hint because the driver may not always be able to satisfy the requested method. Direct—When the driver uses the direct method, the database server sends the complete result set in a single response to the driver when responding to a query. Normally, responses are not cached by the driver. Using this method, the driver must process all the response to a query before another query is submitted. If another query is submitted (using a different statement on the same connection, for example), the driver caches the response to the first query before submitting the second query. Because the driver must cache the responses when there are multiple active queries, avoid using the direct method when using multiple open result sets that contain large amounts of data. Cursor—When the driver uses the cursor method, a server-side cursor is generated. The rows are fetched from the server in blocks. The JDBC Statement method setFetchSize can be used to control the number of rows that are retrieved for each request. Performance tests show that the value of setFetchSize significantly impacts performance when the cursor method is used. There is no simple rule for determining the setFetchSize value that you should use. We recommend that you experiment with different setFetchSize values to find out which value gives the best performance for your application. The cursor method is useful for queries that produce a large amount of data, particularly if multiple open result sets are used. The default is direct.
ServerName	The IP address (use for DataSource connections only).

Table 7-1. Sybase Connection String Properties *(cont.)*

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

Data Types

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

Table 7-2. Sybase Data Types

Sybase Data Type	JDBC Data Type
binary	BINARY
bit	BIT
char	CHAR
datetime	TIMESTAMP
decimal	DECIMAL
float	FLOAT
image	LONGVARBINARY
int	INTEGER
money	DECIMAL
nchar	CHAR
numeric	NUMERIC
nvarchar	VARCHAR
real	REAL
smalldatetime	TIMESTAMP

[1] Sybase Adaptive Server Enterprise 12.5 only

Table 7-2. Sybase Data Types (cont.)

Sybase Data Type	JDBC Data Type
smallint	SMALLINT
smallmoney	DECIMAL
sysname	VARCHAR
text	LONGVARCHAR
timestamp	VARBINARY
tinyint	TINYINT
unichar[1]	CHAR
univarchar[1]	VARCHAR
varbinary	VARBINARY
varchar	VARCHAR

[1] Sybase Adaptive Server Enterprise 12.5 only

NOTE FOR USERS OF ADAPTIVE SERVER 12.5: The Sybase driver supports extended new limits (XNL) for character and binary columns—columns with lengths greater than 255. See Appendix B "GetTypeInfo" on page 133 for details.

SQL Escape Sequences

See Appendix D "SQL Escape Sequences for JDBC" on page 191 for information about the SQL escape sequences supported by the Sybase driver.

Isolation Levels

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

Using Scrollable Cursors

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

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

Large Object (LOB) Support

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

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

- Provides random access to data
- Allows searching for patterns in the data, such as retrieving long data that begins with a specific character string
- Allows determining the length of the data before the data is actually retrieved

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

Batch Inserts and Updates

The Sybase driver provides the following batch mechanisms:

- A JDBC 3.0-compliant mechanism that uses code in the driver to execute batch operations. This is the standard mechanism.
- A mechanism that uses the Sybase native batch functionality. This mechanism may be faster than the standard mechanism, particularly when performance-expensive network round-trips are an issue. Be aware that if the execution of the batch results in an error, the driver cannot determine which statement in the batch caused the error. In addition, if the batch contained a statement that called a stored procedure or executed a trigger, multiple update counts for each batch statement or parameter set are generated.

To use the Sybase native batch mechanism, set the BatchPerformanceWorkaround connection property to true. For more information about specifying connection properties, see "Connection String Properties" on page 80.

A JDBC Support

This appendix provides information about JDBC compatibility and developing JDBC applications for DataDirect Connect *for* JDBC environments.

JDBC Compatibility

Table A-1 shows compatibility among the JDBC specification versions, Java Virtual Machines, and the DataDirect Connect *for* JDBC drivers. The DataDirect Connect *for* JDBC drivers do not support Java Virtual Machines 1.0.2 or 1.1.8.

Table A-1. JDBC Compatibility

JDBC Version*	JDK	Drivers Compatible?
1.22	1.2	Yes
2.0	1.2	Yes
2.0	1.3	Yes
3.0	1.2	No
3.0	1.3	No
3.0	1.4	Yes

*Refers to whether the application is using JDBC 1.22, JDBC 2.0 or JDBC 3.0 features.

Supported Functionality

The following tables list functionality supported for each JDBC object.

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

Blob Object **Methods**	**Version Introduced**	**Supported**	**Comments**
InputStream getBinaryStream ()	2.0 Core	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
byte[] getBytes (long, int)	2.0 Core	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.

Blob Object **Methods**	**Version Introduced**	**Supported**	**Comments**
long length ()	2.0 Core	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
long position (Blob, long)	2.0 Core	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
long position (byte[], long)	2.0 Core	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
OutputStream setBinaryStream (long)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
int setBytes (long, byte[])	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.

Blob Object **Methods**	**Version Introduced**	**Supported**	**Comments**
int setBytes (long, byte[], int, int)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
void truncate (long)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.

CallableStatement Object

CallableStatement Object **Methods**	**Version Introduced**	**Supported**	**Comments**
Array getArray (int)	2.0 Core	No	Throws “unsupported method” exception.
Array getArray (String)	3.0	No	Throws “unsupported method” exception.
BigDecimal getBigDecimal (int)	2.0 Core	Yes	
BigDecimal getBigDecimal (int, int)	1.0	Yes	
BigDecimal getBigDecimal (String)	3.0	No	Throws “unsupported method” exception.
Blob getBlob (int)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARBINARY data types.

CallableStatement Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
Blob getBlob (String)	3.0	No	The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
boolean getBoolean (int)	1.0	Yes	
boolean getBoolean (String)	3.0	No	Throws “unsupported method” exception.
byte getByte (int)	1.0	Yes	
byte getByte (String)	3.0	No	Throws “unsupported method” exception.
byte [] getBytes (int)	1.0	Yes	
byte [] getBytes (String)	3.0	No	Throws “unsupported method” exception.
Clob getClob (int)	2.0 Core	Yes	
Clob getClob (String)	3.0	No	Throws “unsupported method” exception.
Date getDate (int)	1.0	Yes	
Date getDate (int, Calendar)	2.0 Core	Yes	
Date getDate (String)	3.0	No	Throws “unsupported method” exception.
Date getDate (String, Calendar)	3.0	No	Throws “unsupported method” exception.
double getDouble (int)	1.0	Yes	
double getDouble (String)	3.0	No	Throws “unsupported method” exception.
float getFloat (int)	1.0	Yes	
float getFloat (String)	3.0	No	Throws “unsupported method” exception.
int getInt (int)	1.0	Yes	

CallableStatement Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
int getInt (String)	3.0	No	Throws “unsupported method” exception.
long getLong (int)	1.0	Yes	
long getLong (String)	3.0	No	Throws “unsupported method” exception.
Object getObject (int)	1.0	Yes	
Object getObject (int, Map)	2.0 Core	Yes	Map ignored.
Object getObject (String)	3.0	No	Throws “unsupported method” exception.
Object getObject (String, Map)	3.0	No	Throws “unsupported method” exception.
Ref getRef (int)	2.0 Core	No	Throws “unsupported method” exception.
Ref getRef (String)	3.0	No	Throws “unsupported method” exception.
short getShort (int)	1.0	Yes	
short getShort (String)	3.0	No	Throws “unsupported method” exception.
String getString (int)	1.0	Yes	
String getString (String)	3.0	No	Throws “unsupported method” exception.
Time getTime (int)	1.0	Yes	
Time getTime (int, Calendar)	2.0 Core	Yes	
Time getTime (String)	3.0	No	Throws “unsupported method” exception.
Time getTime (String, Calendar)	3.0	No	Throws “unsupported method” exception.
Timestamp getTimestamp (int)	1.0	Yes	

CallableStatement Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
Timestamp getTimestamp (int, Calendar)	2.0 Core	Yes	
Timestamp getTimestamp (String)	3.0	No	Throws “unsupported method” exception.
Timestamp getTimestamp (String, Calendar)	3.0	No	Throws “unsupported method” exception.
URL getURL (int)	3.0	No	Throws “unsupported method” exception.
URL getURL (String)	3.0	No	Throws “unsupported method” exception.
void registerOutParameter (int, int)	1.0	Yes	
void registerOutParameter (int, int, int)	1.0	Yes	
void registerOutParameter (int, int, String)	2.0 Core	Yes	String/typename ignored.
void registerOutParameter (String, int)	3.0	No	Throws “unsupported method” exception.
void registerOutParameter (String, int, int)	3.0	No	Throws “unsupported method” exception.
void registerOutParameter (String, int, String)	3.0	No	Throws “unsupported method” exception.
void setArray (int, Array)	2.0 Core	No	Throws “unsupported method” exception.
void setAsciiStream (String, InputStream, int)	3.0	No	Throws “unsupported method” exception.
void setBigDecimal (String, BigDecimal)	3.0	No	Throws “unsupported method” exception.
void setBinaryStream (String, InputStream, int)	3.0	No	Throws “unsupported method” exception.

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

CallableStatement Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
void setShort (String, short)	3.0	No	Throws “unsupported method” exception.
void setString (String, String)	3.0	No	Throws “unsupported method” exception.
void setTime (String, Time)	3.0	No	Throws “unsupported method” exception.
void setTime (String, Time, Calendar)	3.0	No	Throws “unsupported method” exception.
void setTimestamp (String, Timestamp)	3.0	No	Throws “unsupported method” exception.
void setTimestamp (String, Timestamp, Calendar)	3.0	No	Throws “unsupported method” exception.
void setURL (String, URL)	3.0	No	Throws “unsupported method” exception.
boolean wasNull ()	1.0	Yes	

Clob Object

Clob Object **Methods**	**Version Introduced**	**Supported**	**Comments**
InputStream getAsciiStream ()	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
Reader getCharacterStream ()	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.

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

Connection Object

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

Connection Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
String nativeSQL (String)	1.0	Yes	Always returns same String as passed in.
CallableStatement prepareCall (String)	1.0	Yes	ResultSet.TYPE_SCROLL_ SENSITIVE downgraded to TYPE_SCROLL_INSENSITIVE for the DB2 driver.
CallableStatement prepareCall (String, int, int)	2.0 Core	Yes	ResultSet.TYPE_SCROLL_ SENSITIVE downgraded to TYPE_SCROLL_INSENSITIVE for the DB2 and Sybase drivers.
CallableStatement prepareCall (String, int, int, int)	3.0	No	Throws "unsupported method" exception.
PreparedStatement prepareStatement (String)	1.0	Yes	ResultSet.TYPE_SCROLL_ SENSITIVE downgraded to TYPE_SCROLL_INSENSITIVE for the DB2 driver.
PreparedStatement prepareStatement (String, int)	3.0	No	Throws "unsupported method" exception.
PreparedStatement prepareStatement (String, int, int)	2.0 Core	Yes	ResultSet.TYPE_SCROLL_ SENSITIVE downgraded to TYPE_SCROLL_INSENSITIVE for the DB2 driver.
PreparedStatement prepareStatement (String, int, int, int)	3.0	No	Throws "unsupported method" exception.
PreparedStatement prepareStatement (String, int[])	3.0	No	Throws "unsupported method" exception.
PreparedStatement prepareStatement (String, String [])	3.0	No	Throws "unsupported method" exception.

Connection Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
void releaseSavepoint (Savepoint)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.
void rollback ()	1.0	Yes	
void rollback (Savepoint)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.
void setAutoCommit (boolean)	1.0	Yes	
void setCatalog (String)	1.0	Yes	Support is driver-specific.
void setHoldability (int)	3.0	No	Throws “unsupported method” exception.
void setReadOnly (boolean)	1.0	Yes	
Savepoint setSavepoint ()	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.
Savepoint setSavepoint (String)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.
void setTransactionIsolation (int)	1.0	Yes	
void setTypeMap (Map)	2.0 Core	Yes	Ignored.

ConnectionPoolDataSource Object

ConnectionPoolData Source Object Methods	**Version Introduced**	**Supported**	**Comments**
int getLoginTimeout ()	2.0 Optional	Yes	
PrintWriter getLogWriter ()	2.0 Optional	Yes	
PooledConnection getPooledConnection ()	2.0 Optional	Yes	
PooledConnection getPooledConnection (String, String)	2.0 Optional	Yes	
void setLoginTimeout (int)	2.0 Optional	Yes	
void setLogWriter (PrintWriter)	2.0 Optional	Yes	

DatabaseMetaData 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	

DatabaseMetaData Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
boolean doesMaxRowSizeIncludeBlobs ()	1.0	Yes	Not supported by the SQL Server and Sybase drivers.
ResultSet getAttributes (String, String, String, String)	3.0	No	Throws "unsupported method" exception.
ResultSet getBestRowIdentifier (String, String, String, int, boolean)	1.0	Yes	
ResultSet getCatalogs ()	1.0	Yes	
String getCatalogSeparator ()	1.0	Yes	
String getCatalogTerm ()	1.0	Yes	
ResultSet getColumnPrivileges (String, String, String, String)	1.0	Yes	
ResultSet getColumns (String, String, String, String)	1.0	Yes	
Connection getConnection ()	2.0 Core	Yes	
ResultSet getCrossReference (String, String, String, String, String, String)	1.0	Yes	
int getDatabaseMajorVersion ()	3.0	Yes	
int getDatabaseMinorVersion ()	3.0	Yes	
String getDatabaseProductName ()	1.0	Yes	
String getDatabaseProductVersion ()	1.0	Yes	
int getDefaultTransactionIsolation ()	1.0	Yes	
int getDriverMajorVersion ()	1.0	Yes	
int getDriverMinorVersion ()	1.0	Yes	
String getDriverName ()	1.0	Yes	

DatabaseMetaData Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
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	No	Throws "unsupported method" exception.
int getJDBCMinorVersion ()	3.0	No	Throws "unsupported method" exception.
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	

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

DatabaseMetaData Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
ResultSet getSuperTables (String, String, String)	3.0	No	Throws "unsupported method" exception.
ResultSet getSuperTypes (String, String, String)	3.0	No	Throws "unsupported method" exception.
String getSystemFunctions ()	1.0	Yes	
ResultSet getTablePrivileges (String, String, String)	1.0	Yes	
ResultSet getTables (String, String, String, String [])	1.0	Yes	
ResultSet getTableTypes ()	1.0	Yes	
String getTimeDateFunctions ()	1.0	Yes	
ResultSet getTypeInfo ()	1.0	Yes	
ResultSet getUDTs (String, String, String, int [])	2.0 Core	No	Always returns empty ResultSet.
String getURL ()	1.0	Yes	
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	

DatabaseMetaData Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
boolean nullsAreSortedAtStart ()	1.0	Yes	
boolean nullsAreSortedHigh ()	1.0	Yes	
boolean nullsAreSortedLow ()	1.0	Yes	
boolean othersDeletesAreVisible (int)	2.0 Core	Yes	
boolean othersInsertsAreVisible (int)	2.0 Core	Yes	
boolean othersUpdatesAreVisible (int)	2.0 Core	Yes	
boolean ownDeletesAreVisible (int)	2.0 Core	Yes	
boolean ownInsertsAreVisible (int)	2.0 Core	Yes	
boolean ownUpdatesAreVisible (int)	2.0 Core	Yes	
boolean storesLowerCaseIdentifiers ()	1.0	Yes	
boolean storesLowerCaseQuoted Identifiers ()	1.0	Yes	
boolean storesMixedCaseIdentifiers ()	1.0	Yes	
boolean storesMixedCaseQuoted Identifiers ()	1.0	Yes	
boolean storesUpperCaseIdentifiers ()	1.0	Yes	
boolean storesUpperCaseQuoted Identifiers ()	1.0	Yes	
boolean supportsAlterTableWith AddColumn ()	1.0	Yes	
boolean supportsAlterTableWith DropColumn ()	1.0	Yes	
boolean supportsANSI92EntryLevelSQL ()	1.0	Yes	
boolean supportsANSI92FullSQL ()	1.0	Yes	
boolean supportsANSI92Intermediate SQL ()	1.0	Yes	
boolean supportsBatchUpdates ()	2.0 Core	Yes	

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

DatabaseMetaData Object *(cont.)* **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	
boolean supportsPositionedUpdate ()	1.0	Yes	

DatabaseMetaData Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
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	
boolean supportsTransactions ()	1.0	Yes	
boolean supportsUnion ()	1.0	Yes	

DatabaseMetaData Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
boolean supportsUnionAll ()	1.0	Yes	
boolean updatesAreDetected (int)	2.0 Core	Yes	
boolean usesLocalFilePerTable ()	1.0	Yes	
boolean usesLocalFiles ()	1.0	Yes	

DataSource Object

DataSource Object **Methods**	**Version Introduced**	**Supported**	**Comments**
Connection getConnection ()	2.0 Optional	Yes	
Connection getConnection (String, String)	2.0 Optional	Yes	
int getLoginTimeout ()	2.0 Optional	Yes	
PrintWriter getLogWriter ()	2.0 Optional	Yes	
void setLoginTimeout (int)	2.0 Optional	Yes	
void setLogWriter (PrintWriter)	2.0 Optional	Yes	Enables DataDirect Spy, which traces JDBC information into the specified PrintWriter.

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

ParameterMetaData Object Methods	Version Introduced	Supported	Comments
String getParameterClassName (int)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1 and DB2 iSeries V5R2. "ParameterMetaData is not available" exception is thrown for all other DB2 UDB, DB2 iSeries, Oracle, and SQL Server. Availability for Informix and Sybase varies by statement type.
int getParameterCount ()	3.0	Yes	

ParameterMetaData Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
int getParameterMode (int)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1 and DB2 iSeries V5R2. "ParameterMetaData is not available" exception is thrown for all other DB2 UDB, DB2 iSeries, Oracle, and SQL Server. Availability for Informix and Sybase varies by statement type.
int getParameterType (int)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1 and DB2 iSeries V5R2. "ParameterMetaData is not available" exception is thrown for all other DB2 UDB, DB2 iSeries, Oracle, and SQL Server. Availability for Informix and Sybase varies by statement type.
String getParameterTypeName (int)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1 and DB2 iSeries V5R2. "ParameterMetaData is not available" exception is thrown for all other DB2 UDB, DB2 iSeries, Oracle, and SQL Server. Availability for Informix and Sybase varies by statement type.

ParameterMetaData Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
int getPrecision (int)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1 and DB2 iSeries V5R2. "ParameterMetaData is not available" exception is thrown for all other DB2 UDB, DB2 iSeries, Oracle, and SQL Server. Availability for Informix and Sybase varies by statement type.
int getScale (int)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1 and DB2 iSeries V5R2. "ParameterMetaData is not available" exception is thrown for all other DB2 UDB, DB2 iSeries, Oracle, and SQL Server. Availability for Informix and Sybase varies by statement type.
int isNullable (int)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1 and DB2 iSeries V5R2. "ParameterMetaData is not available" exception is thrown for all other DB2 UDB, DB2 iSeries, Oracle, and SQL Server. Availability for Informix and Sybase varies by statement type.

ParameterMetaData Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
boolean isSigned (int)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1 and DB2 iSeries V5R2. "ParameterMetaData is not available" exception is thrown for all other DB2 UDB, DB2 iSeries, Oracle, and SQL Server. Availability for Informix and Sybase varies by statement type.
boolean jdbcCompliant ()	1.0	Yes	

PooledConnection Object

PooledConnection Object **Methods**	**Version Introduced**	**Supported**	**Comments**
void addConnectionEventListener (ConnectionEventListener)	2.0 Optional	Yes	
void close()	2.0 Optional	Yes	

PooledConnection Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
Connection getConnection()	2.0 Optional	Yes	A pooled connection object can have only one Connection object open (the one most recently created). The purpose of allowing the server (PoolManager implementation) to invoke the method getConnection a second time is to give that application server a way to take a connection away from an application and give it to another user (a rare occurrence). The drivers do not support the "reclaiming" of connections and will throw a SQLException "Reclaim of open connection is not supported."
void removeConnectionEvent Listener (ConnectionEventListener)	2.0 Optional	Yes	

PreparedStatement Object

PreparedStatement Object **Methods**	**Version Introduced**	**Supported**	**Comments**
void addBatch ()	2.0 Core	Yes	
void clearParameters ()	1.0	Yes	

PreparedStatement Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
boolean execute ()	1.0	Yes	
ResultSet executeQuery ()	1.0	Yes	
int executeUpdate ()	1.0	Yes	
ResultSetMetaData getMetaData ()	2.0 Core	Yes	
ParameterMetaData getParameterMetaData ()	3.0	Yes	
void setArray (int, Array)	2.0 Core	No	Throws "unsupported method" exception.
void setAsciiStream (int, InputStream, int)	1.0	Yes	
void setBigDecimal (int, BigDecimal)	1.0	Yes	
void setBinaryStream (int, InputStream, int)	1.0	Yes	When used with Blobs, the DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.
void setBlob (int, Blob)	2.0 Core	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
void setBoolean (int, boolean)	1.0	Yes	
void setByte (int, byte)	1.0	Yes	

PreparedStatement Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
void setBytes (int, byte [])	1.0	Yes	When used with Blobs, the DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.
void setCharacterStream (int, Reader, int)	2.0 Core	Yes	
void setClob (int, Clob)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
void setDate (int, Date)	1.0	Yes	
void setDate (int, Date, Calendar)	2.0 Core	Yes	
void setDouble (int, double)	1.0	Yes	
void setFloat (int, float)	1.0	Yes	
void setInt (int, int)	1.0	Yes	
void setLong (int, long)	1.0	Yes	
void setNull (int, int)	1.0	Yes	
void setNull (int, int, String)	2.0 Core	Yes	
void setObject (int, Object)	1.0	Yes	
void setObject (int, Object, int)	1.0	Yes	
void setObject (int, Object, int, int)	1.0	Yes	
void setQueryTimeout (int)	1.0	Yes	Throws “unsupported method” exception for DB2 and Informix.
void setRef (int, Ref)	2.0 Core	No	Throws “unsupported method” exception.

PreparedStatement Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
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 has been deprecated in the JDBC 3.0 specification.
void setURL (int, URL)	3.0	No	Throws "unsupported method" exception.

Ref Object

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

ResultSet Object

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

ResultSet Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
InputStream getBinaryStream (int)	1.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.
InputStream getBinaryStream (String)	1.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.
Blob getBlob (int)	2.0 Core	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
Blob getBlob (String)	2.0 Core	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2. The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
boolean getBoolean (int)	1.0	Yes	
boolean getBoolean (String)	1.0	Yes	
byte getByte (int)	1.0	Yes	
byte getByte (String)	1.0	Yes	
byte [] getBytes (int)	1.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.
byte [] getBytes (String)	1.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.
Reader getCharacterStream (int)	2.0 Core	Yes	
Reader getCharacterStream (String)	2.0 Core	Yes	

ResultSet Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
Clob getClob (int)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
Clob getClob (String)	2.0 Core	Yes	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
int getConcurrency ()	2.0 Core	Yes	
String getCursorName ()	1.0	No	Throws "unsupported method" exception.
Date getDate (int)	1.0	Yes	
Date getDate (int, Calendar)	2.0 Core	Yes	
Date getDate (String)	1.0	Yes	
Date getDate (String, Calendar)	2.0 Core	Yes	
double getDouble (int)	1.0	Yes	
double getDouble (String)	1.0	Yes	
int getFetchDirection ()	2.0 Core	Yes	
int getFetchSize ()	2.0 Core	Yes	
float getFloat (int)	1.0	Yes	
float getFloat (String)	1.0	Yes	
int getInt (int)	1.0	Yes	
int getInt (String)	1.0	Yes	
long getLong (int)	1.0	Yes	
long getLong (String)	1.0	Yes	
ResultSetMetaData getMetaData ()	1.0	Yes	
Object getObject (int)	1.0	Yes	

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

ResultSet Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
int getType ()	2.0 Core	Yes	
InputStream getUnicodeStream (int)	1.0	No	Throws "unsupported method" exception.
InputStream getUnicodeStream (String)	1.0	No	Throws "unsupported method" exception.
URL getURL (int)	3.0	No	Throws "unsupported method" exception.
URL getURL (String)	3.0	No	Throws "unsupported method" exception.
SQLWarning getWarnings ()	1.0	Yes	
void insertRow ()	2.0 Core	Yes	
boolean isAfterLast ()	2.0 Core	Yes	
boolean isBeforeFirst ()	2.0 Core	Yes	
boolean isFirst ()	2.0 Core	Yes	
boolean isLast ()	2.0 Core	Yes	
boolean last ()	2.0 Core	Yes	
void moveToCurrentRow ()	2.0 Core	Yes	
void moveToInsertRow ()	2.0 Core	Yes	
boolean next ()	1.0	Yes	
boolean previous ()	2.0 Core	Yes	
void refreshRow ()	2.0 Core	Yes	
boolean relative (int)	2.0 Core	Yes	
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	

ResultSet Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
void setFetchSize (int)	2.0 Core	Yes	
void updateArray (int, Array)	3.0	No	Throws “unsupported method” exception.
void updateArray (String, Array)	3.0	No	Throws “unsupported method” exception.
void updateAsciiStream (int, InputStream, int)	2.0 Core	Yes	
void updateAsciiStream (String, InputStream, int)	2.0 Core	Yes	
void updateBigDecimal (int, BigDecimal)	2.0 Core	Yes	
void updateBigDecimal (String, BigDecimal)	2.0 Core	Yes	
void updateBinaryStream (int, InputStream, int)	2.0 Core	Yes	
void updateBinaryStream (String, InputStream, int)	2.0 Core	Yes	
void updateBlob (int, Blob)	3.0	No	The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
void updateBlob (String, Blob)	3.0	No	The SQL Server and Sybase drivers support using with LONGVARBINARY data types.
void updateBoolean (int, boolean)	2.0 Core	Yes	
void updateBoolean (String, boolean)	2.0 Core	Yes	
void updateByte (int, byte)	2.0 Core	Yes	
void updateByte (String, byte)	2.0 Core	Yes	
void updateBytes (int, byte [])	2.0 Core	Yes	

ResultSet Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
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	No	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
void updateClob (String, Clob)	3.0	No	The SQL Server and Sybase drivers support using with LONGVARCHAR data types.
void updateDate (int, Date)	2.0 Core	Yes	
void updateDate (String, Date)	2.0 Core	Yes	
void updateDouble (int, double)	2.0 Core	Yes	
void updateDouble (String, double)	2.0 Core	Yes	
void updateFloat (int, float)	2.0 Core	Yes	
void updateFloat (String, float)	2.0 Core	Yes	
void updateInt (int, int)	2.0 Core	Yes	
void updateInt (String, int)	2.0 Core	Yes	
void updateLong (int, long)	2.0 Core	Yes	
void updateLong (String, long)	2.0 Core	Yes	
void updateNull (int)	2.0 Core	Yes	
void updateNull (String)	2.0 Core	Yes	

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

ResultSetMetaData Object

ResultSetMetaData Object **Methods**	**Version Introduced**	**Supported**	**Comments**
String getCatalogName (int)	1.0	Yes	
String getColumnClassName (int)	2.0 Core	Yes	
int getColumnCount ()	1.0	Yes	
int getColumnDisplaySize (int)	1.0	Yes	
String getColumnLabel (int)	1.0	Yes	
String getColumnName (int)	1.0	Yes	
int getColumnType (int)	1.0	Yes	
String getColumnTypeName (int)	1.0	Yes	
int getPrecision (int)	1.0	Yes	
int getScale (int)	1.0	Yes	
String getSchemaName (int)	1.0	Yes	
String getTableName (int)	1.0	Yes	
boolean isAutoIncrement (int)	1.0	Yes	
boolean isCaseSensitive (int)	1.0	Yes	
boolean isCurrency (int)	1.0	Yes	
boolean isDefinitelyWritable (int)	1.0	Yes	
int isNullable (int)	1.0	Yes	
boolean isReadOnly (int)	1.0	Yes	
boolean isSearchable (int)	1.0	Yes	
boolean isSigned (int)	1.0	Yes	

ResultSetMetaData Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
boolean isWritable (int)	1.0	Yes	

RowSet Object

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

SavePoint Object

SavePoint Object Methods	**Version Introduced**	**Supported**	**Comments**
(all)	3.0	Yes	The DB2 driver only supports with DB2 UDB 8.1, DB2 OS/390, and DB2 iSeries V5R2.

Serializable Object

Serializable Object Methods	**Version Introduced**	**Supported**	**Comments**
(N/A)	java.io	Yes	Implemented by DataSource classes.

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	Throws "unsupported method" exception for DB2 (except for DB2 UDB 8.1) and Informix.
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	No	Throws "unsupported method" exception.
boolean execute (String, int [])	3.0	No	Throws "unsupported method" exception.
boolean execute (String, String [])	3.0	No	Throws "unsupported method" exception.
int [] executeBatch ()	2.0 Core	Yes	
ResultSet executeQuery (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.
int executeUpdate (String)	1.0	Yes	Throws "invalid method call" exception for PreparedStatement and CallableStatement.

Statement Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
int executeUpdate (String, int)	3.0	No	Throws "unsupported method" exception.
int executeUpdate (String, int [])	3.0	No	Throws "unsupported method" exception.
int executeUpdate (String, String [])	3.0	No	Throws "unsupported method" exception.
Connection getConnection ()	2.0 Core	Yes	
int getFetchDirection ()	2.0 Core	Yes	
int getFetchSize ()	2.0 Core	Yes	
ResultSet getGeneratedKeys ()	3.0	No	Throws "unsupported method" exception.
int getMaxFieldSize ()	1.0	Yes	
int getMaxRows ()	1.0	Yes	
boolean getMoreResults ()	1.0	Yes	
boolean getMoreResults (int)	3.0	Yes	
int getQueryTimeout ()	1.0	Yes	Returns 0 for DB2 (except for DB2 UDB 8.1) and Informix.
ResultSet getResultSet ()	1.0	Yes	
int getResultSetConcurrency ()	2.0 Core	Yes	
int getResultSetHoldability ()	3.0	No	Throws "unsupported method" exception.
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.

Statement Object *(cont.)* **Methods**	**Version Introduced**	**Supported**	**Comments**
void setEscapeProcessing (boolean)	1.0	Yes	Ignored.
void setFetchDirection (int)	2.0 Core	Yes	
void setFetchSize (int)	2.0 Core	Yes	
void setMaxFieldSize (int)	1.0	Yes	
void setMaxRows (int)	1.0	Yes	
void setQueryTimeout (int)	1.0	Yes	Throws "unsupported method" exception for DB2 (except for DB2 UDB 8.1) and Informix.

Struct Object

Struct Object **Methods**	**Version Introduced**	**Supported**	**Comments**
(all)	2.0	No	

XAConnection Object

XAConnection Object **Methods**	**Version Introduced**	**Supported**	**Comments**
(all)	2.0 Optional	Yes	

XADataSource Object

XADataSource Object Methods	Version Introduced	Supported	Comments
(all)	2.0 Optional	Yes	

B GetTypeInfo

The following tables provide results returned from the method DataBaseMetaData.getTypeInfo for all of the DataDirect Connect *for* JDBC drivers. The tables are alphabetical first by driver, then by TYPE_NAME, and then by parameter.

DB2 Driver

Table B-1. GetTypeInfo for DB2

TYPE_NAME = bigint*

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -5 (BIGINT)	PRECISION = 20
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	

* BIGINT is supported only for DB2 UDB on Windows and UNIX

Table B-1. GetTypeInfo for DB2 (cont.)

TYPE_NAME = blob

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = length
DATA_TYPE = 2004 (BLOB)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = BLOB
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 2147483647
SEARCHABLE = 1
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = char

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = length
DATA_TYPE = 1 (CHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = '
LITERAL_SUFFIX = '
LOCAL_TYPE_NAME = char
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 254
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = char for bit data

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = length
DATA_TYPE = -2 (BINARY)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = X'
LITERAL_SUFFIX = '
LOCAL_TYPE_NAME = char for bit data
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 254
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

Table B-1. GetTypeInfo for DB2 (cont.)

TYPE_NAME = clob

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = length	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 32700
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = clob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = date

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {d‘	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ‘}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = decimal

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = precision,scale	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 31
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = decimal	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 31	

Table B-1. GetTypeInfo for DB2 (cont.)

TYPE_NAME = double

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 8 (DOUBLE)	PRECISION = 15
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = double	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

TYPE_NAME = float

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

TYPE_NAME = integer

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

Table B-1. GetTypeInfo for DB2 (cont.)

TYPE_NAME = long varchar

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 32704 (UDB) 32700 (OS/390)
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = long varchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = long varchar for bit data

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -4 (LONGVARBINARY)	PRECISION = 32700 (UDB) 32698 (OS/390)
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = X’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = long varchar for bit data	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = numeric

AUTO_INCREMENT = 0	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = precision, scale	NUM_PREC_RADIX =10
DATA_TYPE = 2 (NUMERIC)	PRECISION = 31
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = numeric	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 31	

Table B-1. GetTypeInfo for DB2 (cont.)

TYPE_NAME = real

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = float(4)	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

TYPE_NAME = rowid *

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = not null generated always	NUM_PREC_RADIX = NULL
DATA_TYPE = -2 (B)	PRECISION = 40
FIXED_PREC_SCALE = false	SEARCHABLE = 2**
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = ROWID	UNSIGNED_ATTRIBUTE = true
MAXIMUM_SCALE = 0	

* Only supported for DB2 iSeries V5R2.

** Supported except for WHERE ... LIKE.

TYPE_NAME = smallint

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

***Table B-1. GetTypeInfo for DB2** (cont.)*

TYPE_NAME = time

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 92 (TIME)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = {t'
LITERAL_SUFFIX = '}
LOCAL_TYPE_NAME = time
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 8
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = timestamp

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 93 (TIMESTAMP)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = {ts'
LITERAL_SUFFIX = '}
LOCAL_TYPE_NAME = timestamp
MAXIMUM_SCALE = 6
MINIMUM_SCALE = 6
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 26
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = varchar

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = max length
DATA_TYPE = 12 (VARCHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = '
LITERAL_SUFFIX = '
LOCAL_TYPE_NAME = varchar
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 32704 (UDB) 32698 (OS/390)
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

Table B-1. GetTypeInfo for DB2 (cont.)

TYPE_NAME = varchar for bit data

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION = 32704 (UDB) 32698 (OS/390)
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = X’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar for bit data	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Informix Driver

Table B-2. GetTypeInfo for Informix

TYPE_NAME = blob

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

TYPE_NAME = boolean

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = -7 (BIT)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = boolean
MAXIMUM_SCALE = 0
MINIMUM_SCALE = 0
NULLABLE = 1
NUM_PREC_RADIX = 10
PRECISION = 1
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = byte

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

Table B-2. GetTypeInfo for Informix (cont.)

TYPE_NAME = char

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = length	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 32766
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = char	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = clob

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = clob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = date

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {d’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = date	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

***Table B-2. GetTypeInfo for Informix** (cont.)*

TYPE_NAME = datetime hour to second

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 92 (TIME)	PRECISION = 8
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {t’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime hour to second	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

TYPE_NAME = datetime year to day

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 91 (DATE)	PRECISION = 10
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {d’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime year to day	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = datetime year to fraction(5)

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 5
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 25
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime hour to fraction(5)	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 5	

Table B-2. GetTypeInfo for Informix (cont.)

TYPE_NAME = datetime year to second

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = {ts'	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = datetime hour to second	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 0	

TYPE_NAME = decimal

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = precision, scale	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 32
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = decimal	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 32	

TYPE_NAME = float

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

Table B-2. GetTypeInfo for Informix (cont.)

TYPE_NAME = int8

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = -5 (BIGINT)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = int8	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

TYPE_NAME = integer

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

TYPE_NAME = lvarchar

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

Table B-2. GetTypeInfo for Informix (cont.)

TYPE_NAME = money

AUTO_INCREMENT = false	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = precision,scale	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 32
FIXED_PREC_SCALE = true	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = money	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 32	

TYPE_NAME = nchar

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = length	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 32766
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = nvarchar

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 254
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nvarchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

***Table B-2. GetTypeInfo for Informix** (cont.)*

TYPE_NAME = serial

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

TYPE_NAME = serial8

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

TYPE_NAME = smallfloat

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = smallfloat	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

***Table B-2. GetTypeInfo for Informix** (cont.)*

TYPE_NAME = smallint

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

TYPE_NAME = text

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = text	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = varchar

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 254
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Oracle Driver

Table B-3 provides results for Oracle 8i and 9i. Table B-4 provides additional results for Oracle 9i only.

Table B-3. GetTypeInfo for Oracle

TYPE_NAME = bfile

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

TYPE_NAME = blob

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

TYPE_NAME = char

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = length
DATA_TYPE = 1 (CHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = '
LITERAL_SUFFIX = '
LOCAL_TYPE_NAME = char
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 2000
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

Table B-3. GetTypeInfo for Oracle *(cont.)*

TYPE_NAME = clob

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = NULL
DATA_TYPE = 2005 (CLOB)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = '
LITERAL_SUFFIX = '
LOCAL_TYPE_NAME = clob
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 2147483647
SEARCHABLE = 0
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = date

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 93 (TIMESTAMP)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = {ts'
LITERAL_SUFFIX = '}
LOCAL_TYPE_NAME = date
MAXIMUM_SCALE = 0
MINIMUM_SCALE = 0
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 19
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = long

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = NULL
DATA_TYPE = -1 (LONGVARCHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = '
LITERAL_SUFFIX = '
LOCAL_TYPE_NAME = long
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 2147483647
SEARCHABLE = 0
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

Table B-3. GetTypeInfo for Oracle (cont.)

TYPE_NAME = long raw

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

TYPE_NAME = nchar

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = length	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 2000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = nclob

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nclob	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

***Table B-3. GetTypeInfo for Oracle** (cont.)*

TYPE_NAME = number

AUTO_INCREMENT = false
CASE_SENSITIVE = false
CREATE_PARAMS = precision, scale
DATA_TYPE = 3 (DECIMAL)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = number
MAXIMUM_SCALE = 127
MINIMUM_SCALE = -84
NULLABLE = 1
NUM_PREC_RADIX = 10
PRECISION = 38
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = false

TYPE_NAME = number

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

TYPE_NAME = nvarchar2

AUTO_INCREMENT = NULL
CASE_SENSITIVE = true
CREATE_PARAMS = max length
DATA_TYPE = 12 (VARCHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = '
LITERAL_SUFFIX = '
LOCAL_TYPE_NAME = nvarchar2
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 4000
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

Table B-3. GetTypeInfo for Oracle (cont.)

TYPE_NAME = raw

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

TYPE_NAME = varchar2

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = ’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar2	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Oracle 9i Only

Table B-4. GetTypeInfo for Oracle 9i Only

TYPE_NAME = timestamp	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = fractional_seconds_precision	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2*
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 9	
TYPE_NAME = timestamp with local time zone	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = fractional_seconds_precision	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2*
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp with local time zone	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 9	
TYPE_NAME = timestamp with time zone	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = fractional_seconds_precision	NUM_PREC_RADIX = NULL
DATA_TYPE = 93 (TIMESTAMP)	PRECISION = 19
FIXED_PREC_SCALE = false	SEARCHABLE = 2*
LITERAL_PREFIX = {ts '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’}	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = timestamp with time zone	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = 9	

* Supported except for WHERE ... LIKE

Table B-4. GetTypeInfo for Oracle 9i Only (cont.)

TYPE_NAME = XMLTYPE

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = 2005 (CLOB)	PRECISION = 2147483647
FIXED_PREC_SCALE = false	SEARCHABLE = 0
LITERAL_PREFIX = xmltype('	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ')	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = XMLTYPE	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

* Supported except for WHERE ... LIKE

SQL Server Driver

Table B-5 provides results for SQL Server 7 and SQL Server 2000. Table B-6 provides additional results for SQL Server 2000 only.

Table B-5. GetTypeInfo for SQL Server

TYPE_NAME = binary

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

***Table B-5. GetTypeInfo for SQL Server** (cont.)*

TYPE_NAME = bit

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = -7 (BIT)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = bit
MAXIMUM_SCALE = 0
MINIMUM_SCALE = 0
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 1
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = char

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = length
DATA_TYPE = 1 (CHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = '
LITERAL_SUFFIX = '
LOCAL_TYPE_NAME = char
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 8000
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = datetime

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

***Table B-5. GetTypeInfo for SQL Server** (cont.)*

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 = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = decimal	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 38	

TYPE_NAME = decimal() identity

AUTO_INCREMENT = true	MINIMUM_SCALE = 0
CASE_SENSITIVE = false	NULLABLE = 0
CREATE_PARAMS = precision	NUM_PREC_RADIX = 10
DATA_TYPE = 3 (DECIMAL)	PRECISION = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = decimal() identity	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 0	

TYPE_NAME = float

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

***Table B-5. GetTypeInfo for SQL Server** (cont.)*

TYPE_NAME = image

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

TYPE_NAME = int

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

TYPE_NAME = int identity

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

***Table B-5. GetTypeInfo for SQL Server** (cont.)*

TYPE_NAME = money

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

TYPE_NAME = nchar

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = length	NUM_PREC_RADIX = NULL
DATA_TYPE = 1 (CHAR)	PRECISION = 4000
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = nchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

TYPE_NAME = ntext

AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = NULL
DATA_TYPE = -1 (LONGVARCHAR)	PRECISION = 1073741823
FIXED_PREC_SCALE = false	SEARCHABLE = 1
LITERAL_PREFIX = N’	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = ’	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = ntext	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

Table B-5. GetTypeInfo for SQL Server (cont.)

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 = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = numeric	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = 38	

TYPE_NAME = 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 = 38
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = numeric() 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	

***Table B-5. GetTypeInfo for SQL Server** (cont.)*

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-5. GetTypeInfo for SQL Server (cont.)

TYPE_NAME = smallint identity

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

TYPE_NAME = smallmoney

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

TYPE_NAME = sysname

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = 12 (VARCHAR)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = N’
LITERAL_SUFFIX = ’
LOCAL_TYPE_NAME = sysname
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 0
NUM_PREC_RADIX = NULL
PRECISION = 128
SEARCHABLE = 3
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

Table B-5. GetTypeInfo for SQL Server (cont.)

TYPE_NAME = text

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

TYPE_NAME = 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	

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	

Table B-5. GetTypeInfo for SQL Server (cont.)

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	

TYPE_NAME = varbinary

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

***Table B-5. GetTypeInfo for SQL Server** (cont.)*

TYPE_NAME = varchar

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

SQL Server 2000 Only

Table B-6. GetTypeInfo for SQL Server 2000 Only

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	

Table B-6. GetTypeInfo for SQL Server 2000 Only (cont.)

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	

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	

Sybase Driver

Table B-7. GetTypeInfo for Sybase

TYPE_NAME = binary

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = length
DATA_TYPE = -2 (BINARY)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = 0x
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = binary
MAXIMUM_SCALE = NULL
MINIMUM_SCALE = NULL
NULLABLE = 1
NUM_PREC_RADIX = NULL
PRECISION = 2048
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = bit

AUTO_INCREMENT = NULL
CASE_SENSITIVE = false
CREATE_PARAMS = NULL
DATA_TYPE = -7 (BIT)
FIXED_PREC_SCALE = false
LITERAL_PREFIX = NULL
LITERAL_SUFFIX = NULL
LOCAL_TYPE_NAME = bit
MAXIMUM_SCALE = 0
MINIMUM_SCALE = 0
NULLABLE = 0
NUM_PREC_RADIX = NULL
PRECISION = 1
SEARCHABLE = 2
SQL_DATA_TYPE = NULL
SQL_DATETIME_SUB = NULL
UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = char

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

Table B-7. GetTypeInfo for Sybase (cont.)

TYPE_NAME = datetime

AUTO_INCREMENT = NULL	MINIMUM_SCALE = 6
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 = 6	

TYPE_NAME = decimal

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

TYPE_NAME = float

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

***Table B-7. GetTypeInfo for Sybase** (cont.)*

TYPE_NAME = image

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

TYPE_NAME = int

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

TYPE_NAME = money

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

***Table B-7. GetTypeInfo for Sybase** (cont.)*

TYPE_NAME = nchar

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

TYPE_NAME = numeric

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

TYPE_NAME = nvarchar

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

Table B-7. GetTypeInfo for Sybase *(cont.)*

TYPE_NAME = real

AUTO_INCREMENT = false	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = NULL	NUM_PREC_RADIX = 10
DATA_TYPE = 7 (REAL)	PRECISION = 7
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = NULL	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = real	UNSIGNED_ATTRIBUTE = false
MAXIMUM_SCALE = NULL	

TYPE_NAME = smalldatetime

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

TYPE_NAME = smallint

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

Table B-7. GetTypeInfo for Sybase (cont.)

TYPE_NAME = smallmoney

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

TYPE_NAME = sysname

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

TYPE_NAME = text

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

***Table B-7. GetTypeInfo for Sybase** (cont.)*

TYPE_NAME = timestamp

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

TYPE_NAME = tinyint

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

TYPE_NAME = unichar

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

Table B-7. GetTypeInfo for Sybase (cont.)

TYPE_NAME = univarchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 2048
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = univarchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = varbinary	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = false	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = -3 (VARBINARY)	PRECISION = 2048
FIXED_PREC_SCALE = false	SEARCHABLE = 2
LITERAL_PREFIX = 0x	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = NULL	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varbinary	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	
TYPE_NAME = varchar	
AUTO_INCREMENT = NULL	MINIMUM_SCALE = NULL
CASE_SENSITIVE = true	NULLABLE = 1
CREATE_PARAMS = max length	NUM_PREC_RADIX = NULL
DATA_TYPE = 12 (VARCHAR)	PRECISION = 2048
FIXED_PREC_SCALE = false	SEARCHABLE = 3
LITERAL_PREFIX = '	SQL_DATA_TYPE = NULL
LITERAL_SUFFIX = '	SQL_DATETIME_SUB = NULL
LOCAL_TYPE_NAME = varchar	UNSIGNED_ATTRIBUTE = NULL
MAXIMUM_SCALE = NULL	

C Designing JDBC Applications for Performance Optimization

Developing performance-oriented JDBC applications is not easy. JDBC drivers do not throw exceptions to say that your code is running too slow.

These guidelines were compiled by examining the JDBC implementations of numerous shipping JDBC applications. The guidelines discuss using database metadata materials, retrieving data, selecting JDBC objects and methods, designing JDBC applications, and updating data.

The following table summarizes some common JDBC system performance problems and suggests some possible solutions.

Problem	Solution	See guidelines in…
Network communication is slow	Reduce network traffic	"Using Database Metadata Methods" on page 176
Evaluation of complex SQL queries on the database server is slow and might reduce concurrency	Simplify queries	"Using Database Metadata Methods" on page 176 "Selecting JDBC Objects and Methods" on page 181
Excessive calls from the application to the driver decrease performance	Optimize application-to-driver interaction	"Retrieving Data" on page 179 "Selecting JDBC Objects and Methods" on page 181
Disk input/output is slow	Limit disk input/output	"Designing JDBC Applications" on page 185

Using Database Metadata Methods

Because database metadata methods that generate Resultset objects are slow compared to other JDBC methods, their frequent use can impair system performance. The guidelines in this section will help you to optimize system performance when selecting and using database metadata.

Minimizing the Use of Database Metadata Methods

Compared to other JDBC methods, database metadata methods that generate Resultset objects are relatively slow. Applications should cache information returned from result sets that generate database metadata methods so that multiple executions are not needed.

While almost no JDBC application can be written without database metadata methods, you can improve system performance by minimizing their use. To return all result column information *mandated* by the JDBC specification, a JDBC driver may have to perform complex queries or multiple queries to return the necessary result set for a single call to a database metadata method. These particular elements of the SQL language are performance-expensive.

Applications should cache information from database metadata methods. For example, call getTypeInfo once in the application and cache away the elements of the result set that your application depends on. It is unlikely that any application uses all elements of the result set generated by a database metadata method, so the cache of information should not be difficult to maintain.

Avoiding Search Patterns

Using null arguments or search patterns in database metadata methods results in generating time-consuming queries. In addition, network traffic potentially increases due to unwanted results. Always supply as many non-null arguments to result sets that generate database metadata methods as possible.

Because database metadata methods are slow, applications should invoke them as efficiently as possible. Many applications pass the fewest non-null arguments necessary for the function to return success.

For example:

```
ResultSet WSrs = WSc.getTables (null, null, "WSTable", null);
```

should be:

```
ResultSet WSrs = WSc.getTables ("cat1", "johng", "WSTable", "TABLE");
```

Sometimes, little information is known about the object for which you are requesting information. Any information that the application can send the driver when calling database metadata methods can result in improved performance and reliability.

Using a Dummy Query to Determine Table Characteristics

Avoid using getColumns to determine characteristics about a table. Instead, use a dummy query with getMetadata.

Consider an application that allows the user to choose the columns that will be selected. Should the application use getColumns to return information about the columns to the user or instead prepare a dummy query and call getMetadata?

Case 1: GetColumns Method

```
ResultSet WSrc = WSc.getColumns (... "UnknownTable" ...);
// This call to getColumns will generate a query to
// the system catalogs... possibly a join
// which must be prepared, executed, and produce
// a result set
. . .
WSrc.next();
string Cname = getString(4);
. . .
// user must retrieve N rows from the server
// N = # result columns of UnknownTable
// result column information has now been obtained
```

Case 2: GetMetadata Method

```
// prepare dummy query
PreparedStatement WSps = WSc.prepareStatement
    (... "SELECT * from UnknownTable WHERE 1 = 0" ...);
// query is never executed on the server -
// only prepared
ResultSetMetaData WSsmd=wsps.getMetaData();
int numcols = WSrsmd.getColumnCount();
...
int ctype = WSrsmd.getColumnType(n)
...
// result column information has now been obtained
// Note we also know the column ordering within the
// table!  This information cannot be
// assumed from the getColumns example.
```

In both cases, a query is sent to the server, but in Case 1 the query must be evaluated and form a result set that must be sent to the client. Clearly, Case 2 is the better performing model.

To somewhat complicate this discussion, let us consider a DBMS server that does not natively support preparing a SQL statement. The performance of Case 1 does not change but Case 2 increases minutely because the dummy query must be evaluated instead of only prepared. Because the Where clause of the query always evaluates to FALSE, the query generates no result rows and should execute without accessing table data. For this situation, method 2 still outperforms method 1.

Retrieving Data

To retrieve data efficiently, return only the data that you need, and choose the most efficient method of doing so. The guidelines in this section will help you to optimize system performance when retrieving data with JDBC applications.

Retrieving Long Data

Unless it is necessary, applications should not request long data because retrieving long data across a network is slow and resource-intensive.

Most users don't want to see long data. If the user does want to see these result items, then the application can query the database again, specifying only the long columns in the select list. This method allows the average user to retrieve the result set without having to pay a high performance penalty for network traffic.

Although the best method is to exclude long data from the select list, some applications do not formulate the select list before sending the query to the JDBC driver (that is, some applications `select * from <table name> ...`). If the select list contains long data, then some drivers must retrieve that data at fetch time even if the application does not bind the long data in the result set. When possible, the designer should attempt to implement a method that does not retrieve all columns of the table.

Additionally, although the getClob and getBlob methods allow the application to control how long data is retrieved in the application, the designer must realize that in many cases, the JDBC driver emulates these methods due to the lack of true locator support in the DBMS. In such cases, the driver must retrieve all of the long data across the network before exposing the getClob and getBlob methods.

Sometimes long data must be retrieved. When this is the case, remember that most users do not want to see 100 KB, or more, of text on the screen.

Reducing the Size of Data Retrieved

To reduce network traffic and improve performance, you can reduce the size of any data being retrieved to some manageable limit by calling setMaxRows, setMaxFieldSize, and the driver-specific SetFetchSize. Another method of reducing the size of the data being retrieved is to decrease the column size. If the driver allows you to define the packet size, use the smallest packet size that will meet your needs.

In addition, be careful to return only the rows you need. If you return five columns when you only need two columns, performance is decreased, especially if the unnecessary rows include long data.

Choosing the Right Data Type

Retrieving and sending certain data types can be expensive. When you design a schema, select the data type that can be processed most efficiently. For example, integer data is processed faster than floating-point data. Floating-point data is defined according to internal database-specific formats, usually in a compressed format. The data must be decompressed and converted into a different format so that it can be processed by the wire protocol.

Selecting JDBC Objects and Methods

The guidelines in this section will help you to optimize system performance when selecting and using JDBC objects and methods.

Using Parameter Markers as Arguments to Stored Procedures

When calling stored procedures, always use parameter markers for the argument markers instead of using literal arguments. JDBC drivers can call stored procedures on the database server either by executing the procedure as any other SQL query, or by optimizing the execution by invoking a Remote Procedure Call (RPC) directly into the database server. Executing the stored procedure as a SQL query results in the database server parsing the statement, validating the argument types, and converting the arguments into the correct data types. Remember that SQL is always sent to the database server as a character string, for example, "{call getCustName (12345)}". In this case, even though the application programmer might assume that the only argument to getCustName is an integer, the argument is actually passed inside a character string to the server. The database server would parse the SQL query, isolate the single argument value 12345, then convert the string '12345' into an integer value.

By invoking an RPC inside the database server, the overhead of using a SQL character string is avoided. Instead, the procedure is called only by name with the argument values already encoded into their native data types.

Case 1

Stored Procedure cannot be optimized to use a server-side RPC. The database server must parse the statement, validate the argument types, and convert the arguments into the correct data types.

```
CallableStatement cstmt = conn.prepareCall ("call getCustName (12345)");
ResultSet rs = cstmt.executeQuery ();
```

Case 2

Stored Procedure can be optimized to use a server-side RPC. Because the application calls the procedure by name and the argument values are already encoded, the load on the database server is less.

```
CallableStatement cstmt = conn.prepareCall ("Call getCustName (?)");
cstmt.setLong (1,12345);
ResultSet rs = cstmt.executeQuery();
```

Using the Statement Object Instead of the PreparedStatement Object

JDBC drivers are optimized based on the perceived use of the functions that are being executed. Choose between the PreparedStatement object and the Statement object depending on the planned use. The Statement object is optimized for a single execution of a SQL statement. In contrast, the PreparedStatement object is optimized for SQL statements that will be executed two or more times.

The overhead for the initial execution of a PreparedStatement object is high. The advantage comes with subsequent executions of the SQL statement.

Choosing the Right Cursor

Choosing the appropriate type of cursor allows maximum application flexibility. This section summarizes the performance issues of three types of cursors.

A forward-only cursor provides excellent performance for sequential reads of all of the rows in a table. However, it cannot be used when the rows to be returned are not sequential.

Insensitive cursors used by JDBC drivers are ideal for applications that require high levels of concurrency on the database server and require the ability to scroll forwards and backwards through result sets. The first request to an insensitive cursor fetches all of the rows and stores them on the client. Thus, the first request is very slow, especially when long data is retrieved. Subsequent requests do not require any network traffic and are processed quickly. Because the first request is processed slowly, insensitive cursors should not be used for a single request of one row. Designers should also avoid using insensitive cursors when long data is returned, because memory can be exhausted. Some insensitive cursor implementations cache the data in a temporary table on the database server and avoid the performance issue.

Sensitive cursors, sometimes called keyset-driven cursors, use identifiers, such as a ROWID, that already exist in your database. When you scroll through the result set, the data for the identifiers is retrieved. Because each request generates network traffic, performance can be very slow. However, returning nonsequential rows does not further affect performance. Sensitive cursors are the preferred scrollable cursor model for dynamic situations, when the application cannot afford to buffer the data from an insensitive cursor.

Using get Methods Effectively

JDBC provides a variety of methods to retrieve data from a result set, such as getInt, getString, and getObject. The getObject method is the most generic and provides the worst performance when the non-default mappings are specified. This is because the JDBC driver must do extra processing to determine the type of the value being retrieved and generate the appropriate mapping. Always use the specific method for the data type.

To further improve performance, provide the column number of the column being retrieved, for example, getString(1), getLong(2), and getInt(3), instead of the column name. If the column names are not specified, network traffic is unaffected, but costly conversions and lookups increase. For example, suppose you use:

```
getString("foo")...
```

The JDBC driver may have to convert "foo" to uppercase (if necessary), and then compare "foo" with all the columns in the column list. That is costly. If, instead, the driver was able to go directly to result column 23, then a lot of processing would be saved.

For example, suppose you have a result set that has 15 columns and 100 rows, and the column names are not included in the result set. You are interested in three columns, EMPLOYEENAME (a string), EMPLOYEENUMBER (a long integer), and SALARY (an integer). If you specify getString("EmployeeName"), getLong("EmployeeNumber"), and getInt("Salary"), each column name must be converted to uppercase, and lookups would increase considerably. Performance would improve significantly if you specify getString(1), getLong(2), and getInt(15).

Designing JDBC Applications

The guidelines in this section will help you to optimize system performance when designing JDBC applications.

Managing Connections

Connection management is important to application performance. Optimize your application by connecting once and using multiple statement objects, instead of performing multiple connections. Avoid connecting to a data source after establishing an initial connection.

Although gathering driver information at connect time is a good practice, it is often more efficient to gather it in one step rather than two steps. For example, some applications establish a connection and then call a method in a separate component that reattaches and gathers information about the driver. Applications that are designed as separate entities should pass the established connection object to the data collection routine instead of establishing a second connection.

Another bad practice is to connect and disconnect several times throughout your application to perform SQL statements. Connection objects can have multiple statement objects associated with them. Statement objects, which are defined to be memory storage for information about SQL statements, can manage multiple SQL statements.

You can improve performance significantly with connection pooling, especially for applications that connect over a network or through the World Wide Web. Connection pooling lets you reuse connections. Closing connections does not close the physical connection to the database. When an application requests a connection, an active connection is reused, thus avoiding the network I/O needed to create a new connection.

Connection and statement handling should be addressed before implementation. Spending time and thoughtfully handling connection management improves application performance and maintainability.

Managing Commits in Transactions

Committing transactions is extremely disk I/O intensive and slow. Always turn off autocommit by using the following setting: `WSConnection.setAutoCommit(false)`.

What does a commit actually involve? The database server must flush back to disk every data page that contains updated or new data. This is not a sequential write but a searched write to replace existing data in the table. By default, Autocommit is on when connecting to a data source, and Autocommit mode usually impairs performance because of the significant amount of disk I/O needed to commit every operation.

Furthermore, some database servers do not provide an Autocommit mode. For this type of server, the JDBC driver must explicitly issue a COMMIT statement and a BEGIN TRANSACTION for every operation sent to the server. In addition to the large amount of disk input/output required to support Autocommit mode, a performance penalty is paid for up to three network requests for every statement issued by an application.

Although using transactions can help application performance, do not take this tip too far. Leaving transactions active can reduce throughput by holding locks on rows for long times, preventing other users from accessing the rows. Commit transactions in intervals that allow maximum concurrency.

Choosing the Right Transaction Model

Many systems support distributed transactions; that is, transactions that span multiple connections. Distributed transactions are at least four times slower than normal transactions due to the logging and network I/O necessary to communicate between all the components involved in the distributed transaction. Unless distributed transactions are required, avoid using them. Instead, use local transactions whenever possible.

For the best system performance, design the application to run under a single Connection object.

Updating Data

This section provides general guidelines to help you to optimize system performance when updating data in databases.

Using updateXXX Methods

Although programmatic updates do not apply to all types of applications, developers should attempt to use programmatic updates and deletes. Using the updateXXX methods of the ResultSet object allows the developer to update data without building a complex SQL statement. Instead, the developer simply supplies the column in the result set that is to be updated and the data that is to be changed. Then, before moving the cursor from the row in the result set, the updateRow method must be called to update the database as well.

In the following code fragment, the value of the Age column of the Resultset object rs is retrieved using the method getInt, and the method updateInt is used to update the column with an int value of 25. The method updateRow is called to update the row in the database that contains the modified value.

```
int n = rs.getInt("Age");
// n contains value of Age column in the resultset rs
. . .
rs.updateInt("Age", 25);
rs.updateRow();
```

In addition to making the application more easily maintainable, programmatic updates usually result in improved performance. Because the database server is already positioned on the row for the Select statement in process, performance-expensive operations to locate the row to be changed are not needed. If the row must be located, the server usually has an internal pointer to the row available (for example, ROWID).

Using getBestRowIndentifier()

Use getBestRowIndentifier() to determine the optimal set of columns to use in the Where clause for updating data. Pseudo-columns often provide the fastest access to the data, and these columns can only be determined by using getBestRowIndentifier().

Some applications cannot be designed to take advantage of positional updates and deletes. Some applications might formulate the Where clause by using all searchable result columns by calling getPrimaryKeys(), or by calling getIndexInfo() to find columns that might be part of a unique index. These methods usually work, but might result in fairly complex queries.

Consider the following example:

```
ResultSet WSrs = WSs.executeQuery
     ("SELECT first_name, last_name, ssn, address, city, state, zip
        FROM emp");
// fetchdata
...
WSs.executeQuery ("UPDATE EMP SET ADDRESS = ?
     WHERE first_name = ? and last_name = ? and ssn = ?
     and address = ? and city = ? and state = ?
     and zip = ?");
// fairly complex query
```

Applications should call getBestRowIndentifier() to retrieve the optimal set of columns (possibly a pseudo-column) that identifies a specific record. Many databases support special columns that are not explicitly defined by the user in the table definition but are "hidden" columns of every table (for example, ROWID and TID). These pseudo-columns generally provide the fastest access to the data because they typically are pointers to the exact location of the record. Because pseudo-columns are not part of the explicit table definition, they are not returned from getColumns. To determine if pseudo-columns exist, call getBestRowIndentifier().

Consider the previous example again:

```
...
ResultSet WSrowid = getBestRowIndentifier()
   (.... "emp", ...);
...
WSs.executeQuery ("UPDATE EMP SET ADDRESS = ?
     WHERE first_name = ? and last_name = ? and ssn = ?
     and address = ? and city = ? and state = ?
     and zip = ?");
// fastest access to the data!
```

If your data source does not contain special pseudo-columns, then the result set of getBestRowIndentifier() consists of the columns of the most optimal unique index on the specified table (if a unique index exists). Therefore, your application does not need to call getIndexInfo to find the smallest unique index.

Conclusion

With thoughtful design and implementation, the performance of JDBC applications can be improved. By the appropriate use of DatabaseMetaData methods, retrieving only required data, selecting functions that optimize performance, and managing connections and updates, your applications can run more efficiently and generate less network traffic.

D SQL Escape Sequences for JDBC

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

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

The escape sequence used by JDBC is:

```
{extension}
```

The escape sequence is recognized and parsed by the DataDirect Connect *for* JDBC driver, which replaces the escape sequences with data store-specific grammar.

Date, Time, and Timestamp Escape Sequences

The escape sequence for date, time, and timestamp literals is:

```
{literal-type 'value'}
```

where *literal-type* is one of the following:

literal-type	Description	Value Format
d	Date	`yyyy-mm-dd`
t	Time	`hh:mm:ss [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 DataDirect Connect *for* JDBC drivers, as listed in Table D-1.

Example:

```
SELECT {fn UCASE(NAME)} FROM EMP
```

Table D-1. Scalar Functions Supported

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

***Table D-1. Scalar Functions Supported** (cont.)*

Data Store	String Functions	Numeric Functions	Timedate Functions	System Functions
Informix	CONCAT	ABS	CURDATE	DATABASE
	LEFT	ACOS	CURTIME	USER
	LENGTH	ASIN	DAYOFMONTH	
	LTRIM	ATAN	DAYOFWEEK	
	REPLACE	ATAN2	MONTH	
	RTRIM	COS	NOW	
	SUBSTRING	COT	TIMESTAMPADD	
		EXP	TIMESTAMPDIFF	
		FLOOR	YEAR	
		LOG		
		LOG10		
		MOD		
		PI		
		POWER		
		ROUND		
		SIN		
		SQRT		
		TAN		
		TRUNCATE		

***Table D-1. Scalar Functions Supported** (cont.)*

Data Store	String Functions	Numeric Functions	Timedate Functions	System Functions
Oracle	ASCII	ABS	CURDATE	IFNULL
	BIT_LENGTH	ACOS	DAYNAME	USER
	CHAR	ASIN	DAYOFMONTH	
	CONCAT	ATAN	DAYOFWEEK	
	INSERT	ATAN2	DAYOFYEAR	
	LCASE	CEILING	HOUR	
	LEFT	COS	MINUTE	
	LENGTH	COT	MONTH	
	LOCATE	EXP	MONTHNAME	
	LOCATE2	FLOOR	NOW	
	LTRIM	LOG	QUARTER	
	OCTET_LENGTH	LOG10	SECOND	
	REPEAT	MOD	WEEK	
	REPLACE	PI	YEAR	
	RIGHT	POWER		
	RTRIM	ROUND		
	SOUNDEX	SIGN		
	SPACE	SIN		
	SUBSTRING	SQRT		
	UCASE	TAN		
		TRUNCATE		

Table D-1. Scalar Functions Supported *(cont.)*

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

***Table D-1. Scalar Functions Supported** (cont.)*

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

Outer Join Escape Sequences

JDBC supports the SQL92 left, right, and full outer join syntax. The escape sequence for outer joins is:

```
{oj outer-join}
```

where *outer-join* is:

```
table-reference {LEFT | RIGHT | FULL} OUTER JOIN
{table-reference | outer-join} ON search-condition
```

where:

table-reference is a 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 D-2 lists the outer join escape sequences supported by DataDirect Connect *for* JDBC for each data store.

Table D-2. Outer Join Escape Sequences Supported

Data Store	Outer Join Escape Sequences
DB2	Left outer joins Right outer joins Nested outer joins
Informix	Left outer joins Right outer joins Nested outer joins

***Table D-2. Outer Join Escape Sequences Supported** (cont.)*

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

Procedure Call Escape Sequences

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

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

where:

procedure-name specifies the name of a stored procedure.

parameter specifies a stored procedure parameter.

NOTE: For DB2, a schema name cannot be used when calling a procedure. Also, literal parameter values are not supported.

E Using DataDirect Test

This chapter provides information about DataDirect Test, a software component that allows you to test and learn the JDBC API, and contains a tutorial that takes you through a working example of its use.

DataDirect Test contains menu selections that correspond to specific JDBC functions—for example, connecting to a database or passing a SQL statement. It allows you to:

- Execute a single JDBC method or execute multiple JDBC methods simultaneously, so that you can easily perform some common tasks, such as returning result sets
- Display the results of all JDBC function calls in one window, while displaying fully commented, Java JDBC code in an alternate window

DataDirect Test Tutorial

This DataDirect Test tutorial explains how to use the most important features of DataDirect Test (and the JDBC API) and assumes that you can connect to a database with the standard available demo table or fine-tune the sample SQL statements shown in this example as appropriate for your environment.

NOTE: The step-by-step examples used in this tutorial do not show typical clean-up routines (for example, closing result sets and connections). These steps have been omitted to simplify the examples. Do not forget to add these steps when you use equivalent code in your applications.

Configuring DataDirect Test

The default DataDirect Test configuration file is:

install_dir/testforjdbc/Config.txt

where *install_dir* is your DataDirect Connect *for* JDBC installation directory. This file can be edited as appropriate for your environment using any text editor. All parameters are configurable, but the most commonly configured parameters are:

Drivers	A list of colon-separated JDBC driver classes.
DefaultDriver	The default JDBC driver that appears in the Get Driver URL window.
Databases	A list of comma-separated JDBC URLs. The first item in the list appears as the default in the database selection window. You can use one of these URLs as a template when you make a JDBC connection. The default Config.txt file contains example URLs for most databases.
InitialContextFactory	Set to com.sun.jndi.fscontext.RefFSContextFactory if you are using file system data sources, or com.sun.jndi.ldap.LdapCtxFactory if you are using LDAP.
ContextProviderURL	The location of the .bindings file if you are using file system data sources, or your LDAP Provider URL if you are using LDAP.
Datasources	A list of comma-separated JDBC data sources. The first item in the list appears as the default in the data source selection window.

Starting DataDirect Test

How you start DataDirect Test depends on your platform:

- **As a Java application on Windows**—Run the TestForJDBC.bat file located in the testforjdbc directory.

  On Windows 9*x* and Me, double-clicking TestForJDBC.bat opens a DOS window and displays the error "Out of environment space." To prevent this, use the following procedure:

  a After installing DataDirect Connect *for* JDBC, locate the TestForJDBC.bat file in the testforjdbc directory beneath the installation directory.

  b Right-click TestForJDBC.bat and select Properties. After the properties display, select the Memory tab.

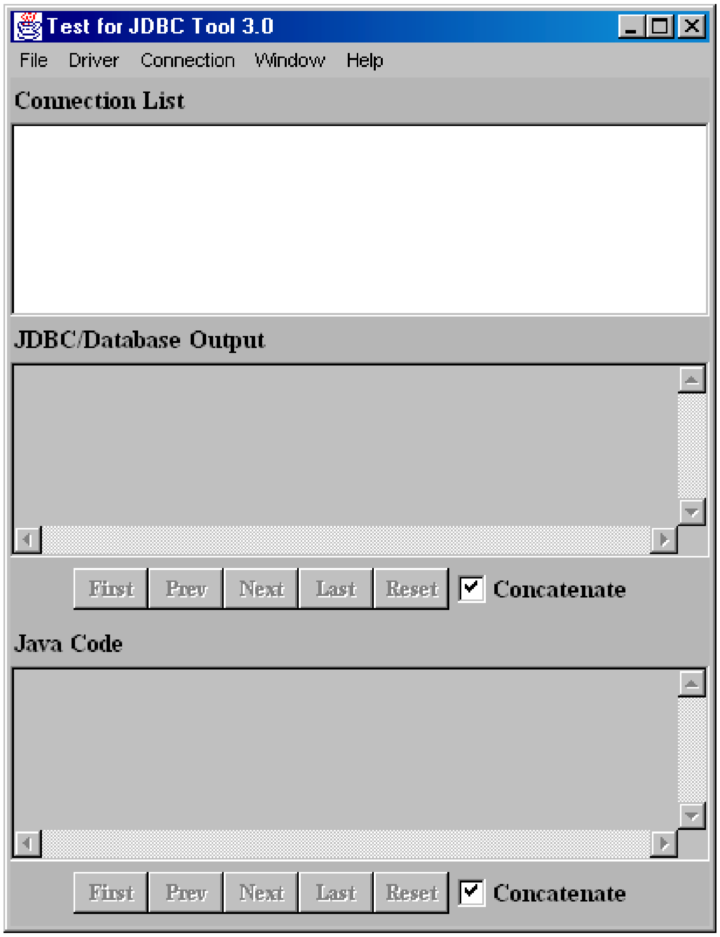
  c On the Memory tab, locate the Initial environment setting. From this drop-down list, select 1024. Then, select the Protected check box. Click **OK**. A testforjdbc shortcut is created in the same directory with TestForJDBC.bat.

  d Double-click either TestForJDBC.bat or the testforjdbc shortcut. DataDirect Test will open normally without generating the error.

  NOTE: Do not delete the testforjdbc shortcut; the 1024 environment setting will be lost if the shortcut is deleted.

- **As a Java application on UNIX**—Run the TestForJDBC.sh shell script located in the testforjdbc directory beneath the installation directory.

After you start DataDirect Test, the Test for JDBC Tool window appears.



The main Test for JDBC Tool window shows the following information:

- In the Connection List box, a list of available connections.
- In the JDBC/Database scroll box, a report indicating whether the last action succeeded or failed.
- In the Java Code scroll box, the actual Java code used to implement the last action.

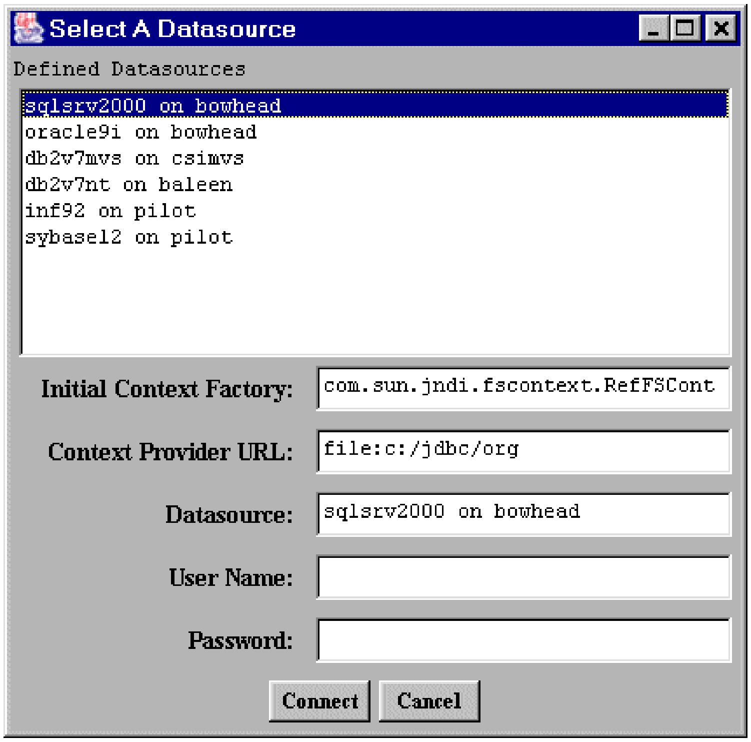
TIP: DataDirect Test windows contain two Concatenate check boxes. Select a Concatenate check box to see a cumulative record of previous actions; otherwise, only the last action is shown. Selecting Concatenate can degrade performance, particularly when displaying large resultSets.

Connecting Using DataDirect Test

There are two methods to connect using DataDirect Test: through a data source or through driver/database selection.

Connecting through a Data Source

1 From the main Test for JDBC Tool window menu, select **Connection / Connect to DB via Data Source**. DataDirect Test displays the Select A Datasource window.

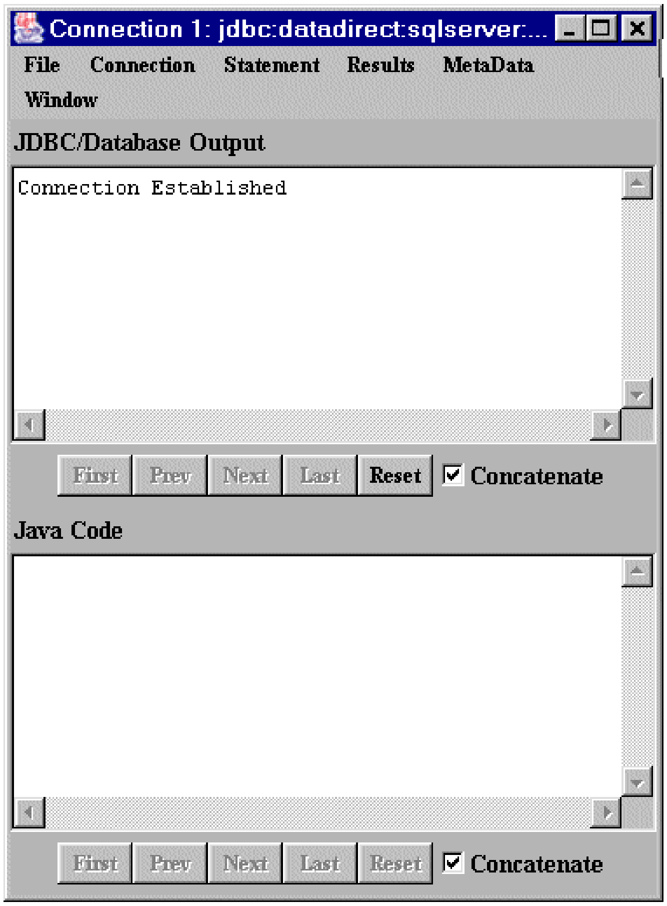


```
sqlsrv2000 on bowhead
oracle9i on bowhead
db2v7mvs on csimvs
db2v7nt on baleen
inf92 on pilot
sybase12 on pilot
```

Initial Context Factory:	com.sun.jndi.fscontext.RefFSCont
Context Provider URL:	file:c:/jdbc/org
Datasource:	sqlsrv2000 on bowhead
User Name:	
Password:	

2 Select a data source from the Defined Datasources pane. In the User Name and Password fields, type the required user and password connection properties; then, click **Connect**. For information about JDBC connection properties, see the appropriate DataDirect Connect *for* JDBC driver chapter.

3 If the connection was successful, the Connection window appears and shows the `Connection Established` message in the JDBC/Database Output scroll box.



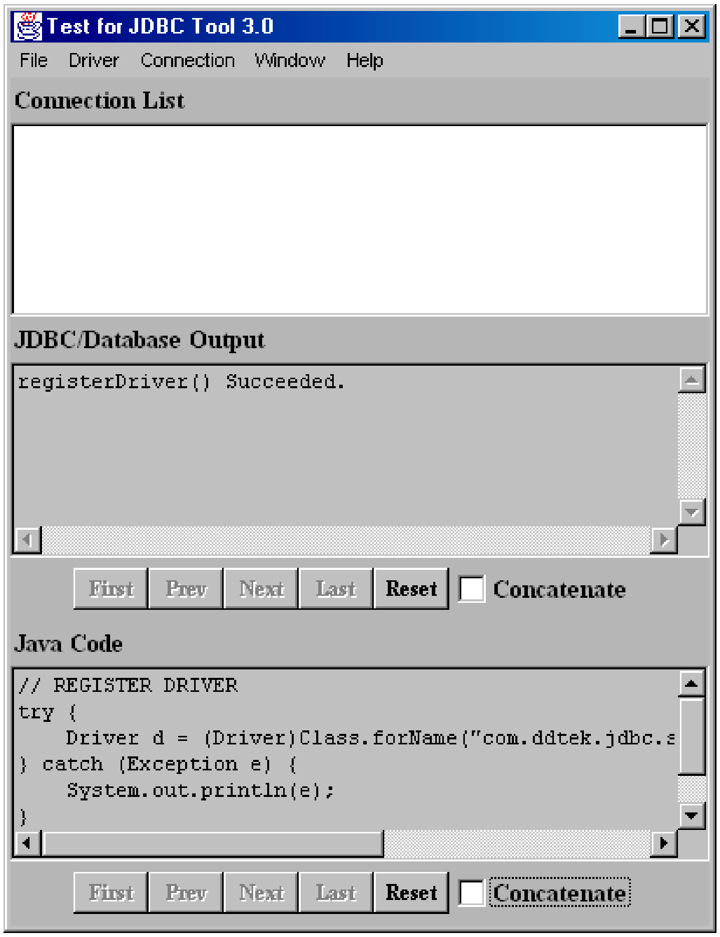
```
Connection Established
```

Connecting Through Driver/Database Selection

1 From the main Test for JDBC Tool window menu, select **Driver** / **Register Driver**. DataDirect Test prompts for a JDBC driver name.

2 In the Please Supply a Driver URL field, make sure that a driver is specified (for example `com.ddtek.jdbc.sqlserver.SQLServerDriver`); then, click **OK**.

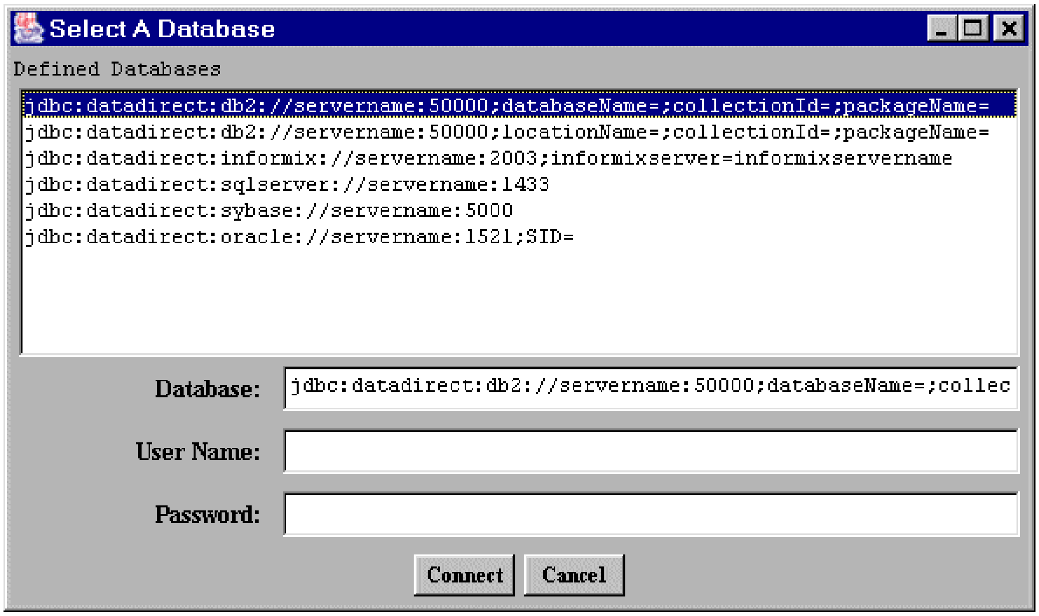
If the DataDirect Connect *for* JDBC Driver was registered successfully, the main Test for JDBC Tool window appears with a confirmation in the JDBC/Database Output scroll box.



```
registerDriver() Succeeded.
```

```
// REGISTER DRIVER
try {
    Driver d = (Driver)Class.forName("com.ddtek.jdbc.s
} catch (Exception e) {
    System.out.println(e);
}
```

3 Select **Connection / Connect to DB** from the main Test for JDBC Tool menu. JDBC prompts with a list of default connection URLs.



```
jdbc:datadirect:db2://servername:50000;databaseName=;collectionId=;packageName=
jdbc:datadirect:db2://servername:50000;locationName=;collectionId=;packageName=
jdbc:datadirect:informix://servername:2003;informixserver=informixservername
jdbc:datadirect:sqlserver://servername:1433
jdbc:datadirect:sybase://servername:5000
jdbc:datadirect:oracle://servername:1521;SID=
```

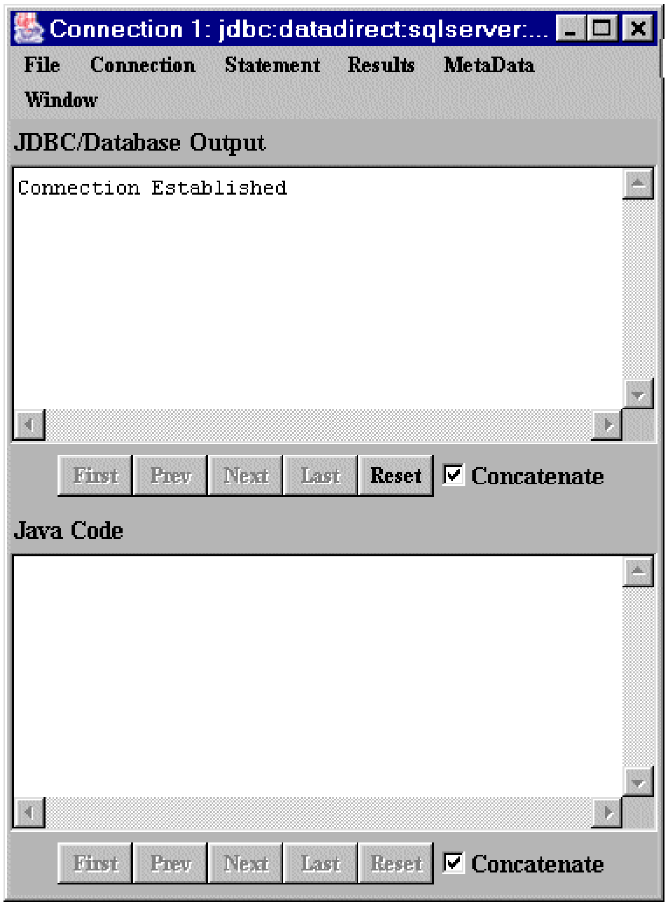
Database: jdbc:datadirect:db2://servername:50000;databaseName=;collec
User Name:
Password:

4 Select one of the default DataDirect Connect *for* JDBC driver connection URLs. In the Database field, modify the default values of the connection URL appropriately for your environment.

NOTE: There are two entries for DB2, one with locationName and one with databaseName. If you are connecting to OS/390, select the entry containing locationName. If you are connecting to UDB, select the entry containing databaseName.

5 In the User Name and Password fields, type the required user and password connection properties; then, click **Connect**. For information about JDBC connection properties, see the appropriate DataDirect Connect *for* JDBC driver chapter.

6 If the connection was successful, the Connection window appears and shows the `Connection Established` message in the JDBC/Database Output scroll box.



```
Connection Established
```

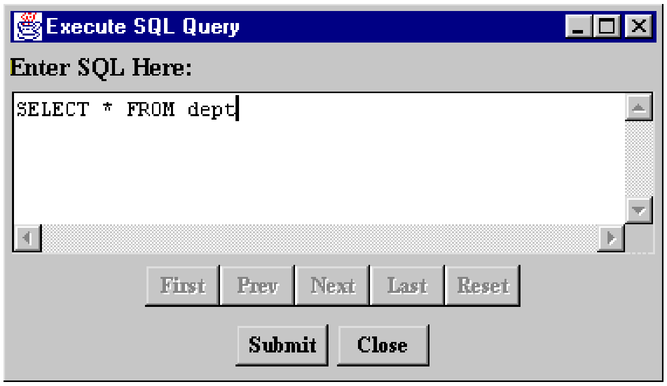
Executing a Simple Select Statement

This example explains how to execute a simple Select statement and retrieve the results.

1 From the Connection window menu, select **Connection / Create Statement**. The connection window indicates that the creation of the statement was successful.

2 Select **Statement / Execute Stmt Query**. DataDirect Test displays a dialog box that prompts for a SQL statement.

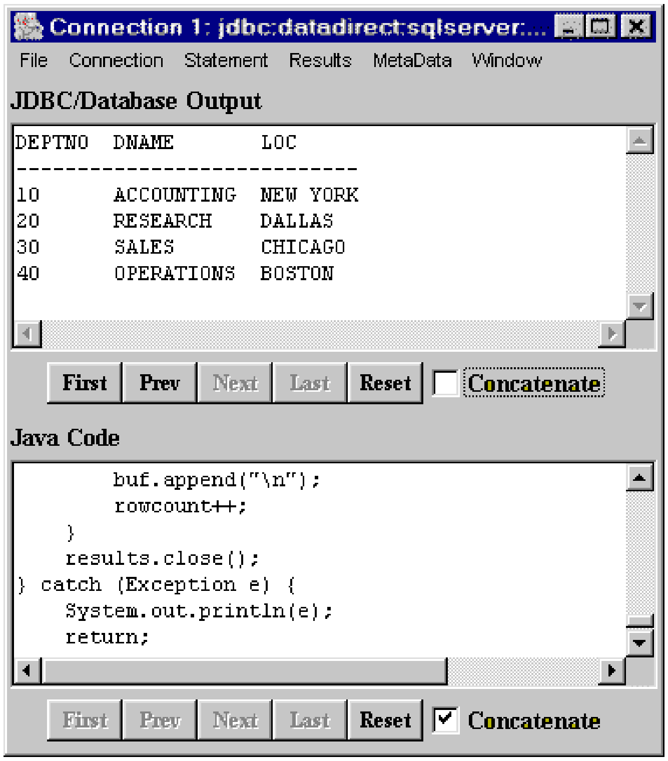
3 Specify the Select statement that you want to execute.



```
SELECT * FROM dept
```

Click **Submit**; then, click **Close**.

4 Select **Results / Show All Results**. The data from your result set is displayed.


Connection 1: jdbc:datadirect:sqlserver:...
File Connection Statement Results MetaData Window
JDBC/Database Output
First Prev Next Last Reset Concatenate
Java Code
First Prev Next Last Reset Concatenate

```
DEPTNO  DNAME       LOC
----------------------------
10      ACCOUNTING  NEW YORK
20      RESEARCH    DALLAS
30      SALES       CHICAGO
40      OPERATIONS  BOSTON
```

```
        buf.append("\n");
        rowcount++;
    }
    results.close();
} catch (Exception e) {
    System.out.println(e);
    return;
```

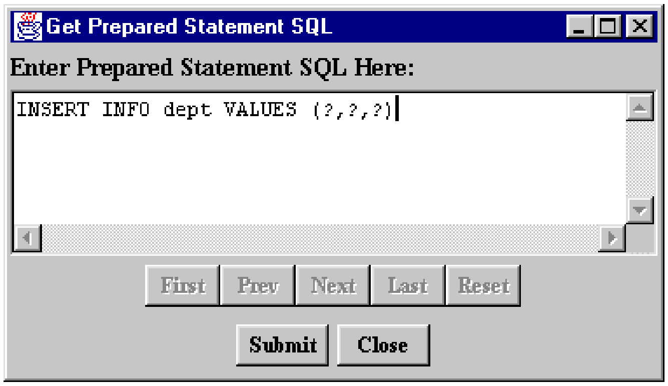
5 Scroll through the code in the Java Code scroll box to see which JDBC calls have been implemented by DataDirect Test.

Executing a Prepared Statement

This example explains how to execute a parameterized statement multiple times.

1 From the Connection window menu, select **Connection / Create Prepared Statement**. DataDirect Test prompts you for a SQL statement.

2 Specify the Insert statement that you want to execute.



Get Prepared Statement SQL
Enter Prepared Statement SQL Here:
First
Prev
Next
Last
Reset
Submit
Close

```
INSERT INFO dept VALUES (?,?,?)
```

Click **Submit**; then, click **Close**.

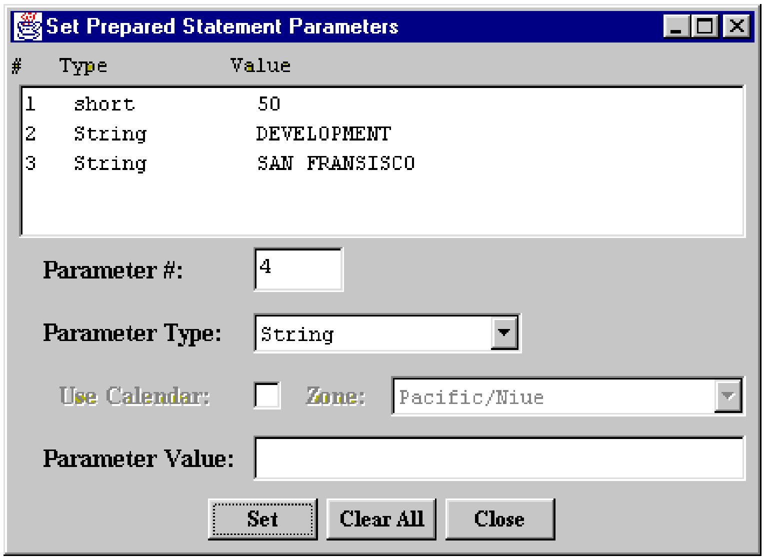
3 Select **Statement / Set Prepared Parameters**. To set the value and type for each parameter:

a Type the parameter number.

b Select the parameter type.

c Type the parameter value.

d Click **Set** to pass this information to the JDBC driver.



```
#    Type              Value
1    short             50
2    String            DEVELOPMENT
3    String            SAN FRANSISCO
```

Parameter #: 4

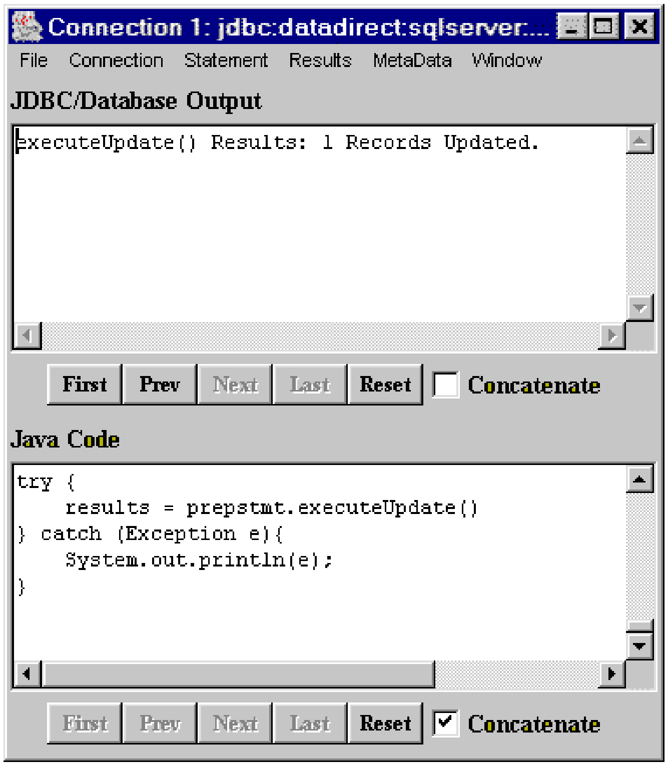
Parameter Type: String

Use Calendar: [ ] Zone: Pacific/Niue

Parameter Value:

4 When you are finished, click **Close**.

5 Select **Statement / Execute Stmt Update**. The JDBC/Database Output scroll box indicates that one row has been inserted.

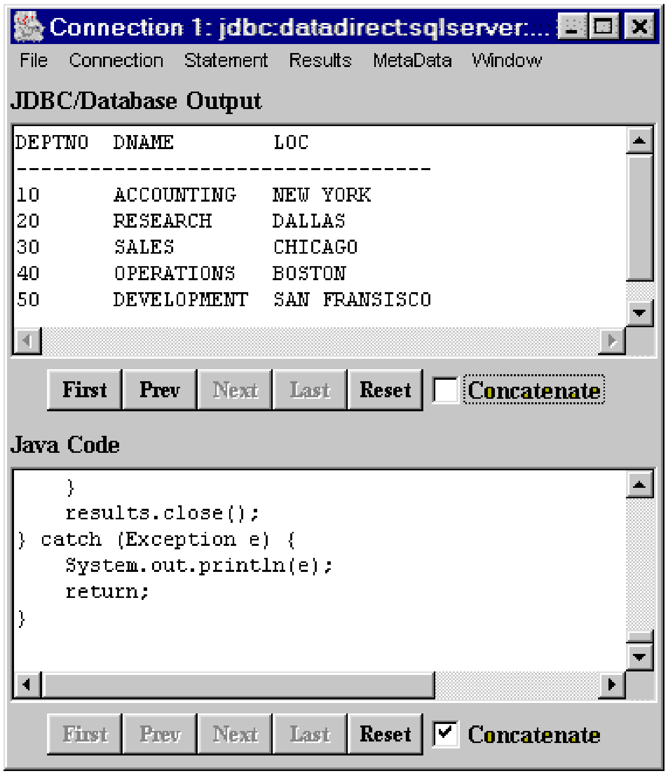

Connection 1: jdbc:datadirect:sqlserver:...
File Connection Statement Results MetaData Window
JDBC/Database Output
First Prev Next Last Reset Concatenate
Java Code
First Prev Next Last Reset Concatenate

```
executeUpdate() Results: 1 Records Updated.
```

```
try {
    results = prepstmt.executeUpdate()
} catch (Exception e){
    System.out.println(e);
}
```

6 If you want to insert multiple records, repeat Step 3 and Step 5 for each record.

7 If you repeat the steps described in "Executing a Simple Select Statement" on page 211, you will see that the previously inserted records are also returned.



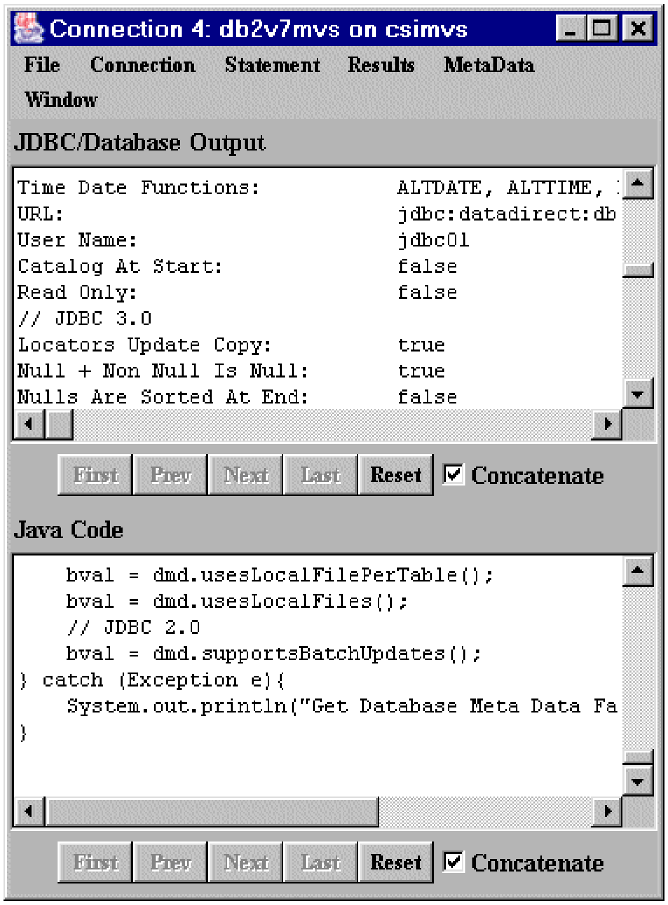
```
DEPTNO  DNAME         LOC
----------------------------------
10      ACCOUNTING    NEW YORK
20      RESEARCH      DALLAS
30      SALES         CHICAGO
40      OPERATIONS    BOSTON
50      DEVELOPMENT   SAN FRANSISCO
```

```
    }
    results.close();
} catch (Exception e) {
    System.out.println(e);
    return;
}
```

Retrieving Database Metadata

1 From the Connection window menu, select **Connection / Get DB Meta Data**.

2 Select **MetaData / Show Meta Data**. Information about the JDBC driver and the database to which you are connected is returned.



```
Time Date Functions:          ALTDATE, ALTTIME, 
URL:                          jdbc:datadirect:db
User Name:                    jdbc01
Catalog At Start:             false
Read Only:                    false
// JDBC 3.0
Locators Update Copy:         true
Null + Non Null Is Null:      true
Nulls Are Sorted At End:      false
```

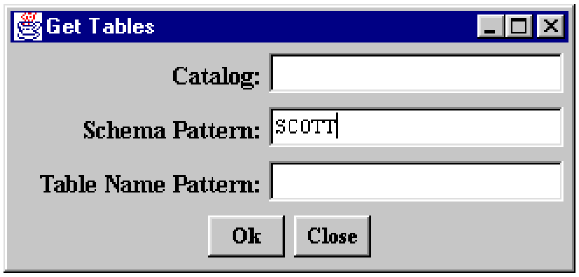
```
    bval = dmd.usesLocalFilePerTable();
    bval = dmd.usesLocalFiles();
    // JDBC 2.0
    bval = dmd.supportsBatchUpdates();
} catch (Exception e){
    System.out.println("Get Database Meta Data Fa
}
```

3 Scroll through the Java code in the Java Code scroll box to find out which JDBC calls have been implemented by DataDirect Test.

Metadata also allows you to query the database catalog (enumerate the tables in the database, for example). In this example, we will query all tables owned by the user SCOTT.

4 Select **MetaData** / **Tables**.

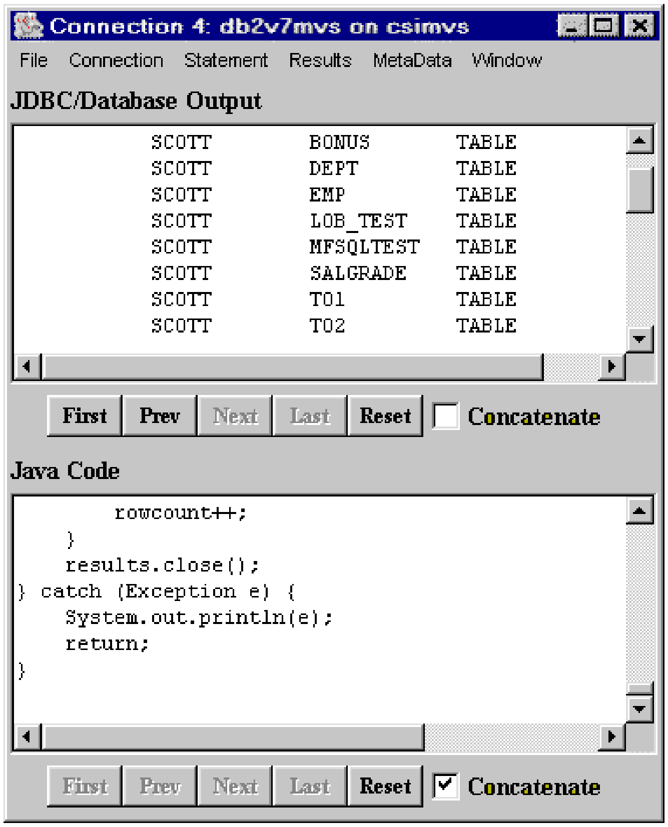
5 In the Schema Pattern field, type `SCOTT`.



Catalog:	
Schema Pattern:	SCOTT
Table Name Pattern:	

6 Click **Ok**. The Connection window indicates that getTables() succeeded.

7 Select **Results / Show All Results**. All tables owned by SCOTT are returned.



```
            SCOTT           BONUS          TABLE
            SCOTT           DEPT           TABLE
            SCOTT           EMP            TABLE
            SCOTT           LOB_TEST       TABLE
            SCOTT           MFSQLTEST      TABLE
            SCOTT           SALGRADE       TABLE
            SCOTT           T01            TABLE
            SCOTT           T02            TABLE
```

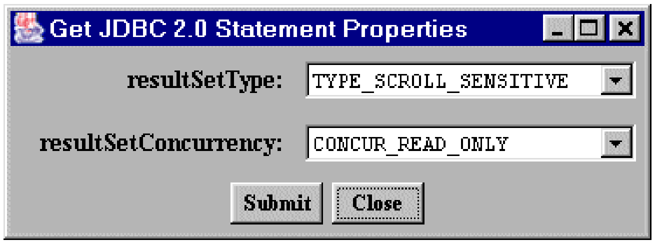
```
        rowcount++;
    }
    results.close();
} catch (Exception e) {
    System.out.println(e);
    return;
}
```

Scrolling Through a Result Set

NOTE: Scrollable result sets are supported by JDBC 2.0 and higher and require a Java 2 Platform (JDK 1.2)-compatible Java Virtual Machine.

1 From the Connection window menu, select **Connection / Create JDBC 2.0 Statement**. DataDirect Test prompts for a result set type and concurrency.

2 In the resultSetType field, select **TYPE_SCROLL_SENSITIVE**. In the resultSetConcurrency field, select **CONCUR_READ_ONLY**.

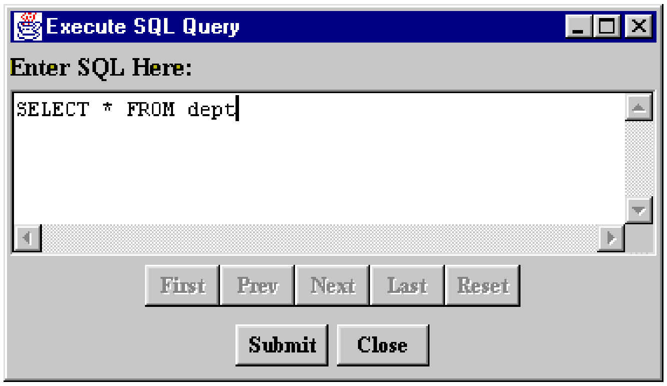


resultSetType: TYPE_SCROLL_SENSITIVE
resultSetConcurrency: CONCUR_READ_ONLY

Click **Submit**; then, click **Close**.

3 Select **Statement / Execute Stmt Query**.

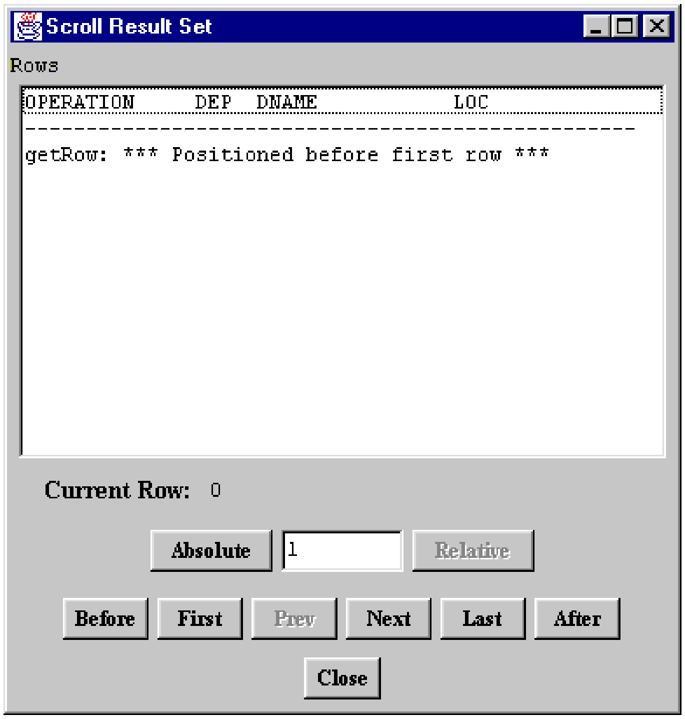
4 Specify the Select statement that you want to execute.



```
SELECT * FROM dept
```

Click **Submit**; then, click **Close**.

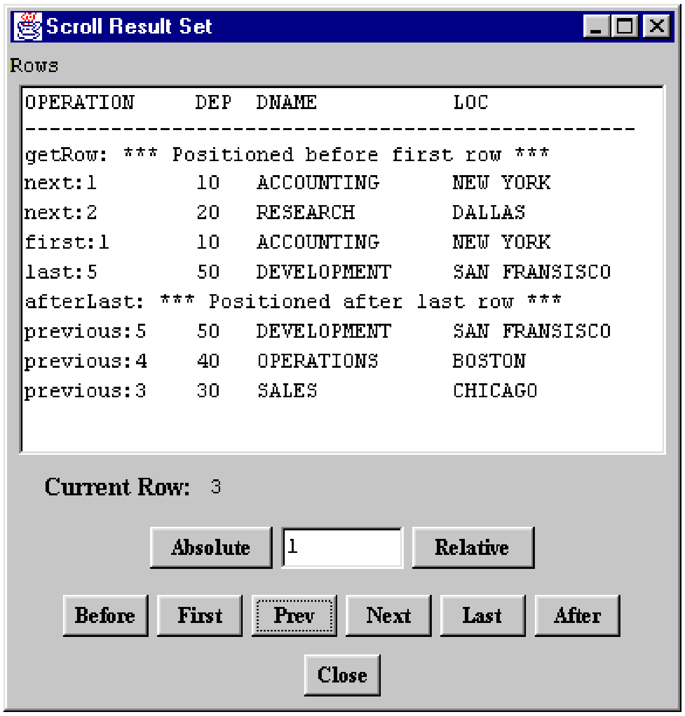
5 Select **Results / Scroll Results.** The Scroll Result Set window indicates that the cursor is positioned before the first row.



```
OPERATION      DEP  DNAME            LOC
------------------------------------------------------
getRow: *** Positioned before first row ***
```

Current Row: 0

6 Click the **Absolute, Relative, Before, First, Prev, Next, Last,** and **After** buttons as appropriate to navigate through the result set. After each action, the Scroll Result Set window displays the data at the current position of the cursor.



```
OPERATION     DEP  DNAME           LOC
--------------------------------------------------
getRow: *** Positioned before first row ***
next:1        10   ACCOUNTING      NEW YORK
next:2        20   RESEARCH        DALLAS
first:1       10   ACCOUNTING      NEW YORK
last:5        50   DEVELOPMENT     SAN FRANSISCO
afterLast: *** Positioned after last row ***
previous:5    50   DEVELOPMENT     SAN FRANSISCO
previous:4    40   OPERATIONS      BOSTON
previous:3    30   SALES           CHICAGO
```

Current Row: 3

1

7 Click **Close**.

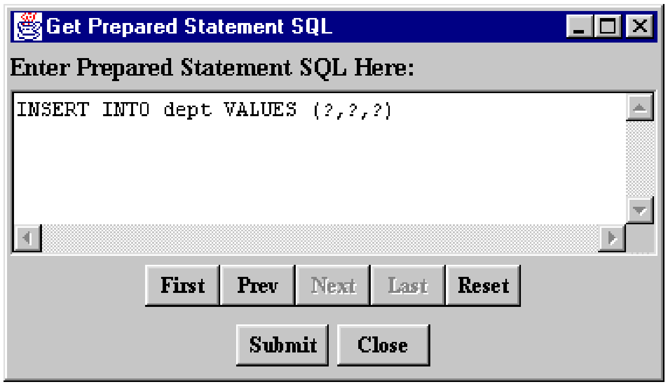
Batch Execution on a Prepared Statement

Batch execution on a prepared statement allows you to update or insert multiple records simultaneously. In some cases, this can significantly improve system performance because fewer round-trips to the database are required.

NOTE: Batch execution on a prepared statement is supported by the JDBC 2.0 specification and requires a Java 2 Platform (JDK 1.2)-compatible Java Virtual Machine.

1 From the Connection window menu, select **Connection / Create Prepared Statement**.

2 Specify the Insert statement that you want to execute.


Get Prepared Statement SQL
Enter Prepared Statement SQL Here:
First
Prev
Next
Last
Reset
Submit
Close

```
INSERT INTO dept VALUES (?,?,?)
```

Click **Submit**; then, click **Close**.

3 Select **Statement / Add Stmt Batch**.

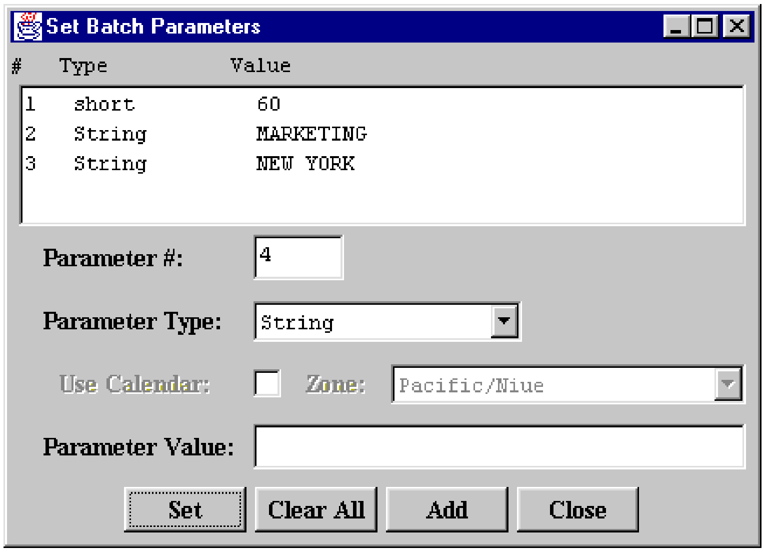
4 For each parameter:

a Type the parameter number.

b Select the parameter type.

c Type the parameter value.

d Click **Set**.



#	Type	Value
1	short	60
2	String	MARKETING
3	String	NEW YORK

Parameter #: 4

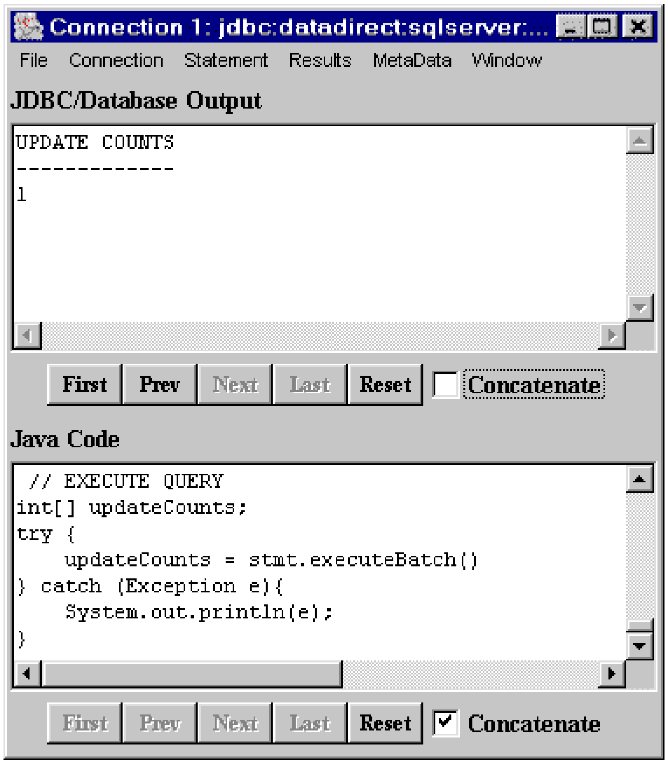
Parameter Type: String

Use Calendar: ☐ Zone: Pacific/Niue

Parameter Value:

5 Click **Add** to add the specified set of parameters to the batch. To add multiple parameter sets to the batch, repeat Step 3 through Step 5 as many times as necessary. When you are finished adding parameter sets to the batch, click **Close**.

6 Select **Statement / Execute Stmt Batch**. DataDirect Test displays the rowcount for each of the elements in the batch.

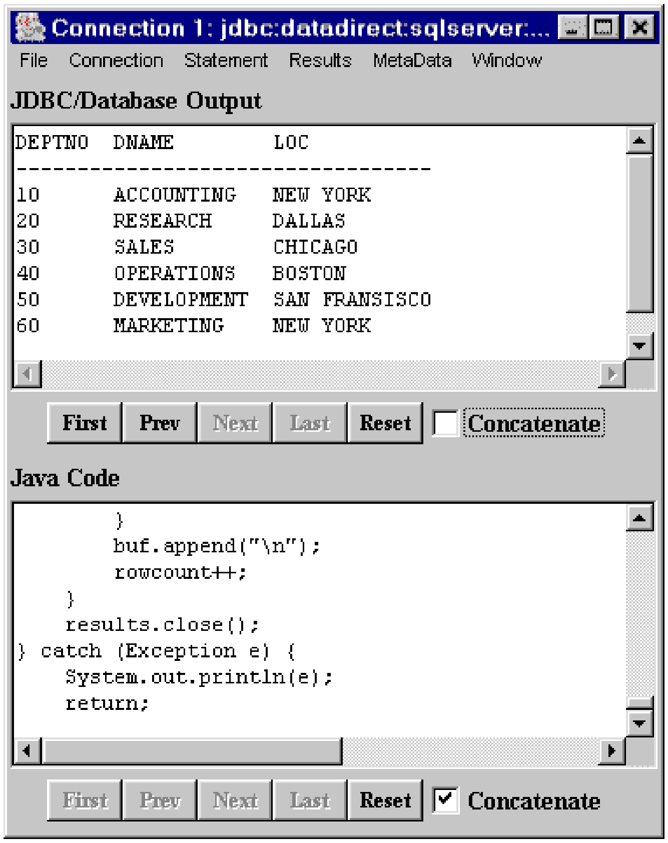

Connection 1: jdbc:datadirect:sqlserver:...
File Connection Statement Results MetaData Window
JDBC/Database Output
First Prev Next Last Reset Concatenate
Java Code
First Prev Next Last Reset Concatenate

```
UPDATE COUNTS
-------------
1
```

```
 // EXECUTE QUERY
int[] updateCounts;
try {
    updateCounts = stmt.executeBatch()
} catch (Exception e){
    System.out.println(e);
}
```

7 If you re-execute the Select statement from "Executing a Simple Select Statement" on page 211, you see that the previously inserted records are returned.

.



```
DEPTNO  DNAME         LOC
----------------------------------
10      ACCOUNTING    NEW YORK
20      RESEARCH      DALLAS
30      SALES         CHICAGO
40      OPERATIONS    BOSTON
50      DEVELOPMENT   SAN FRANSISCO
60      MARKETING     NEW YORK
```

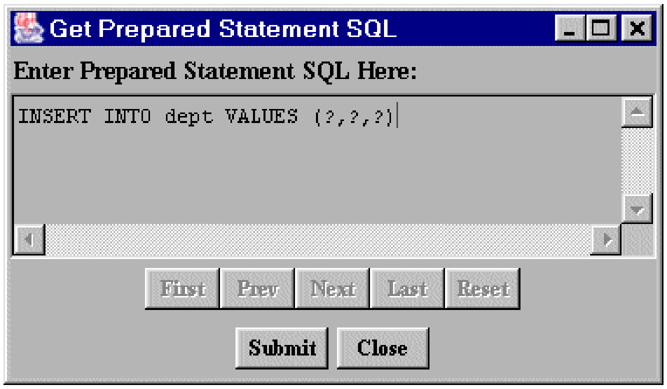
```
        }
        buf.append("\n");
        rowcount++;
    }
    results.close();
} catch (Exception e) {
    System.out.println(e);
    return;
```

Returning ParameterMetaData

NOTE: Returning ParameterMetaData is a JDBC 3.0 feature and requires a JDK 1.4 Java Virtual Machine.

1 From the Connection window menu, select **Connection / Create Prepared Statement**.

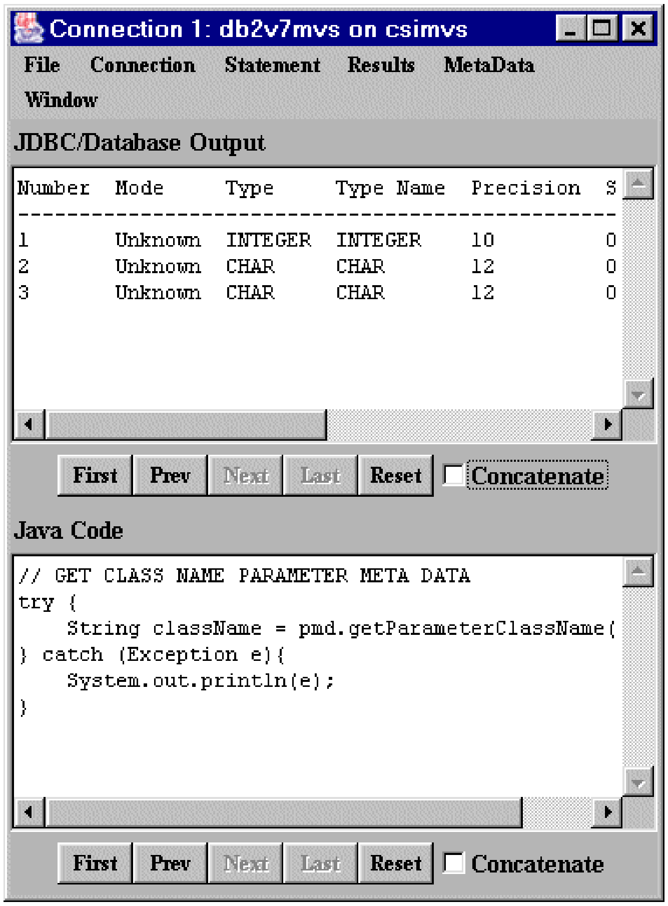
2 Specify the prepared statement that you want to execute.


Get Prepared Statement SQL
Enter Prepared Statement SQL Here:
First
Prev
Next
Last
Reset
Submit
Close

```
INSERT INTO dept VALUES (?,?,?)
```

Click **Submit**; then, click **Close**.

3 Select **Statement / Get ParameterMetaData**. The Connection window displays ParameterMetaData.



```
Number  Mode     Type     Type Name  Precision  S
-------------------------------------------------
1       Unknown  INTEGER  INTEGER    10         0
2       Unknown  CHAR     CHAR       12         0
3       Unknown  CHAR     CHAR       12         0
```

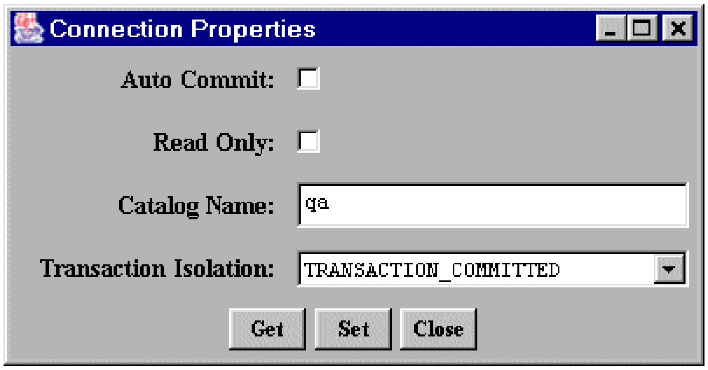
```
// GET CLASS NAME PARAMETER META DATA
try {
    String className = pmd.getParameterClassName(
} catch (Exception e){
    System.out.println(e);
}
```

Establishing Savepoints

NOTE: Savepoints is a JDBC 3.0 feature and requires a JDK 1.4 Java Virtual Machine.

1 From the Connection window menu, select **Connection / Connection Properties**.

2 Select TRANSACTION_COMMITTED from the Transaction Isolation drop-down list. Do not select the Auto Commit check box.



Connection Properties

Auto Commit: ☐

Read Only: ☐

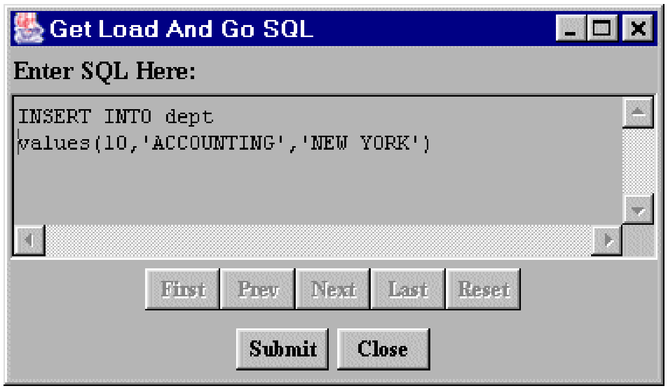
Catalog Name: qa

Transaction Isolation: TRANSACTION_COMMITTED

Click **Set**; then, click **Close**.

3 From the Connection window menu, select **Connection / Load and Go**. The Get Load and Go SQL window appears.

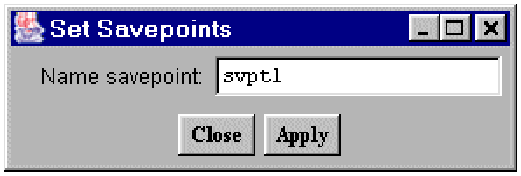
4 Specify the statement that you want to execute.



```
INSERT INTO dept
values(10,'ACCOUNTING','NEW YORK')
```

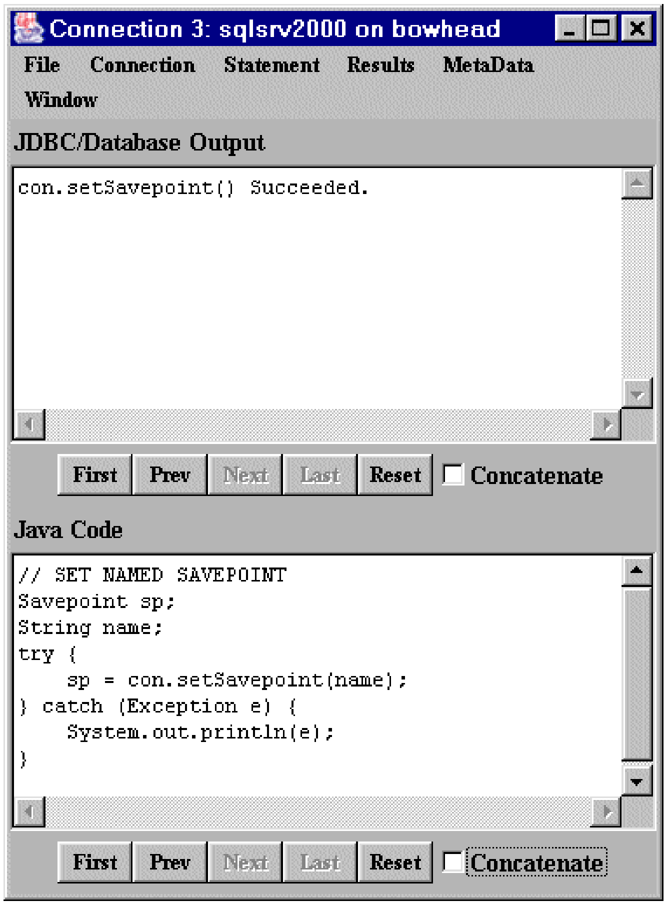
Click **Submit**.

5 Select **Connection / Set Savepoint**. In the Set Savepoints window, specify a savepoint name.



Name savepoint: svpt1

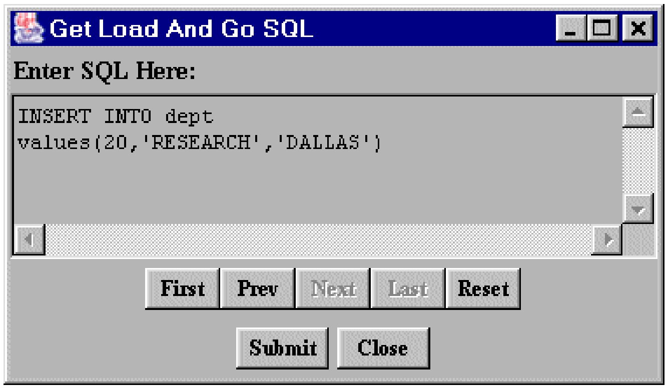
Click **Apply**; then, click **Close**. The Connection window indicates whether or not the savepoint succeeded.



```
con.setSavepoint() Succeeded.
```

```
// SET NAMED SAVEPOINT
Savepoint sp;
String name;
try {
    sp = con.setSavepoint(name);
} catch (Exception e) {
    System.out.println(e);
}
```

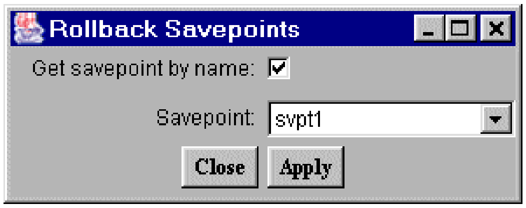
6 Return to the Get Load and Go SQL window and specify another statement.



```
INSERT INTO dept
values(20,'RESEARCH','DALLAS')
```

Click **Submit**.

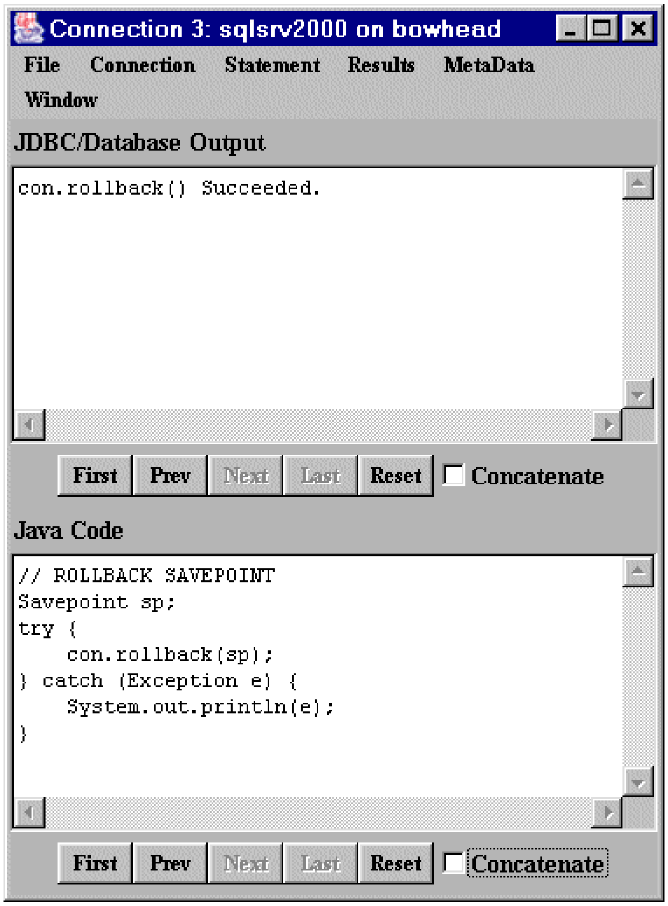
7 Select **Connection / Rollback Savepoint**. In the Rollback Savepoints window, specify the savepoint name.



Get savepoint by name: ✓

Savepoint: svpt1

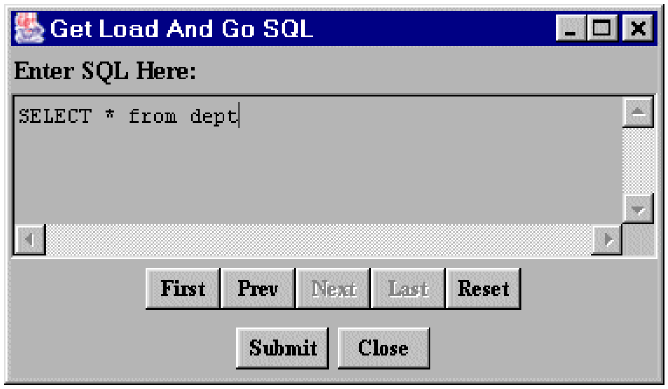
Click **Apply**; then, click **Close**. The Connection window indicates whether or not the savepoint rollback succeeded.



```
con.rollback() Succeeded.
```

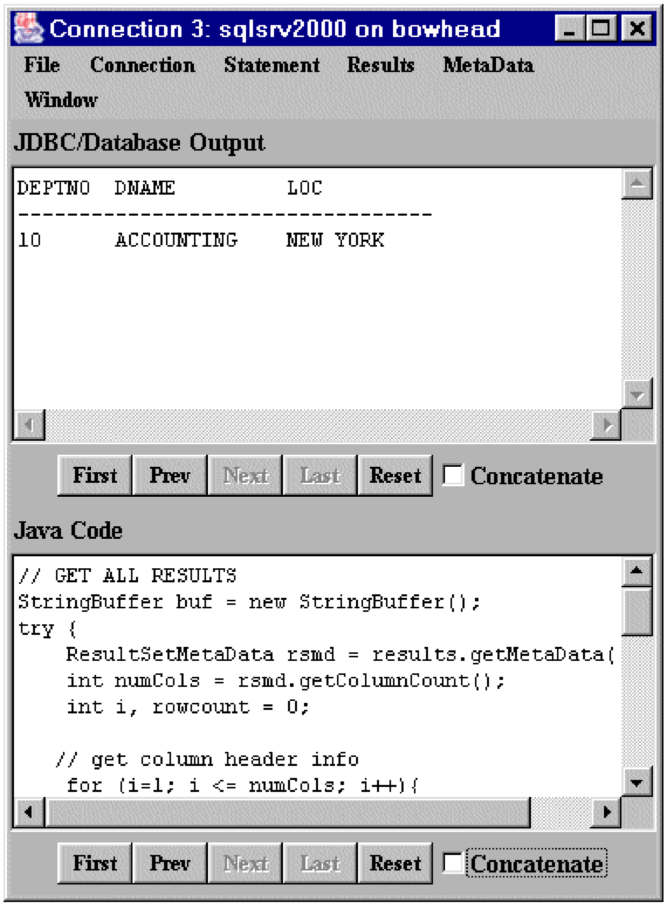
```
// ROLLBACK SAVEPOINT
Savepoint sp;
try {
    con.rollback(sp);
} catch (Exception e) {
    System.out.println(e);
}
```

8 Return to the Get Load and Go SQL window and specify another statement.



```
SELECT * from dept
```

Click **Submit**; then, click **Close**. The Connection window displays data that was inserted before the first Savepoint. The second insert was rolled back.


Connection 3: sqlsrv2000 on bowhead
File Connection Statement Results MetaData Window
JDBC/Database Output
First Prev Next Last Reset Concatenate
Java Code
First Prev Next Last Reset Concatenate

```
DEPTNO  DNAME          LOC
----------------------------------
10      ACCOUNTING     NEW YORK
```

```
// GET ALL RESULTS
StringBuffer buf = new StringBuffer();
try {
    ResultSetMetaData rsmd = results.getMetaData(
    int numCols = rsmd.getColumnCount();
    int i, rowcount = 0;

   // get column header info
    for (i=1; i <= numCols; i++){
```

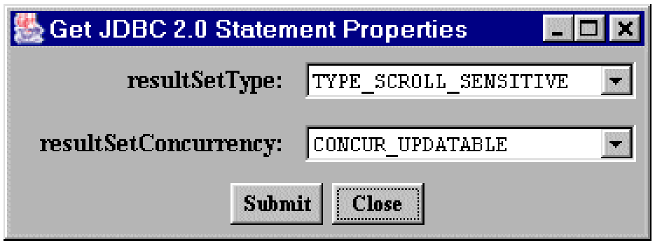
Updatable Result Sets

The following examples illustrate Updatable result sets by deleting, inserting, and updating a row.

Deleting a Row

1 From the Connection window menu, select **Connection / Create JDBC 2.0 Statement**.

2 In the resultSetType field, select **TYPE_SCROLL_SENSITIVE**. In the resultSetConcurrency field, select **CONCUR_UPDATABLE**.



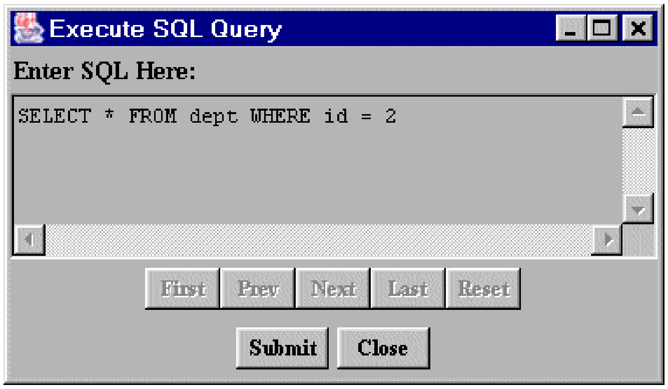
resultSetType: TYPE_SCROLL_SENSITIVE

resultSetConcurrency: CONCUR_UPDATABLE

Click **Submit**; then, click **Close**.

3 Select **Statement / Execute Stmt Query**.

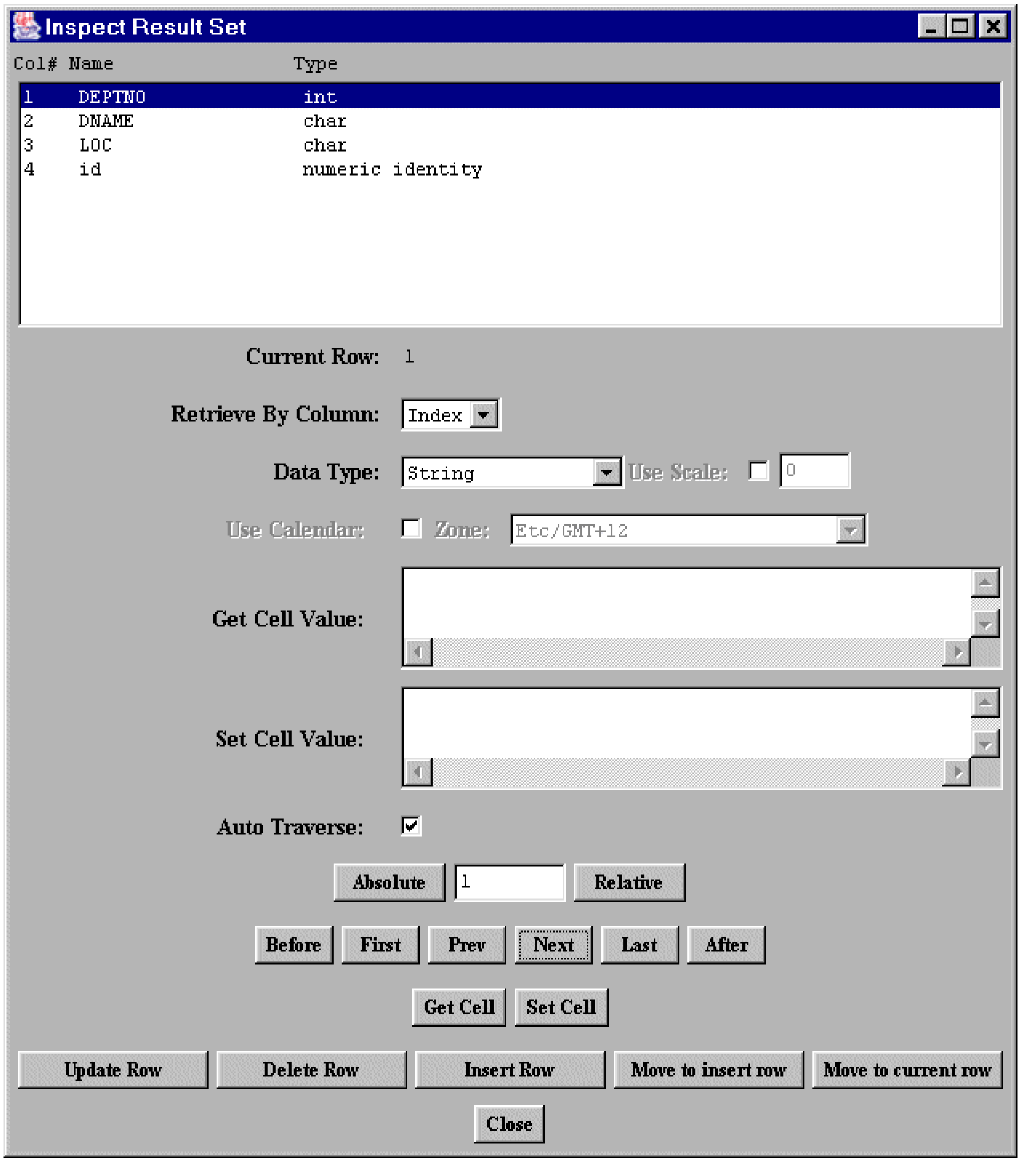
4 Specify the Select statement that you want to execute.



```
SELECT * FROM dept WHERE id = 2
```

Click **Submit**; then, click **Close**.

5 Select **Results / Inspect Results**. The Inspect Result Set window is displayed.



```
Col# Name          Type
1    DEPTNO        int
2    DNAME         char
3    LOC           char
4    id            numeric identity
```

Current Row: 1

Retrieve By Column: Index

Data Type: String

Use Scale: ☐ 0

Use Calendar: ☐ Zone: Etc/GMT+12

Get Cell Value:

Set Cell Value:

Auto Traverse: ✓

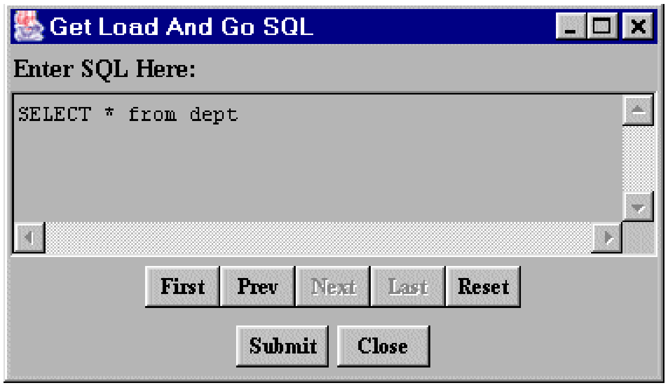
Absolute 1 Relative

6 Click **Next**. Current Row changes to 1.

7 Click **Delete Row**.

8 To verify the result, return to the Connection menu and select **Connection / Load and Go**. The Get Load and Go SQL window appears.

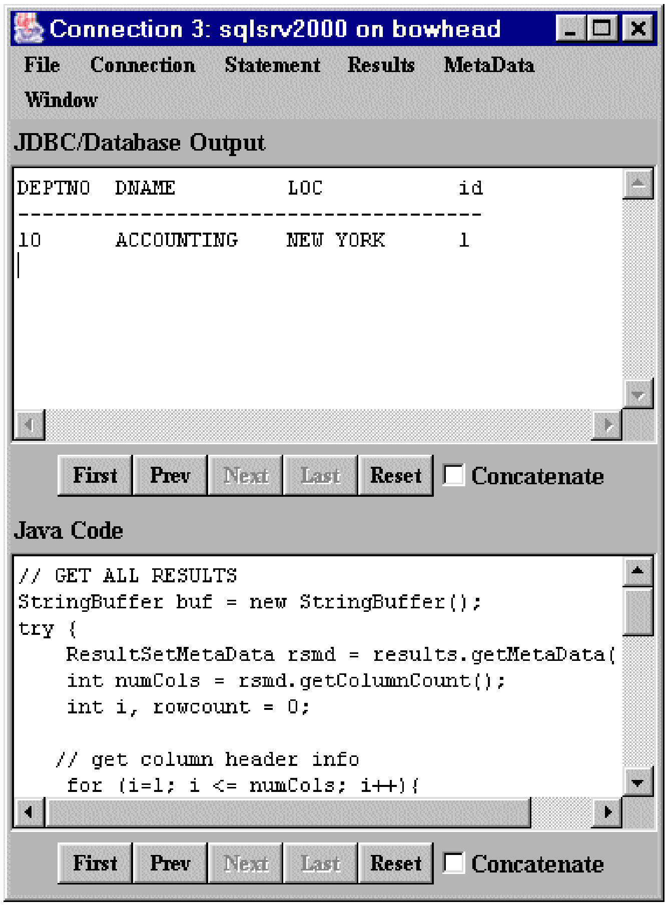
9 Specify the statement that you want to execute.



```
SELECT * from dept
```

Click **Submit**; then, click **Close**.

10 The Connection window shows one row returned.



```
DEPTNO  DNAME          LOC            id
---------------------------------------
10      ACCOUNTING     NEW YORK       1
```

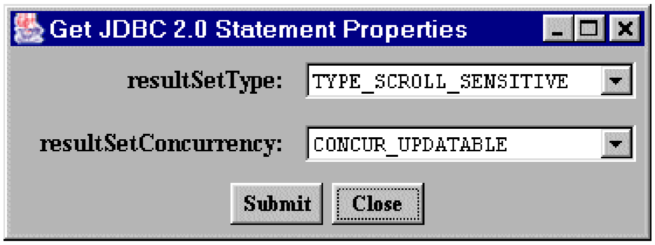
```
// GET ALL RESULTS
StringBuffer buf = new StringBuffer();
try {
    ResultSetMetaData rsmd = results.getMetaData(
    int numCols = rsmd.getColumnCount();
    int i, rowcount = 0;

   // get column header info
    for (i=1; i <= numCols; i++){
```

Inserting a Row

1 From the Connection window menu, select **Connection / Create JDBC 2.0 Statement**.

2 In the resultSetType field, select **TYPE_SCROLL_SENSITIVE**. In the resultSetConcurrency field, select **CONCUR_UPDATABLE**.

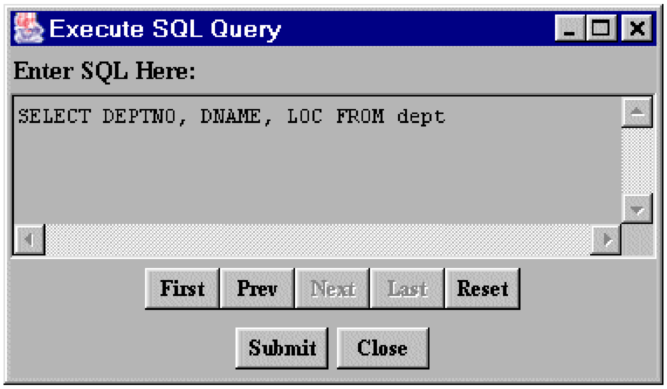


resultSetType: TYPE_SCROLL_SENSITIVE
resultSetConcurrency: CONCUR_UPDATABLE

Click **Submit**; then, click **Close**.

3 Select **Statement / Execute Stmt Query**.

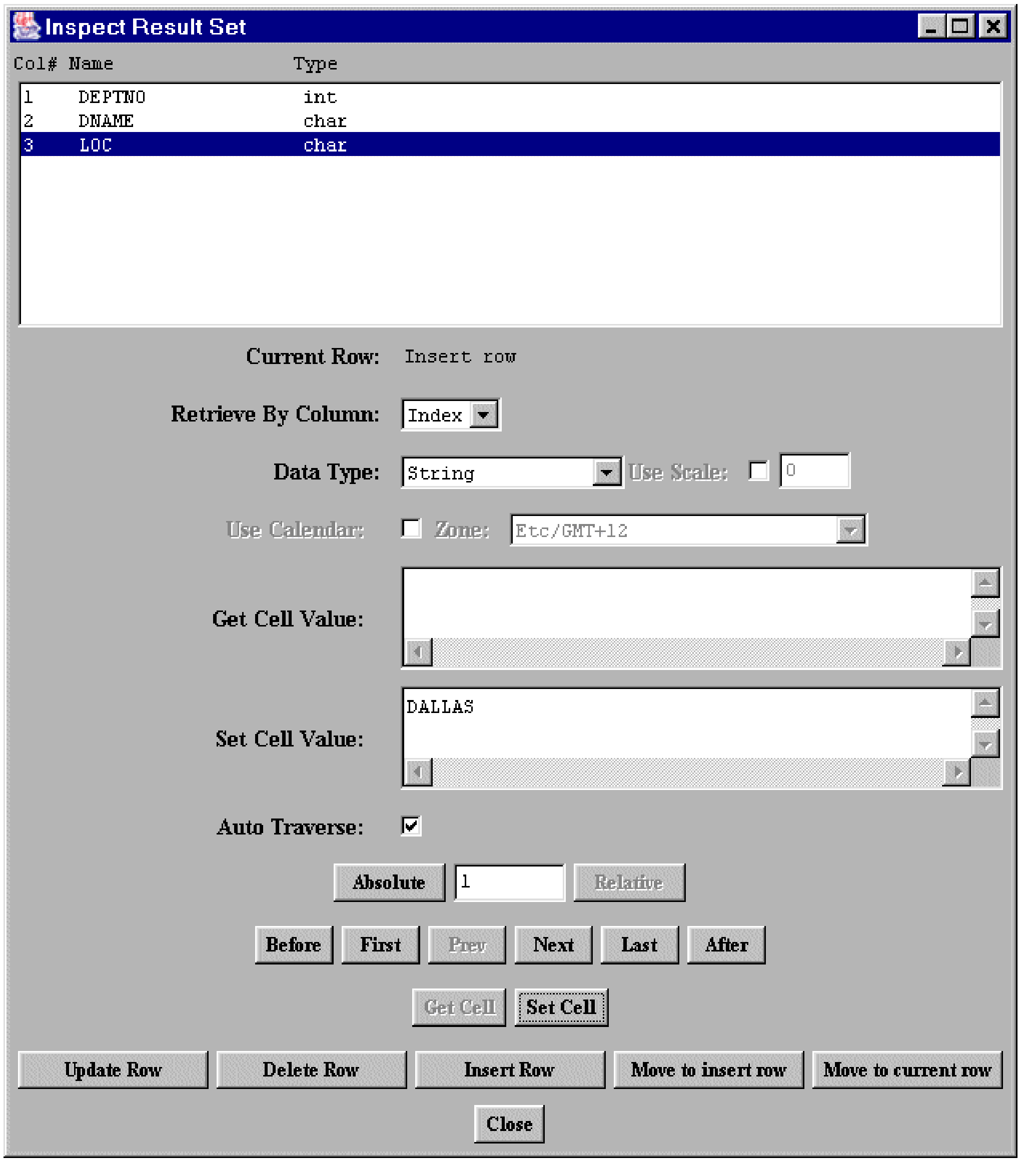
4 Specify the Select statement that you want to execute.



```
SELECT DEPTNO, DNAME, LOC FROM dept
```

Click **Submit**; then, click **Close**.

5 Select **Results / Inspect Results**. The Inspect Result Set window is displayed.


Inspect Result Set

```
Col# Name            Type
1    DEPTNO          int
2    DNAME           char
3    LOC             char
```

Current Row: Insert row

Retrieve By Column: Index

Data Type: String   Use Scale: ☐ 0

Use Calendar: ☐ Zone: Etc/GMT+12

Get Cell Value:

Set Cell Value: DALLAS

Auto Traverse: ✓

6 Click **Move to insert row**; Current Row is now Insert row.

7 Change Data Type to int. In Set Cell Value, enter 20. Click **Set Cell**.

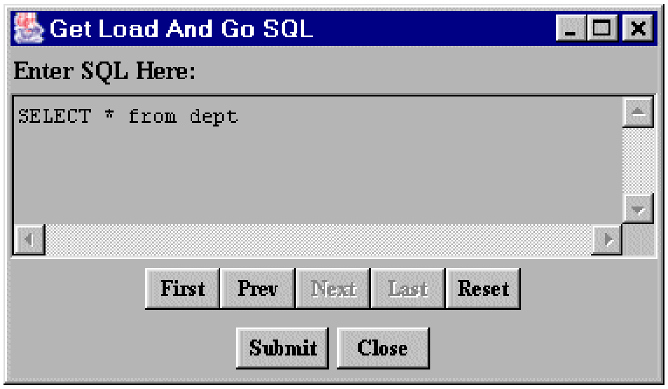
8 Select the second row in the top pane. Change the Data Type to String. In Set Cell Value, enter RESEARCH. Click **Set Cell**.

9 Select the third row in the top pane. In Set Cell Value, enter DALLAS. Click **Set Cell**.

10 Click **Insert Row**.

11 To verify the result, return to the Connection menu and select **Connection / Load and Go**. The Get Load and Go SQL window appears.

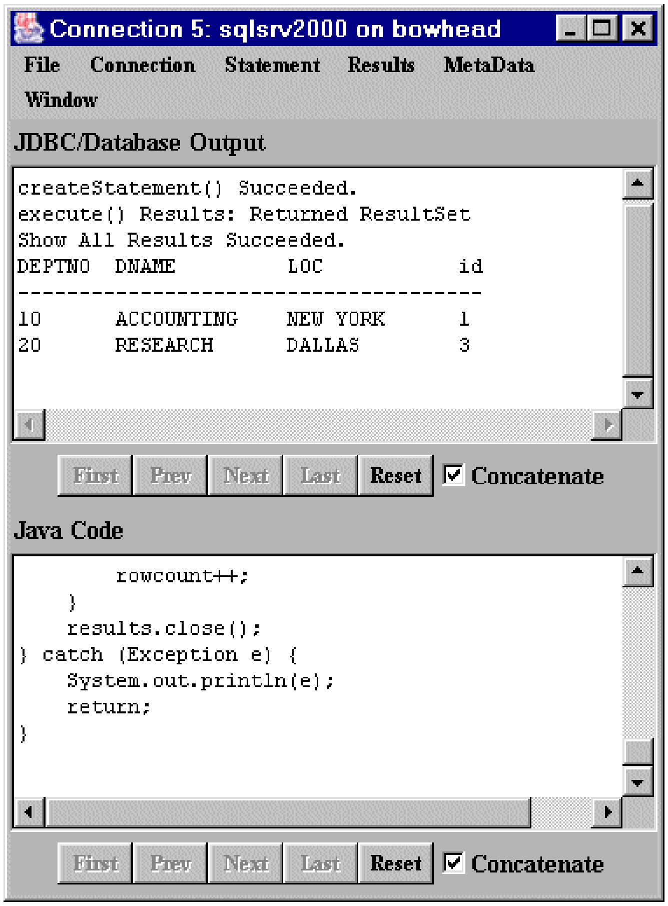
12 Specify the statement that you want to execute.



```
SELECT * from dept
```

Click **Submit**; then, click **Close**.

13 The Connection window shows two rows returned.


Connection 5: sqlsrv2000 on bowhead
File Connection Statement Results MetaData
Window
JDBC/Database Output
First Prev Next Last Reset Concatenate
Java Code
First Prev Next Last Reset Concatenate

```
createStatement() Succeeded.
execute() Results: Returned ResultSet
Show All Results Succeeded.
DEPTNO  DNAME          LOC           id
----------------------------------------
10      ACCOUNTING     NEW YORK      1
20      RESEARCH       DALLAS        3
```

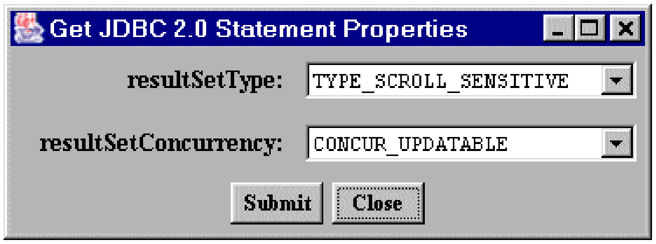
```
        rowcount++;
    }
    results.close();
} catch (Exception e) {
    System.out.println(e);
    return;
}
```

Note that the ID will be 3 for the row just inserted, because it is an auto increment column.

Updating a Row

1 From the Connection window menu, select **Connection / Create JDBC 2.0 Statement**.

2 In the resultSetType field, select **TYPE_SCROLL_SENSITIVE**. In the resultSetConcurrency field, select **CONCUR_UPDATABLE**.



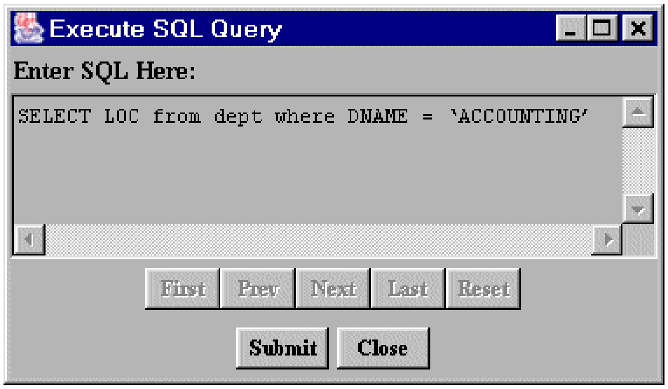
resultSetType: TYPE_SCROLL_SENSITIVE

resultSetConcurrency: CONCUR_UPDATABLE

Click **Submit**; then, click **Close**.

3 Select **Statement / Execute Stmt Query**.

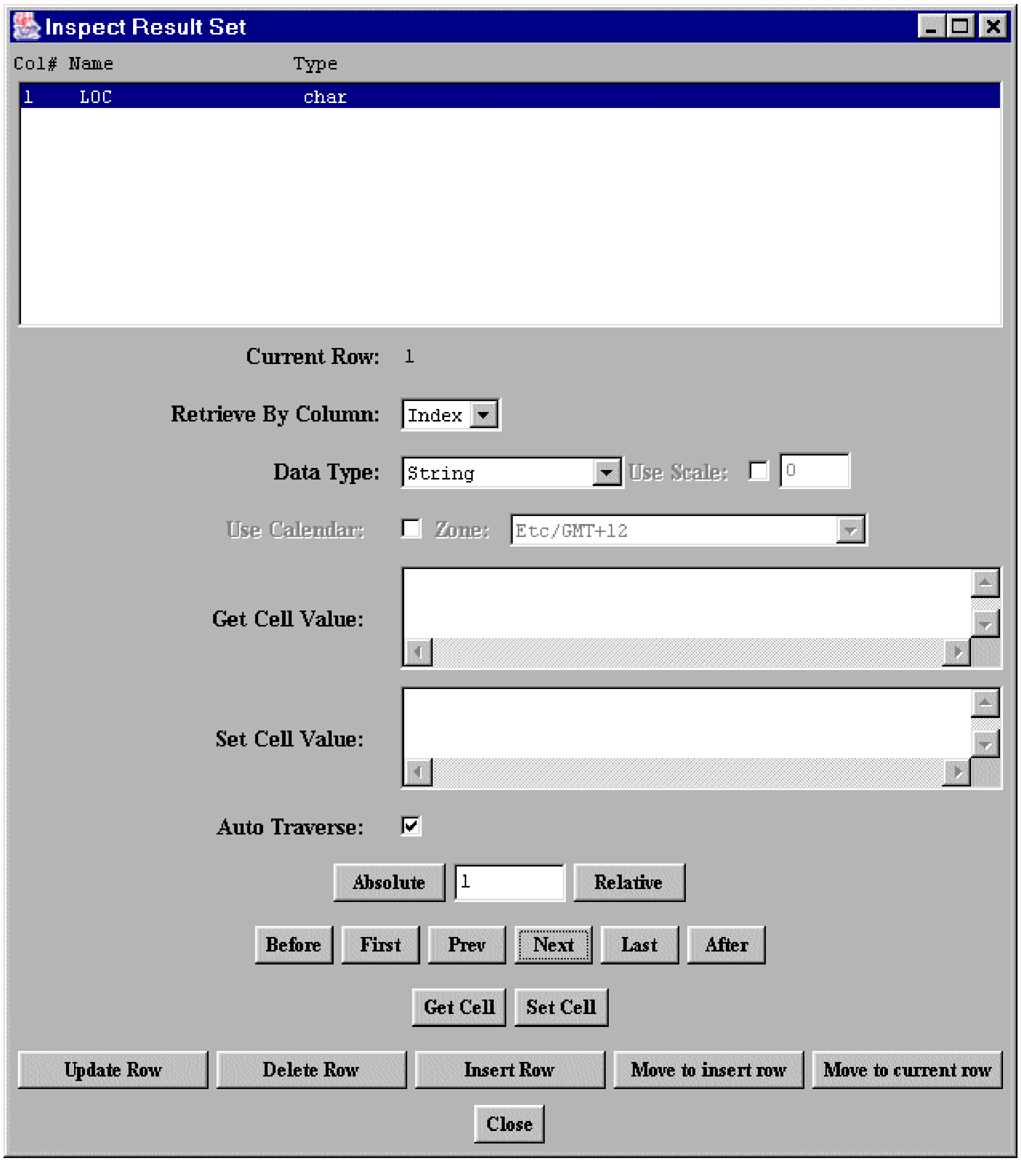
4 Specify the Select statement that you want to execute.



```
SELECT LOC from dept where DNAME = 'ACCOUNTING'
```

Click **Submit**; then, click **Close**.

5 Select **Results / Inspect Results**. The Inspect Result Set window is displayed.



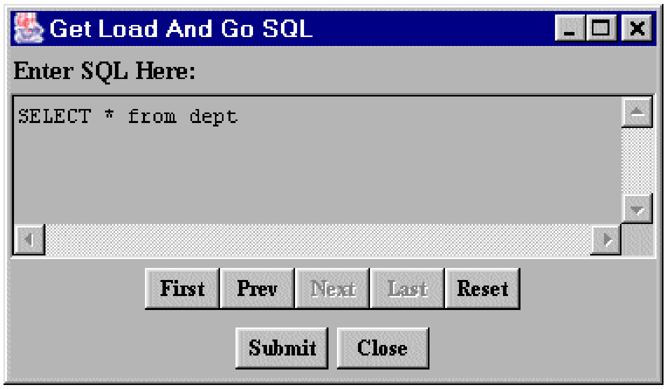
6 Click **Next**. Current Row changes to 1.

7 In Set Cell Value, enter RALEIGH. Click **Set Cell**.

8 Click **Update Row.**

9 To verify the result, return to the Connection menu and select **Connection / Load and Go**. The Get Load and Go SQL window appears.

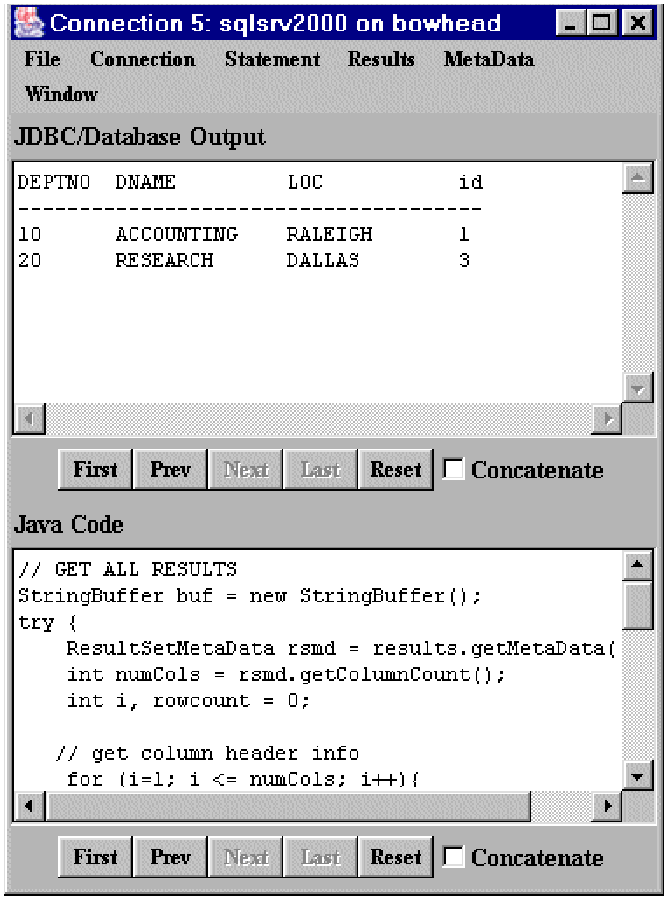
10 Specify the statement that you want to execute.



```
SELECT * from dept
```

Click **Submit**; then, click **Close**.

11 The Connection window shows LOC for accounting changed from NEW YORK to RALEIGH.


Connection 5: sqlsrv2000 on bowhead
File Connection Statement Results MetaData Window
JDBC/Database Output
First Prev Next Last Reset Concatenate
Java Code
First Prev Next Last Reset Concatenate

```
DEPTNO  DNAME          LOC            id
---------------------------------------
10      ACCOUNTING     RALEIGH        1
20      RESEARCH       DALLAS         3
```

```
// GET ALL RESULTS
StringBuffer buf = new StringBuffer();
try {
    ResultSetMetaData rsmd = results.getMetaData(
    int numCols = rsmd.getColumnCount();
    int i, rowcount = 0;

   // get column header info
    for (i=1; i <= numCols; i++){
```

Large Object (LOB) Support

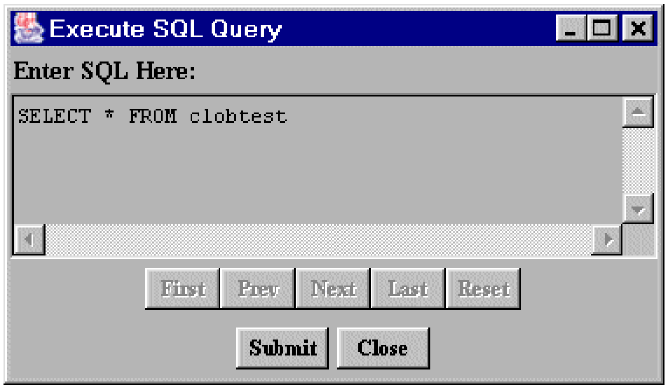
NOTE: Large object (LOB) support (Blobs and Clobs) is a JDBC 3.0 feature and requires a JDK 1.4 Java Virtual Machine.

The following example uses Clob data; however, this procedure also applies to Blob data. This example illustrates only one of several ways in which LOB data can be processed.

1 From the Connection window menu, select **Connection / Create Statement**.

2 Select **Statement / Execute Stmt Query**.

3 Specify the Select statement that you want to execute.

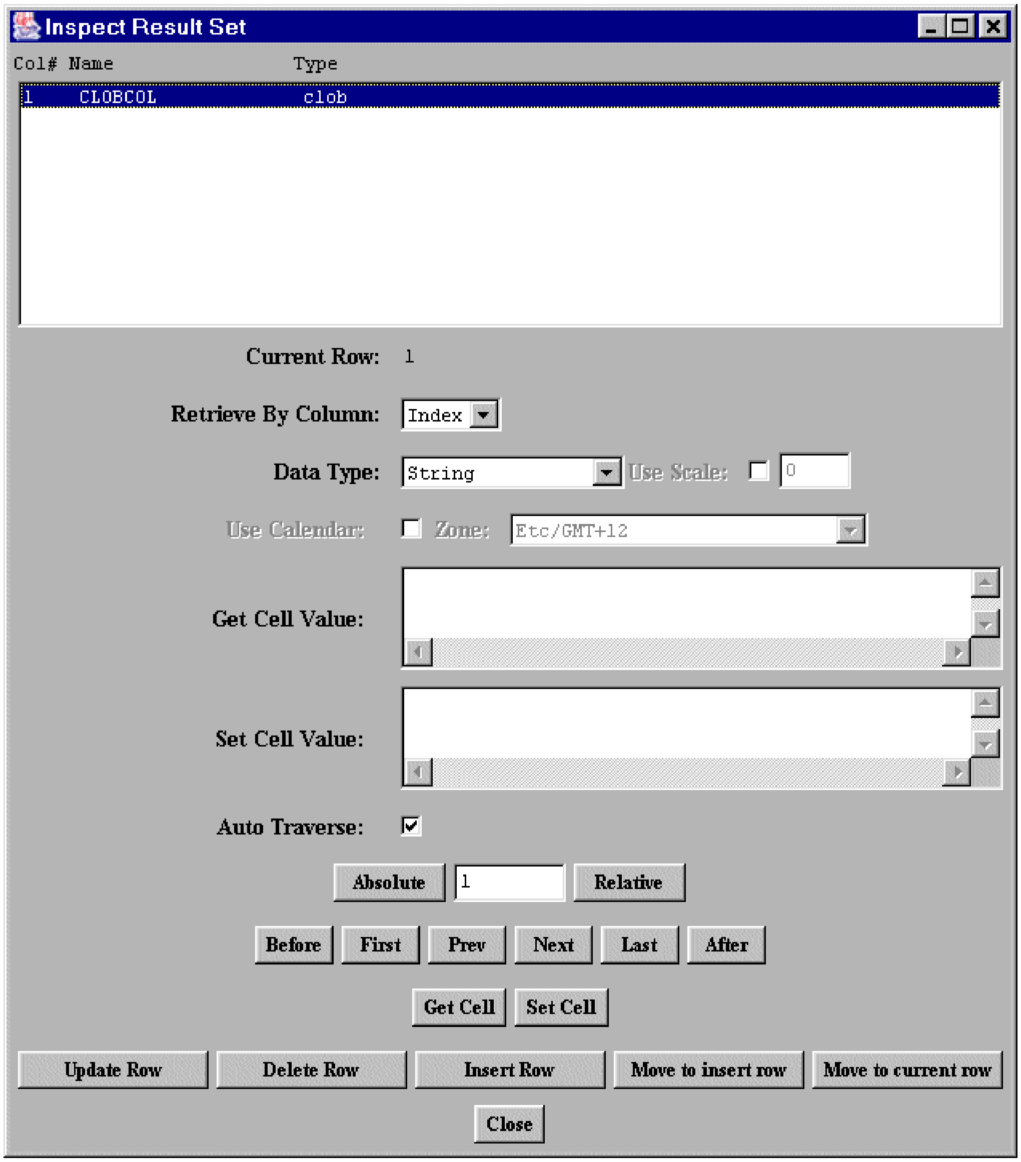


```
SELECT * FROM clobtest
```

Click **Submit**; then, click **Close**.

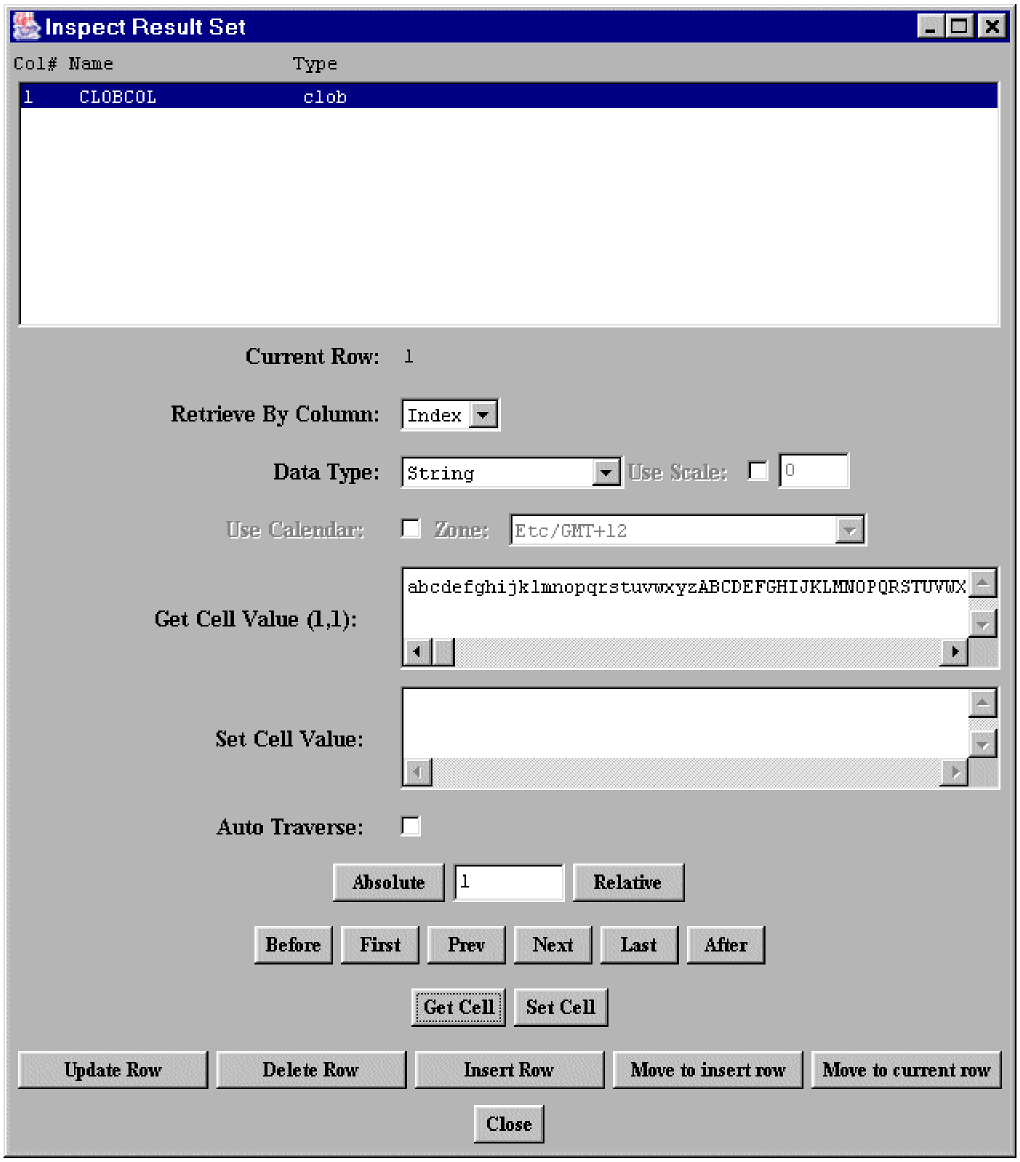
4 Select **Results / Inspect Results**. The Inspect Result Set window is displayed.

5 Click **Next**. Current Row changes to 1.



6 Deselect Auto Traverse. This disables automatic traversal to the next row.

7 Click **Get Cell**.



```
Col# Name          Type
1    CLOBCOL       clob
```

Current Row: 1

Retrieve By Column: Index

Data Type: String

Use Scale: 0

Use Calendar: Zone: Etc/GMT+12

Get Cell Value (1,1): abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWX

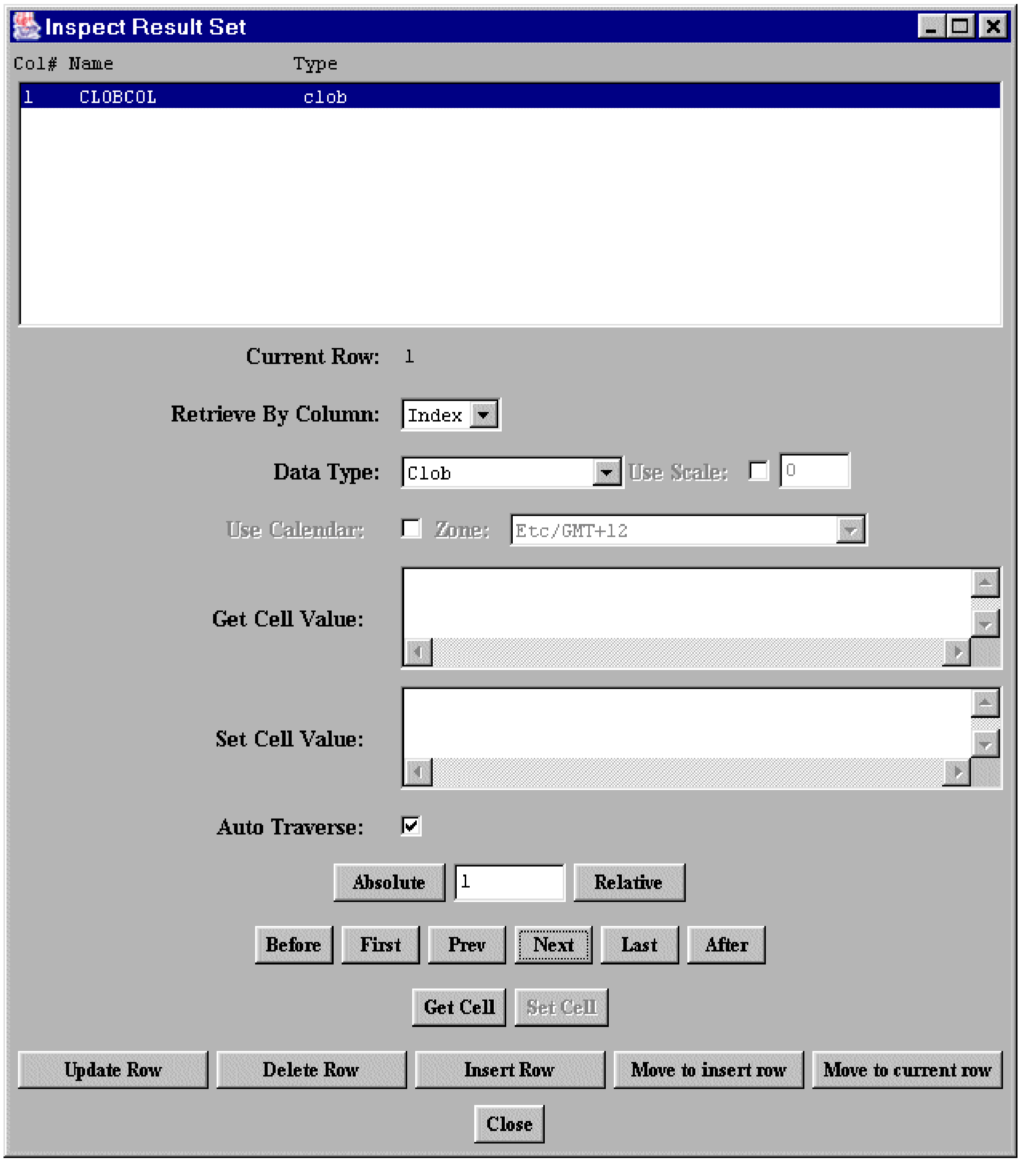
Set Cell Value:

Auto Traverse:

Absolute: 1

8 Values are returned in the Get Cell Value field.

9 Change the Data Type to Clob


Inspect Result Set

```
Col# Name          Type
1    CLOBCOL       clob
```

Current Row: 1

Retrieve By Column: Index

Data Type: Clob

Use Scale: ☐ 0

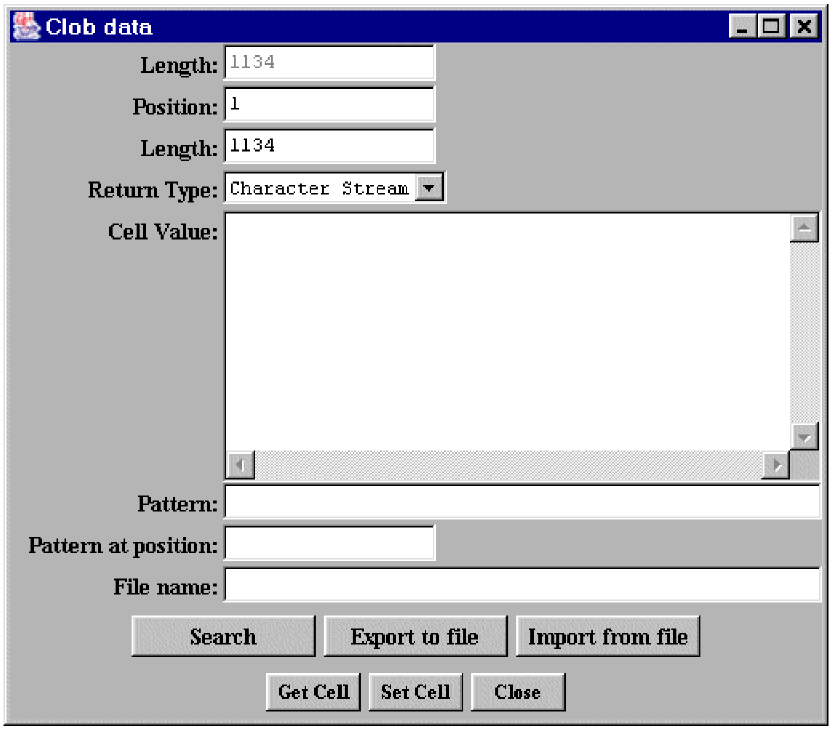
Use Calendar: ☐ Zone: Etc/GMT+12

Get Cell Value:

Set Cell Value:

Auto Traverse: ☑

10 Click **Get Cell**. The Clob Data window appears.


Clob data

Length: 1134

Position: 1

Length: 1134

Return Type: Character Stream

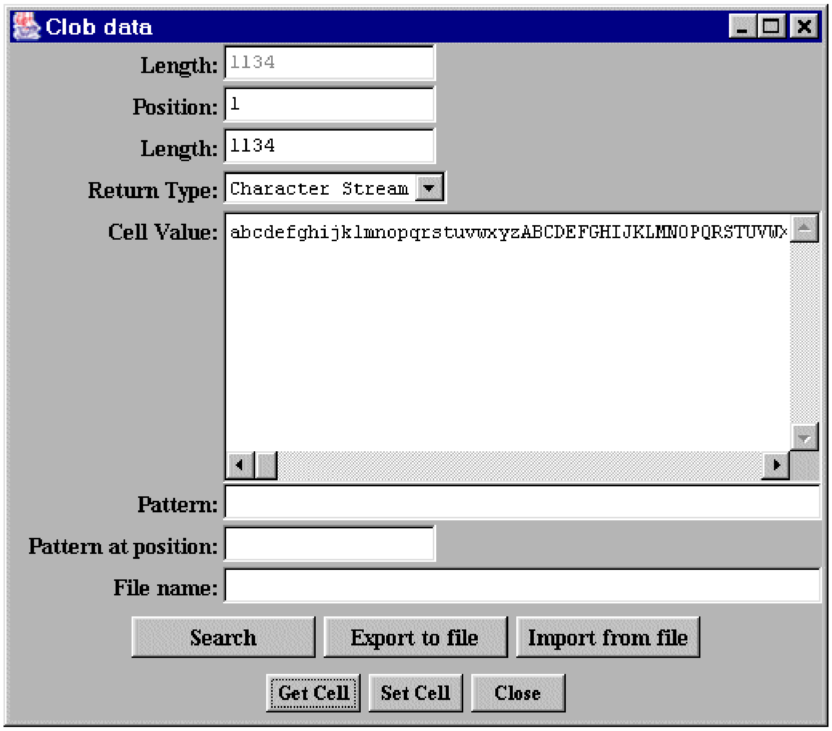
Cell Value:

Pattern:

Pattern at position:

File name:

11 Click **Get Cell**.



Length:	1134
Position:	1
Length:	1134
Return Type:	Character Stream
Cell Value:	abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVW
Pattern:	
Pattern at position:	
File name:	

12 Values are returned in the Get Cell Value field.

F Tracking JDBC Calls with DataDirect Spy

DataDirect Spy passes calls issued by an application to an underlying DataDirect Connect *for* JDBC driver and logs detailed information about those calls. It provides the following advantages:

- Logging is JDBC 1.22-, JDBC 2.0-, and JDBC 3.0-compliant, including support for the JDBC 2.0 Optional Package.
- Logging is consistent, regardless of the DataDirect Connect *for* JDBC driver used.
- All parameters and function results for JDBC calls can be logged.
- Logging works with all DataDirect Connect *for* JDBC drivers.
- Logging can be enabled, without changing the application, using a DataDirect Connect *for* JDBC data source.

Loading the DataDirect Spy Driver

To use DataDirect Spy driver, you first must register it with the JDBC Driver Manager. You can register the DataDirect Spy driver in any of the following ways:

- **Method 1**: Set the Java property sql.drivers using the Java -D option. The sql.drivers property is defined as a colon-separated list of driver class names. For example:

  ```
  com.ddtek.jdbcspy.SpyDriver:
  sun.jdbc.odbc.JdbcOdbcDriver
  ```

The sql.drivers property can be set like other Java properties, using the -D option. For example:

```
java -Dsql.drivers=com.ddtek.jdbcspy.SpyDriver
```

- **Method 2**: Set the Java property sql.drivers from within your Java application or applet. To do this, code the following lines in your Java application or applet, and call DriverManager.getConnection():

```
Properties p = System.getProperties();
p.put ("sql.drivers", "com.ddtek.jdbcspy.SpyDriver");
System.setProperties (p);
```

- **Method 3**: Explicitly load the driver class using the standard Class.forName() method. To do this, code the following line and call DriverManager.getConnection():

```
Class.forName("com.ddtek.jdbcspy.SpyDriver");
```

Checking the DataDirect Spy Version

To check the version of your DataDirect Spy, change to the directory containing DataDirect Spy, and at a command prompt, enter the command:

On Windows:

```
java -classpath spy_dir\spy.jar com.ddtek.jdbcspy.SpyDriver
```

On UNIX:

```
java -classpath spy_dir/spy.jar com.ddtek.jdbcspy.SpyDriver
```

where *spy_dir* is the directory containing DataDirect Spy.

DataDirect Spy URL Syntax and Attributes

DataDirect Spy uses the following format as a connection URL:

```
jdbc:spy:{original-url};[key=value]...
```

where:

`original-url` is the connection URL of the underlying JDBC driver, for example:

```
jdbc:datadirect:DB2://db2host:1433;user=john;password=
johnspw
```

`key-value` is any of the following:

log=System.out	Redirects logging to the Java output standard.
log=(file)*filename*	Redirects logging to the file specified by *filename*. By default, DataDirect Spy will use the stream specified in DriverManager.setLogStream().
load=*classname*	Loads the driver specified by *classname*. For example, com.ddtek.jdbc. db2.DB2Driver.
linelimit=*numberofchars*	The maximum number of characters, specified by *numberofchars*, that DataDirect Spy will log on one line. The default is 0 (no maximum limit).

logIS={yes | no | nosingleread}	Specifies whether DataDirect Spy logs activity on InputStream and Reader objects. When logIS=nosingleread, logging on InputStream and Reader objects is active; however logging of the single-byte read InputStream.read() or single-character Reader.read() is suppressed. This avoids the generation of large log files containing single-byte / single character read messages. The default is no.
logTName={yes | no}	Specifies whether DataDirect Spy logs the name of the current thread. The default is no.
timestamp={yes | no}	Specifies whether a timestamp should be included on each line of the DataDirect Spy log.

DataDirect Spy URL Examples

Example A:

```
jdbc:spy:{jdbc:datadirect:odbc:Oracle7};load=sun.jdbc.odbc.JdbcOdbcDriver;
log=(file)C:\temp\spy.log
```

Using this example, DataDirect Spy would:

1. Load the JDBC-ODBC bridge.
2. Log all JDBC activity to the file c:\temp\spy.log.

Example B:

```
jdbc:spy:{jdbc:datadirect:DB2://db2host:1433;user=john;password=johnspw};
log=System.out;linelimit=80
```

Using this example, DataDirect Spy would:

1 Load the DataDirect Connect *for* JDBC DB2 driver.

2 Log all JDBC activity to the standard output file.

3 Log a maximum of 80 characters for each line.

Using DataDirect Spy with JDBC Data Sources

The DataDirect Connect *for* JDBC drivers implement the following features defined by the JDBC 2.0 Optional Package:

- JNDI for Naming Databases
- Connection Pooling
- JTA

You can use DataDirect Spy to track JDBC calls with each of these features. The com.ddtek.jdbcx.datasource.*Driver*DataSource class, where *Driver* is the driver name, supports the SpyAttributes connection attribute, which specifies a semi-colon-separated list of DataDirect Spy attributes as described in "DataDirect Spy URL Syntax and Attributes" on page 257. For more information about configuring JDBC data sources, see "Connecting Through Data Sources" on page 30.

The following DB2 example creates a JDBC data source and specifies that all JDBC calls must be logged in the file /tmp/spy.log, including the name of the current thread.

```
...
DB2DataSource sds=new DB2DataSource():
sds.setServerName("MyServer");
sds.setPortNumber(1234);
sds.setSpyAttributes("log=(file)/tmp/spy.log;logTName=yes");
Connection conn=sds.getConnection("scott","tiger");
...
```

DataDirect Spy Log Example

NOTE: Numbers in bold superscript are note indicators. See the notes following the example for the referenced text.

```
All rights reserved.[1]
registerDriver:driver[className=com.ddtek.jdbcspy.SpyDriver,
context=null,com.ddtek.jdbcspy.SpyDriver@1ec49f][2]

*Driver.connect(jdbc:spy:{jdbc:datadirect:sqlserver://QANT:4003;
  databaseName=Test;})
    trying driver[className=com.ddtek.jdbcspy.SpyDriver,
context=null,com.ddtek.jdbcspy.SpyDriver@1ec49f][3]

spy>> Driver.connect(String url, Properties info)
spy>> url = jdbc:spy:{jdbc:datadirect:sqlserver://QANT:4003;databaseName=
Test;
OSUser=qauser;OSPassword=null12}
spy>> info = {password=tiger, user=scott}
spy>> OK (Connection[1])[4]

getConnection returning driver[className=com.ddtek.jdbcspy.SpyDriver,
context=null,com.ddtek.jdbcspy.SpyDriver@1ec49f][5]

spy>> Connection[1].getWarnings()
spy>> OK[6]

spy>> Connection[1].createStatement
spy>> OK (Statement[1])[7]

spy>> Statement[1].executeQuery(String sql)
spy>> sql = select empno,ename,job from emp where empno=7369
spy>> OK (ResultSet[1])[8]

spy>> ResultSet[1].getMetaData()
spy>> OK (ResultSetMetaData[1])[9]
spy>> ResultSetMetaData[1].getColumnCount()
spy>> OK (3)10

spy>> ResultSetMetaData[1].getColumnLabel(int column)
spy>> column = 1
spy>> OK (EMPNO)11

spy>> ResultSetMetaData[1].getColumnLabel(int column)
spy>> column = 2
spy>> OK (ENAME)12

spy>> ResultSetMetaData[1].getColumnLabel(int column)
spy>> column = 3
spy>> OK (JOB)13

spy>> ResultSet[1].next()
spy>> OK (true)14

spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 1
spy>> OK (7369)15

spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 2
spy>> OK (SMITH)16

spy>> ResultSet[1].getString(int columnIndex)
spy>> columnIndex = 3
spy>> OK (CLERK)17

spy>> ResultSet[1].next()
spy>> OK (false)18

spy>> ResultSet[1].close()
spy>> OK19

spy>> Connection[1].close()
spy>> OK20
```

NOTES:

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

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

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

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

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

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

[7, 8] The statement "select empno,ename,job from emp where empno=7369" is created.

[9, 10, 11, 12, 13] Some metadata is requested.

[14, 15, 16, 17] The first row is fetched and its data retrieved.

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

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

[20] The application finishes and disconnects.

G Connection Pool Manager

Establishing JDBC connections is resource-expensive, especially when the JDBC API is used in a middle-tier server environment. In this type of environment, performance can be improved significantly when connection pooling is used. Connection pooling means that connections are reused rather than created each time a connection is requested. Your application can use connection pooling through the DataDirect Connection Pool Manager.

Connection pooling is performed in the background and does not affect how an application is coded; however, the application must use a DataSource object (an object implementing the DataSource interface) to obtain a connection instead of using the DriverManager class. A class implementing the DataSource interface may or may not provide connection pooling. A DataSource object registers with a JNDI naming service. Once a DataSource object is registered, the application retrieves it from the JNDI naming service in the standard way.

Checking the DataDirect Connection Pool Manager Version

To check the version of your DataDirect Connection Pool Manager, navigate to the directory containing the DataDirect Connection Pool Manager, and at a command prompt, enter the command:

On Windows:

```
java -classpath poolmgr_dir\pool.jar com.ddtek.pool.PoolManagerInfo
```

On UNIX:

```
java -classpath poolmgr_dir/pool.jar com.ddtek.pool.PoolManagerInfo
```

where *poolmgr_dir* is the directory containing the DataDirect Connection Pool Manager.

Alternatively, you can obtain the name and version of the DataDirect Connection Pool Manager programmatically by invoking the following static methods: com.ddtek.pool.PoolManagerInfo.getPoolManagerName() and com.ddtek.pool.PoolManagerInfo.getPoolManagerVersion().

Creating a Data Source

This section contains sample code that is provided as an example of using the DataDirect Connection Pool Manager to allow your applications to handle connection pooling.

Creating a Pooled DataDirect Data Source Object

This example shows how to create a DataDirect Connect *for* JDBC DataSource object and register it to a JNDI naming service. The DataSource class is provided by your DataDirect Connect *for* JDBC driver and is database-dependent. In the following example we use Oracle, so the DataSource class is OracleDataSource. Refer to the specific driver chapters in this book for the name of the DataSource class for your driver.

If you want the client application to use a non-pooled data source, the application can specify the JNDI name of this data source object as registered in the following code example ("jdbc/PooledSparkyOracle").

If you want the client application to use a pooled data source, the application must specify the JNDI name as registered in the code example in the section on creating a data source. See "Creating a Data Source Using the DataDirect Connection Pool Manager" on page 266 for details.

```
//************************************************************************
//
// This code creates a DataDirect Connect for JDBC data source and
// registers it to a JNDI naming service. This DataDirect Connect for
// JDBC data source uses the DataSource implementation provided by
// DataDirect Connect for JDBC Drivers.
//
// If the client application uses a non-pooled data source, it
// may use the JNDI name <jdbc/PooledSparkyOracle> as registered
// in this example.
//
// If the client application uses a pooled data source, this example code
// must be modified to use the JNDI name <jdbc/SparkyOracle> as
// registered by the example code in section 5.2 of this document.
//
//************************************************************************

// From DataDirect Connect for JDBC:
import com.ddtek.jdbcx.oracle.OracleDataSource;

import javax.sql.*;
import java.sql.*;
import javax.naming.*;
import javax.naming.directory.*;
import java.util.Hashtable;

public class OracleDataSourceRegisterJNDI
{
   public static void main(String argv[])
   {
      try {
         // Set up data source reference data for naming context:
         // ----------------------------------------------------
         // Create a class instance that implements the interface
         // ConnectionPoolDataSource
```

```
            OracleDataSource ds = new OracleDataSource();

            ds.setDescription(
               "Oracle on Sparky - Oracle Data Source");
            ds.setServerName("sparky");
            ds.setPortNumber(1521);
            ds.setUser("scott");
            ds.setPassword("test");

            // Set up environment for creating initial context
            Hashtable env = new Hashtable();
            env.put(Context.INITIAL_CONTEXT_FACTORY,
               "com.sun.jndi.fscontext.RefFSContextFactory");
            env.put(Context.PROVIDER_URL, "file:c:\\JDBCDataSource");
            Context ctx = new InitialContext(env);

            // Register the data source to JNDI naming service
            ctx.bind("jdbc/PooledSparkyOracle", ds);

         } catch (Exception e) {
            System.out.println(e);
            return;
         }
      } // Main
   } // class OracleDataSourceRegisterJNDI
```

Creating a Data Source Using the DataDirect Connection Pool Manager

The following Java code example creates a data source for DataDirect Connect *for* JDBC and registers it to a JNDI naming service. The PooledConnectionDataSource class is provided by the DataDirect com.ddtek.pool package. In the following code example, the PooledConnectionDataSource object references a pooled DataDirect Connect *for* JDBC data source object. Therefore, the example performs a lookup by setting the DataSourceName attribute to the JNDI name of a registered pooled data source (in this example, jdbc/PooledSparkyOracle, which is the DataDirect

Connect *for* JDBC DataSource object created in section "Creating a Pooled DataDirect Data Source Object" on page 264).

Client applications that use this data source must perform a lookup using the registered JNDI name (jdbc/SparkyOracle in this example).

```
//************************************************************************
//
// This code creates a data source and registers it to a JNDI naming
// service. This data source uses the PooledConnectionDataSource
// implementation provided by the DataDirect com.ddtek.pool package.
//
// This data source refers to a previously registered pooled data source.
//
// This data source registers its name as <jdbc/SparkyOracle>.
// Client applications using pooling must perform a lookup for this name.
//
//************************************************************************

// From the DataDirect connection pooling package:
import com.ddtek.pool.PooledConnectionDataSource;

import javax.sql.*;
import java.sql.*;
import javax.naming.*;
import javax.naming.directory.*;
import java.util.Hashtable;

public class PoolMgrDataSourceRegisterJNDI
{
   public static void main(String argv[])
   {
      try {
         // Set up data source reference data for naming context:
         // ----------------------------------------------------
         // Create a pooling manager's class instance that implements
         // the interface DataSource
         PooledConnectionDataSource ds = new PooledConnectionDataSource();

         ds.setDescription("Sparky Oracle - Oracle Data Source");
         // Refer to a previously registered pooled data source to access
         // a ConnectionPoolDataSource object
         ds.setDataSourceName("jdbc/PooledSparkyOracle");

         // The pool manager will be initiated with 5 physical connections
         ds.setInitialPoolSize(5);

         // The pool maintenance thread will make sure that there are 5
         // physical connections available
         ds.setMinPoolSize(5);

         // The pool maintenance thread will check that there are no more
         // than 10 physical connections available
         ds.setMaxPoolSize(10);

         // The pool maintenance thread will wake up and check the pool
         // every 20 seconds
         ds.setPropertyCycle(20);

         // The pool maintenance thread will remove physical connections
         // that are inactive for more than 300 seconds
         ds.setMaxIdleTime(300);

         // Set tracing off since we choose not to see output listing
         // of activities on a connection
         ds.setTracing(false);

         // Set up environment for creating initial context
         Hashtable env = new Hashtable();
         env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.fscontext.RefFSContextFactory");
         env.put(Context.PROVIDER_URL, "file:c:\\JDBCDataSource");
         Context ctx = new InitialContext(env);

         // Register the data source to JNDI naming service
         // for application to use
         ctx.bind("jdbc/SparkyOracle", ds);

      } catch (Exception e) {
         System.out.println(e);
```

```
            return;
        }

    } // Main
} // class PoolMgrDataSourceRegisterJNDI
```

Connecting to a Data Source

Whether connection pooling is used does not affect application code. It does not require any code changes to the application because the application performs a lookup on a JNDI name of a previously registered data source. If the data source specifies a connection pooling implementation during JNDI registration (as described in "Creating a Data Source Using the DataDirect Connection Pool Manager" on page 266), the client application benefits from faster connections through connection pooling.

The following example shows code that can be used to look up and use a JNDI-registered data source for connections. You specify the JNDI lookup name for the data source you created (as described in "Creating a Data Source Using the DataDirect Connection Pool Manager" on page 266).

```
//********************************************************************
//
// Test program to look up and use a JNDI-registered data source.
//
// To run the program, specify the JNDI lookup name for the
// command-line argument, for example:
//
//    java  TestDataSourceApp  <jdbc/SparkyOracleSequeLink>
//
//********************************************************************
import javax.sql.*;
import java.sql.*;
import javax.naming.*;
import java.util.Hashtable;
```

```
public class TestDataSourceApp
{
   public static void main(String argv[])
   {
      String strJNDILookupName = "";

      // Get the JNDI lookup name for a data source
      int nArgv = argv.length;
      if (nArgv != 1) {
         // User does not specify a JNDI lookup name for a data source,
         System.out.println(
            "Please specify a JNDI name for your data source");
         System.exit(0);
      } else {
         strJNDILookupName = argv[0];
      }

      DataSource ds = null;
      Connection con = null;
      Context ctx = null;
      Hashtable env = null;

      long nStartTime, nStopTime, nElapsedTime;

      // Set up environment for creating InitialContext object
      env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY,
         "com.sun.jndi.fscontext.RefFSContextFactory");
      env.put(Context.PROVIDER_URL, "file:c:\\JDBCDataSource");

      try {
         // Retrieve the DataSource object that bound to the logical
         // lookup JNDI name
         ctx = new InitialContext(env);
         ds = (DataSource) ctx.lookup(strJNDILookupName);
      } catch (NamingException eName) {
         System.out.println("Error looking up " +
            strJNDILookupName + ": " +eName);
         System.exit(0);
      }
      int numOfTest = 4;
      int [] nCount = {100, 100, 1000, 3000};

      for (int i = 0; i < numOfTest; i ++) {
         // Log the start time
         nStartTime = System.currentTimeMillis();
         for (int j = 1; j <= nCount[i]; j++) {
            // Get Database Connection
            try {
               con = ds.getConnection("scott", "tiger");
               // Do something with the connection
               // ...

               // Close Database Connection
               if (con != null) con.close();
            } catch (SQLException eCon) {
               System.out.println("Error getting a connection: " + eCon);
               System.exit(0);
            } // try getConnection
         } // for j loop

         // Log the end time
         nStopTime = System.currentTimeMillis();

         // Compute elapsed time
         nElapsedTime = nStopTime - nStartTime;
         System.out.println("Test number " + i + ": looping " +
            nCount[i] + " times");
         System.out.println("Elapsed Time: " + nElapsedTime + "\n");
      } // for i loop

      // All done
      System.exit(0);

   } // Main
} // TestDataSourceApp
```

NOTE: The DataDirect Connect *for* JDBC DataSource object class implements the DataSource interface for non-pooling in addition to ConnectionPoolDataSource for pooling. To use a non-pooling data source, use the JNDI name registered in the example code in "Creating a Pooled DataDirect Data Source Object" on page 264 and run the TestDataSourceApp. For example:

```
java TestDataSourceApp jdbc/PooledSparkyOracle
```

Terminating the Pool Manager

The Pool Manager requires notification of application termination to close connections in the pool properly. If your application runs on JRE 1.3 or higher, notification occurs automatically, so the application does not have to send notification. For JRE 1.2, the application must explicitly notify the pool manager of termination, using a special close method defined by DataDirect. This is illustrated in the following:

```
if (ds instanceof com.ddtek.pool.PooledConnectionDataSource){
            com.ddtek.pool.PooledConnectionDataSource pcds =
(com.ddtek.pool.PooledConnectionDataSource) ds;
            pcds.close();
            }
```

Description of DataDirect Connection Pool Manager Classes

This section describes the methods used by the DataDirect Connection Pool Manager classes, PooledConnectionDataSourceFactory and PooledConnectionDataSource.

PooledConnectionDataSourceFactory

This class is used to create a pooled connection data source object from a Reference object that is stored in a naming or directory service. The methods in this object are typically invoked by a JNDI service provider. They are not normally invoked by a user application. The following table describes the methods in the PooledConnectionDataSourceFactory class.

PooledConnectionDataSourceFactory ()

Methods	**Description**
java.lang.Object getObjectInstance (java.lang.Object refObj, javax.naming.Name name, javax.naming.Context nameCtx, java.util.Hashtable env)	Creates a PooledConnectDataSource object from a Reference object that is stored in a naming or directory service. This is an implementation of the method of the same name defined in the javax.naming.spi.ObjectFactory interface. Refer to the Javadoc for this interface for a description of this method.
NOTE: The following methods are inherited from the java.lang.Object class: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait. Refer to the java.lang.Object class for a description of these methods.	

PooledConnectionDataSource

This class is used to create a pooled connection data source.

PooledConnectionDataSource

Methods	**Description**
void close ()	Closes the connection pool. The application should verify that no new connections are created after this method is called.
java.sql.Connection getConnection ()	Obtains a physical connection from the connection pool.

PooledConnectionDataSource

Methods	**Description**
java.sql.Connection getConnection (java.lang.String user, java.lang.String password)	Obtains a physical connection from the connection pool, where *user* is the user requesting the connection and *password* is the password for the connection.
java.lang.String getDataSourceName ()	Obtains the JNDI name that is used to look up the DataDirect Connect *for* JDBC pooled data source object referenced by this data source.
java.lang.String getDescription ()	Obtains the description of the data source.
int getInitialPoolSize ()	Obtains the value of the initial pool size, which is the number of physical connections created when the connection pool is initialized.
int getLoginTimeout ()	Obtains the value of the login timeout, which is the time allowed for the database login to be validated.
java.io.PrintWriter getLogWriter ()	Obtains the writer to which trace information will be logged.
int getMaxIdleTime ()	Obtains the value of the maximum idle time, which is the time a physical connection can remain idle in the connection pool before it is removed from the connection pool.
int getMaxPoolSize ()	Obtains the value of the maximum pool size, which is the maximum number of physical connections allowed within a single pool at any time. When this number is reached, additional connections that would normally be placed in a connection pool are closed.
int getMinPoolSize ()	Obtains the value of the minimum pool size, which is the minimum number of physical connections that will be kept open.
int getPropertyCycle ()	Obtains the property cycle, which specifies how often the pool maintenance thread wakes up and checks the connection pool.

PooledConnectionDataSource

Methods	Description
javax.naming.Reference getReference ()	Obtains a javax.naming.Reference object for this data source. The Reference object contains all the state information needed to recreate an instance of this data source using the PooledConnectionDataSourceFactory object. This method is typically called by a JNDI service provider when this data source is bound to a JNDI naming service.
boolean isTracing ()	Indicates whether tracing is enabled.
void setDataSourceName (java.lang.String dataSourceName)	Sets the JNDI name, which is used to look up the DataDirect Connect *for* JDBC pooled data source object referenced by this data source.
void setDescription (java.lang.String description)	Sets the description, where *description* is the description of the data source.
void setInitialPoolSize (int initialPoolSize)	Sets the value of the initial pool size, which is the number of physical connections created when the connection pool is initialized.
void setLoginTimeout (int i)	Sets the value of the login timeout, where *i* is the login timeout, which is the time allowed for the database login to be validated.
void setLogWriter java.io.PrintWriter printWriter)	Sets the writer, where *printWriter* is the writer to which the stream will be printed.
void setMaxIdleTime (int maxIdleTime)	Sets the value of the maximum idle time, which is the time a physical connection can remain idle in the connection pool before it is removed from the connection pool.
void setMaxPoolSize (int maxPoolSize)	Sets the value of the maximum pool size, which is the maximum number of physical connections allowed within a single pool at any time. When this number is reached, additional connections that would normally be placed in a connection pool are closed.

PooledConnectionDataSource **Methods**	**Description**
void setMinPoolSize (int minPoolSize)	Sets the value of the minimum pool size, which is the minimum number of physical connections that will be kept open.
void setPropertyCycle (int propertyCycle)	Sets the property cycle, which specifies how often the pool maintenance thread wakes up and checks the connection pool.
void setTracing (boolean tracing)	Enables or disables tracing where *tracing* is true or false. If true, tracing is enabled; if false, it is disabled.
NOTE: The following methods are inherited from the java.lang.Object class: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait. Refer to the java.lang.Object class for a description of these methods.	

Index