Pervasive PSQL v11

JDBC Driver Guide

Developing Applications Using the Pervasive JDBC Driver

Pervasive Software Inc.
12365 Riata Trace Parkway
Building B
Austin, TX 78727 USA

Telephone: 512 231 6000 or 800 287 4383
Fax: 512 231 6010
Email: database@pervasive.com
Web: http://www.pervasivedb.com

disclaimer

PERVASIVE SOFTWARE INC. LICENSES THE SOFTWARE AND DOCUMENTATION PRODUCT TO YOU OR YOUR COMPANY SOLELY ON AN "AS IS" BASIS AND SOLELY IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF THE ACCOMPANYING LICENSE AGREEMENT. PERVASIVE SOFTWARE INC. MAKES NO OTHER WARRANTIES WHATSOEVER, EITHER EXPRESS OR IMPLIED, REGARDING THE SOFTWARE OR THE CONTENT OF THE DOCUMENTATION; PERVASIVE SOFTWARE INC. HEREBY EXPRESSLY STATES AND YOU OR YOUR COMPANY ACKNOWLEDGES THAT PERVASIVE SOFTWARE INC. DOES NOT MAKE ANY WARRANTIES, INCLUDING, FOR EXAMPLE, WITH RESPECT TO MERCHANTABILITY, TITLE, OR FITNESS FOR ANY PARTICULAR PURPOSE OR ARISING FROM COURSE OF DEALING OR USAGE OF TRADE, AMONG OTHERS.

trademarks

JDBC Driver Guide

January 2013

Contents

Tables

About This Manual

This manual is your guide to developing Pervasive PSQL applications that use the Java API for executing SQL statements (JDBC).

Who Should Read This Manual

This document is designed for the user who is developing Pervasive PSQL applications using the Java API for executing SQL statements (JDBC).

Pervasive Software Inc. would appreciate your comments and suggestions about this manual. As a user of our documentation, you are in a unique position to provide ideas that can have a direct impact on future releases of this and other manuals. If you have comments or suggestions for the product documentation, post your request at the Community Forum on the Pervasive Software Web site.

Manual Organization

This manual begins with an overview of the new features, then provides links to chapters containing additional details where appropriate. *JDBC Driver Guide* is divided into the following sections:

- Chapter 1—Introduction to the Pervasive JDBC Driver

  This chapter introduces the JDBC API for Java programming using the Pervasive relational database engine.

- Chapter 2—Programming with the Pervasive JDBC 2 Driver

  This chapter introduces the JDBC API for Java programming using the Pervasive relational database engine.

- Chapter 3—JDBC API Reference

  This chapter provides a link to the JDBC API reference.

This manual also contains an index.

Typographical Conventions

The documentation uses the following typographical conventions.

Convention	Explanation
bold	Bold typeface usually indicates elements of a graphical user interface, such as menu names, dialog box names, commands, options, buttons, and so forth. Bold typeface is also applied occasionally in a standard typographical use for emphasis.
italics	Italics indicate a variable that must be replaced with an appropriate value. For example, *user_name* would be replaced with an actual user name. Italics is also applied occasionally in a standard typographical use for emphasis, such as for a book title.
cAsE	Uppercase text is used typically to improve readability of code syntax, such as SQL syntax, or examples of code. Case is significant for some operating systems. For such instances, the subject content mentions whether literal text must be uppercase or lowercase.
`monospace`	Monospace text is used typically to improve readability of syntax examples and code examples, to indicate results returned from code execution, or for text displayed on a command line. The text may appear uppercase or lowercase, depending on context.
', ", and “ ”	Straight quotes, both single and double, are used in code and syntax examples to indicate when a single or double quote is required. Curly double quotes are applied in the standard typographical use for quotation marks.
|	The vertical rule indicates an OR separator to delineate items for which you must choose one item or another. See explanation for angle brackets below.
[ ]	Square brackets indicate optional items. Code syntax not enclosed by brackets is required syntax.
< >	Angle brackets indicate that you must select one item within the brackets. For example, <yes | no> means you must specify either “yes” or “no.”
. . .	Ellipsis indicates that the preceding item can be repeated any number of times in succession. For example, [*parameter* . . .] indicates that *parameter* can be repeated. Ellipsis following brackets indicate the entire bracketed content can be repeated.
::=	The symbol ::= means one item is defined in terms of another. For example, a::=b means that item “a” is defined in terms of “b.”
%*string*%	A variable defined by the Windows operating system. *String* represents the variable text. The percent signs are literal text.
$*string*	An environment variable defined by the Linux operating system. *String* represents the variable text. The dollar sign is literal text.

chapter 1

Introduction to the Pervasive JDBC Driver

An Overview of Pervasive's support for JDBC Development

This chapter introduces you to Pervasive's JDBC interface. The chapter contains the following topics:

Pervasive PSQL JDBC Support

JDBC is the standard API that Java programmers use to develop database and Internet applications using Java. JDBC is included in Sun Microsystems's Java Developer Kit in versions 1.1 and higher. JDBC is a package that primarily consists of interfaces that developers can use to develop SQL based database applications using the Java programming language.

JDBC is the counterpart of ODBC in Java and is heavily influenced by ODBC and relational databases.

Detailed information on the JDBC API is available at java.sun.com.

JDBC Requirements

The Pervasive JDBC driver works in conjunction with Pervasive PSQL. You can use the Server or Workgroup engines.

JDBC Features

The following is a summary of features of the Pervasive JDBC driver:

- 100% Java certified
- JDBC 2 compliant, type 4 driver
- Supports thread safe operation
- Supports transactions isolation levels supported by the Pervasive PSQL engine, for example READ_COMMITTED, serializable
- Performs result set caching to reduce network access
- Supports binary data through the longvarbinary data type (2 GB limit)
- Supports long char data through the longvarchar data types (2 GB limit)
- Supports stored procedures with parameters
- Encrypts connection strings to provide security
- Transmits data flows between the server and client in a native binary format that is nontrivial to decode for added security

Pervasive JDBC 2 Driver

JDBC 2 API is supported in Pervasive PSQL. JDBC version 2 is included with JDK version 1.2 and higher. The Pervasive JDBC driver supports this JDBC 2 standard including optional packages.

Table 1 Summary of New Functionality with Pervasive JDBC driver

Feature Element	Notes
Connection Strings	Extra parameter added that allows you to specify which code page to use with the connection.
Improved cursor support	The driver now supports CONCUR_UPDATABLE, TYPE_SCROLL_INSENSITIVE, and TYPE_SCROLL_SENSITIVE
DataSource Interface Support	Register Pervasive PSQL databases in JNDI and your applications can be shielded from Pervasive-specific driver features.

Specifications

The Pervasive JDBC driver is a pure Java type 4 net protocol-based driver. It conforms to the core JDBC 2.x requirements. The driver is similar to the ODBC client driver in functionality and depends on the Pervasive PSQL Engine's ODBC Interface on the server side for the bulk of processing.

Upgrading from the Pervasive JDBC 1 Driver

If you have used the previous Pervasive JDBC 1 Driver, the following topics discuss changes that have occurred with the JDBC 2 Driver.

JDBC API Improvements

- Scrollable – ability to perform both relative and absolute positioning.
- Updateable – ability to insert, update, and delete without the need to execute more SQL.
- Dynamic & Static server-side cursors – ability to choose whether you want to see changes made by you or other users.
- Batch Updates – ability to queue up many operations and have them execute at once.

- Character Encoding support – allows you to change encoding per connection.

JDBC Optional Package Support

- DataSource interface - allows data source objects to be registered in JNDI.
- Connection Pooling support – complete implementation of the connection pooling interfaces.

Backward Compatibility

Pervasive JDBC version 2 is backward compatible. Applications compiled with previous releases of Pervasive JDBC drivers will work with the new driver without the need to recompile. Applets will need to change the jar file name from pervasiveJDBC.jar to pvjdbc2.jar in the HTML file.

The new driver is accessed through a different package name:

- com.pervasive.jdbc.v2.Driver will load a JDBC 2 driver
- pervasive.jdbc.PervasiveDriver will load the JDBC 1 driver

Class Names

With the Pervasive JDBC version 2 driver, class names have changed to comply with Sun recommended standard. All Pervasive classes now are prefaced by com.pervasive.jdbc.v2. In older versions, classes were prefaced by pervasive.jdbc.

Pervasive JDBC Driver Limitations

Unsupported APIs

Pervasive's JDBC driver does not support the following JDBC interfaces:

- Array
- Blob
- Clob
- Ref
- Struct
- SQLData
- SQLInput
- SQLOutput

These are not supported due to the fact the Pervasive PSQL engine does not currently support the underlying SQL 3 data types.

Driver Limitations

- You cannot use long data in "out" parameters
- The smallest actual fetch size is two rows.
- You cannot have an updateable result set with a join.
- You cannot have an updateable result set with a "group by."
- The JDBC driver will not store data in UnicodeBig or UnicodeLittle formats.

chapter

2

Programming with the Pervasive JDBC 2 Driver

An Overview of the JDBC 2 Functionality in Pervasive PSQL

The following are the sections found in this chapter:

How to Set Up your Environment

This section contains information about proper configuration for use of the JDBC interface.

Setting the CLASSPATH

So that Java applications and applets recognize the Pervasive PSQL JDBC Driver, set your CLASSPATH environment variable to include the pvjdbc2.jar, pvjdbc2x.jar, and jpscs.jar files. By default, these files are installed on Windows platforms in the *install_directory*\bin folder under Program Files. On Linux, the files are installed by default to /usr/local/psql/bin.

From Windows:

```
set CLASSPATH=%CLASSPATH%;<path to pvjdbc2.jar
   directory>/pvjdbc2.jar
set CLASSPATH=%CLASSPATH%;<path to pvjdbc2x.jar
   directory>/pvjdbc2x.jar
set CLASSPATH=%CLASSPATH%;<path to jpscs.jar directory>
   /jpscs.jar
```

From Linux:

```
export CLASSPATH=$CLASSPATH:<path to pvjdbc2.jar
   directory>/pvjdbc2.jar
export CLASSPATH=$CLASSPATH:<path to pvjdbc2x.jar
   directory>/pvjdbc2x.jar
export CLASSPATH=$CLASSPATH:<path to jpscs.jar
   directory>/jpscs.jar
```

Setting the SYSTEM PATH

If you connect to the database engine using shared memory or IPX, the JDBC driver must find pvjdbc2.dll. Ensure that your PATH variable on Windows contains the location of the DLL:

```
set PATH=%PATH%;<path to pvjdbc2.dll directory>
```

If you connect to the database engine using sockets, typically no DLL is required.

Loading the Pervasive JDBC Driver into the Java Environment

After setting the CLASSPATH, you can now reference the Pervasive JDBC Driver from your Java application. You do this by using the java.lang.Class class:

```
Class.forName("com.pervasive.jdbc.v2.Driver");
```

IPv6 Environments

If you want to use the Pervasive PSQL JDBC driver in an IPv6-only environment, we recommend that you also use Java JRE 1.7. You may encounter issues with license counts or client-tracking problems if your application uses Java JRE 1.6 or earlier in an IPv6-only environment.

You may also encounter issues with license counts for the following combination of conditions:

1. A machine runs multiple applications using the Pervasive PSQL JDBC driver and the applications connect to the database engine with a combination of IPv4 and IPv6 addresses.
2. The SYSTEM PATH on the machine does not include the location of pvjdbc2.dll. See also Setting the SYSTEM PATH.

Specifying a Data Source

After loading the PervasiveDriver class into your Java environment, you need to pass a URL-style string to the java.sql.DriverManager class to connect to a Pervasive PSQL database. The syntax for URL for the Pervasive JDBC driver is:

```
jdbc:pervasive://<machinename>:<portnumber>/
    <datasource>
```

<machinename>	is the host name or IP address of the machine that runs the Pervasive DB Server.
<portnumber>	is the port on which the Pervasive dB Server is listening. By default it is 1583.
<datasource>	is the name of the ODBC DSN on the Pervasive database server that the application intends to use.

For example, if your Pervasive PSQL engine is on a machine named DBSERV, and you wish to connect to the DEMODATA database, your URL would look like this (assuming the server is configured to use the default port):

```
jdbc:pervasive://DBSERV/DEMODATA
```

So to connect to the database using the DriverManager class, you would use the syntax:

```
Connection conn =
    DriverManager.getConnection("jdbc:pervasive://
    DBSERV:1583/DEMODATA", loginString, passwordString);
```

where "loginString" is the string that represents a user login and "passwordString" is the string that represents a user password.

Note The Pervasive PSQL engine must be running on the specified host for JDBC applets and applications to access data.

Developing JDBC Applets

To develop web based applications using JDBC, you need to place the JDBC jar file in the codebase directory containing the applet classes.

For example, if you are developing an application called MyFirstJDBCapplet, you need to place the pvjdbc2.jar file in the directory containing the MyFirstJDBCapplet class. For example, it might be C:\inetpub\wwwroot\myjdbc\.

This enables the client web browser to be able to download the JDBC driver over the network and connect to the database.

You also need to put the archive parameter within the <APPLET> tag. For example:

```
<applet CODE="MyFirstJDBCapplet.class"
          ARCHIVE="pvjdbc2.jar" WIDTH=641 HEIGHT=554>
```

Note that the Pervasive PSQL engine must be running on the Web server that hosts the applet.

JDBC Programming Tasks

This section highlights important concepts for JDBC programming.

Connection String Overview

The JDBC driver requires a URL to connect to a database. The URL syntax for the Pervasive JDBC driver is:

jdbc:pervasive://*machinename*:*port number*/*datasource*[;encoding=;encrypt=;encryption=]

machinename is the host name or ip address of the machine that runs the Pervasive PSQL server.

port number is the port on which the Pervasive PSQL server is listening. By default it is 1583.

datasource is the name of the ODBC engine data source on the Pervasive PSQL server that the application intends to use.

encoding= is the character encoding, which allows you to filter data you read through a specified code page so that it is formatted and sorted correctly.

encrypt= specifies whether the JDBC driver should use encrypted network communications, also known as wire encryption.

encryption= specifies the minimum level of encryption allowed by the JDBC driver.

Note A Pervasive PSQL v11 SP3 engine needs to be running on the specified host to run JDBC applications.

Connection String Elements

The following shows how to connect to a Pervasive PSQL database using JDBC:

Driver Classpath

```
com.pervasive.jdbc.v2
```

Statement to Load Driver

```
Class.forName("com.pervasive.jdbc.v2.Driver");
```

URL

```
jdbc:pervasive://server:port/
DSN[;encoding=;encrypt=;encryption=]
```

or

```
jdbc:pervasive://server:port/
DSN[?pvtranslate=&encrypt=&encryption=]
```

Table 2 Connection String Elements

Argument	Description
server	The server name using an ID or a URL.
port	The default port for the relational interface is 1583. If no port is specified, the default is used.
DSN	Name of the DSN to set up on the server using regular ODBC methods.
encoding	See Using Character Encoding
encrypt	Determines whether the JDBC driver should use encrypted network communications, also known as wire encryption. (See Wire Encryption in *Advanced Operations Guide*.) Values: **always**, **never** If this option is not specified, the driver reflects the server's setting, the equivalent of the value "if needed." If the value **always** is specified, the JDBC driver uses encryption or else return an error if wire encryption is not allowed by the server. If the value **never** is specified, the JDBC driver does not use encryption and returns an error if wire encryption is required by the server. To use wire encryption with the JDBC driver, another JAR file is required to be in your classpath. This JAR, jpscs.jar, is installed by default and uses Java Cryptography Extensions (JCE).
encryption	Determines the minimum level of encryption allowed by the JDBC driver. Values: **low**, **medium**, **high** Default: **medium** These values correspond to 40-bit, 56-bit, and 128-bit encryption, respectively. The following example specifies that the JDBC driver uses UTF-8 encoding, always requires encryption and requires at least the low level of encryption or it returns an error code. `jdbc:pervasive://host/demodata?encoding=UTF-8&encrypt=always&encryption=low`

JDBC Connection String Example

The following code shows how to connect to a Pervasive database using the JDBC driver:

```
// Load the Pervasive PSQL JDBC driver
Class.forName("com.pervasive.jdbc.v2.Driver")

// Pervasive JDBC URL Syntax:
// jdbc:pervasive://<hostname or ip address > :
//   <port num (1583 by default)>/<odbc engine DSN>

String myURL = "jdbc:pervasive://127.0.0.1:1583/
    demodata";
 try
    {

// m_Connection =
    DriverManager.getConnection(myURL,username,
    password);
}
   catch(SQLException e)
    {
       e.printStackTrace();

       // other exception handling

}
```

Using Character Encoding

Java uses wide characters for strings. Character data must be translated to a code page for exchange with the database engine. Character data encoding is specified using the “encoding” attribute in the connection string passed to the driver manager.

Encoding Attribute

The encoding attribute specifies a particular code page to use for translating character data. If the encoding attribute is absent, the default operating system code page for the client machine is used. The assumption is that the client and server use the same operating system encoding.

Example of Using Character Encoding

```
public static void main(String[] args)
{
    //specify latin 2 encoding
    String url = "jdbc:pervasive://MYSERVR:1583/
SWEDISH_DB;encoding=cp850";
    try{
```

```
Class.forName("com.pervasive.jdbc.v2.Driver");
       Connection conn =
DriverManager.getConnection(url);
       Statement stmt = conn.createStatement();
       ResultSet rs = stmt.executeQuery("select *
from SwedishTable");
       rs.close();
       stmt.close();
       conn.close();
    }
    catch(Exception e)
    {
       e.printStackTrace();
    }
}
```

Notes on Character Encoding

If one database contains tables that were populated using two different encodings, two distinct connections need to be established. The database engine can identify only one encoding per connection, so each separate connection must specify its own encoding.

The Pervasive JDBC driver uses Java native support for code pages. The list of supported code pages can be obtained from the java.sun.com Web site.

Developing Web-based Applications

This section describes how to create web-based applications with the Pervasive JDBC driver.

Applets

To develop web based applications using JDBC, you need to place the JDBC jar file in the codebase directory containing the applet classes.

For example, if you are developing an application called MyFirstJDBCapplet, you need to place the `pvjdbc2.jar` file (or the pervasive jdbc package) in the directory containing the MyFirstJDBCapplet class. For example, it might be `C:\inetpub\wwwroot\myjdbc\`. This enables the client web browser to be able to download the JDBC driver over the network and connect to the database.

Also, if you use the JAR file, you need to put the archive parameter within the `<APPLET>` tag. For example,

```
<applet CODE="MyFirstJDBCapplet.class"
ARCHIVE="pvjdbc2.jar" WIDTH=641 HEIGHT=554>
```

Note The Pervasive PSQL engine must be running on the web server that hosts the applet.

Servlets and Java Server Pages

Servlets and Java Server Pages (JSP can be used to create web-based applications with the Pervasive JDBC Driver.

The following is a sample Java Server Page for displaying one table in the DEMODATA sample database included with Pervasive PSQL

```
<%@ page import="java.sql.*" %>
<%@ page import="java.util.*" %>

 <%
 Class.forName("com.pervasive.jdbc.v2.Driver");
 Connection con =
    DriverManager.getConnection("jdbc:pervasive://
    localhost:1583/DEMODATA");
 PreparedStatement stmt = con.prepareStatement("SELECT *
    FROM Course ORDER BY Name");
 ResultSet rs = stmt.executeQuery();
 %>
```

```
 <html>
 <head>
 <title>Pervasive PSQL JSP Sample</title>
 </head>
 <body>

<h1>Pervasive PSQL JSP Sample</h1>
<h2>Course table in DEMODATA database</h2>
<p>
This example opens the Course table from the DEMODATA
database included with Pervasive PSQL and
displays the contents of the table
</p>

 <table border=1 cellpadding=5>
 <tr>
 <th>Name</th>
 <th>Description</th>
 <th>Credit Hours</th>
 <th>Department Name</th>
 </tr>


 <% while(rs.next()) { %>
    <tr>
    <td><%= rs.getString("Name") %></td>
    <td><%= rs.getString("Description") %></td>
    <td><%= rs.getString("Credit_Hours") %></td>
    <td><%= rs.getString("Dept_Name") %></td>
    </tr>
 <% } %>


</table>

</body>
</html>
```

Information on Servlets and JSP

For more information about servlets and JSP, see Sun's Web site at java.sun.com.

JDBC 2.0 Standard Extension API

Because connection strings are vendor-specific, Sun specified a DataSource interface. It takes advantage of JNDI, which functions as a Java registry. The DataSource interface allows JDBC developers to create named databases. As a developer, you register the database in JNDI along with the vendor-specific driver information. Then, your JDBC applications can be completely database agnostic and be "pure JDBC."

The Pervasive JDBC driver now supports JDBC 2.0 Standard Extension API. Currently, the Pervasive JDBC driver supports the following interfaces

- javax.sql.ConnectionEvent
- javax.sql.ConnectionEventListener
- javax.sql.ConnectionPoolDataSource
- javax.sql.DataSource
- javax.sql.PooledConnection

Note These interfaces are packaged separately in pvjdbc2x.jar in order to keep the core JDBC API 100% pure java.

Although at this time Pervasive does not provide implementation of RowSet interfaces, Pervasive JDBC driver has been tested with Sun's implementation of RowSet interface.

DataSource

Sun has provided a way for application developers to write applications that are driver independent. By using DataSource interface and JNDI, applications can access data using standard methods and eliminate driver specific elements such as connection strings. In order to use DataSource interface, a database has to be registered with a JNDI service provider. An application can then access it by name.

The following is an example of using the DataSource interface:

```
// this code will have to be executed by the
// administrator in order to register the
// DataSource.
// This sample uses Sun's reference JNDI
// implementation
```

```
public void registerDataSources()
    {
// this example uses the JNDI file system
// object as its registry

    Context ctx;
    jndiDir = "c:\\jndi";

        try
        {
          Hashtable env = new Hashtable (5);
         env.put (Context.INITIAL_CONTEXT_FACTORY,

"com.sun.jndi.fscontext.RefFSContextFactory");

      env.put(Context.PROVIDER_URL, jndiDir);
          ctx = new InitialContext(env);
        }
        catch (Exception e)
        {
         System.out.println(e.toString());
        }

        //register demodata as regular data source
        com.pervasive.jdbc.v2.DataSource ds = new
com.pervasive.jdbc.v2.DataSource();
        String dsName = "";

        try
        {
           // Set the user name, password, driver
type and network protocol
            ds.setUser("administrator");
            ds.setPassword("admin");
            ds.setPortNumber("1583");
            ds.setDatabaseName("DEMODATA");
            ds.setServerName("127.0.0.1");

ds.setDataSourceName("DEMODATA_DATA_SOURCE");
       ds.setEncoding("cp850");
            dsName = "jdbc/demodata";

       // Bind it
        try
        {
            ctx.bind(dsName,ds);
             System.out.println("Bound data source
[" + dsName + "]");
        }
```

```
        catch (NameAlreadyBoundException ne)
        {
            System.out.println("Data source [" +
dsName + "] already bound");
        }
        catch (Throwable e)
          {
           System.out.println("Error in JNDI
binding occurred:");
           throw new Exception(e.toString());
        }
      }
    }
}

//in order to use this DataSource in application the
following code needs to be executed

public DataSource lookupDataSource(String ln)
throws SQLException
    {
      Object ods = null;
   Context ctx;

        try
        {
          Hashtable env = new Hashtable (5);
         env.put (Context.INITIAL_CONTEXT_FACTORY,

"com.sun.jndi.fscontext.RefFSContextFactory");

// this will create the jndi directory
// and return its name
// if the directory does not already exist

String jndiDir = "c:\\jndi";

      env.put(Context.PROVIDER_URL, jndiDir);
          ctx = new InitialContext(env);
        }
        catch (Exception e)
        {
         System.out.println(e.toString());
        }
        try
        {
            ods = ctx.lookup(ln);
            if (ods != null)
             System.out.println("Found data source
[" + ln + "]");
```

```
            else
               System.out.println("Could not find
data source [" + ln + "]");
        }
        catch (Exception e)
        {
            throw new SQLException(e.toString());
        }

        return (DataSource)ods;
    }

// note that ConnectionPoolDataSource is
// handled similarly.
```

Connection and Concurrency

A single Pervasive JDBC connection can easily serve multiple threads. However, while the Connection may be thread-safe, the objects created by the Connection are not. For example, a user can create four threads. Each of these threads could be given their own Statement object (all created by the same Connection object). All four threads could be sending or requesting data over the same connection at the same time. This works because all four Statement objects have a reference to the same Connection object and their reading and writing is synchronized on this object. However, thread #1 cannot access the Statement object in thread #2 without this access being synchronized. The above is true for all other objects in the JDBC API.

Scrollable Result Sets

Scrollable result sets allow you to move forward and backward through a result set. This type of movement is classified as either relative or absolute. You can position absolutely on any scrollable result set by calling the methods `first()`, `last()`, `beforeFirst()`, `afterLast()`, and `absolute()`. Relative positioning is done with the methods `next()`, `previous()`, and `relative()`.

A scrollable result set can also either be updateable or read-only. This refers to whether or not you are able to make changes to the underlying database. Another term, sensitivity, refers to whether these changes are reflected in your current result set.

A sensitive result set will reflect any insert, updates, or deletes made to it. In Pervasive PSQL's case, an insensitive result set does not reflect any changes made to it (it is a static snapshot of the data). In other words, you do not see your updates or those made by anyone else.

Sensitive and insensitive result sets correspond to dynamic and static in ODBC, respectively. A sensitive result set reflects your own changes and can reflect others changes if the transaction isolation level is set to `READ_COMMITTED`. Transaction isolation is set using the Connection object. The result set type is set upon statement creation.

If your result set is insensitive, then it is possible to make calls to the method `getRow()` in order to determine your current row number. On an insensitive result set, you can also make calls to `isLast()`, `isFirst()`, `isBeforeFirst()`, and `isAfterLast()`. On a sensitive result set, you can only make calls to `isBeforeFirst()` and `isAfterLast()`. Also, on an insensitive result set, the driver will honor the fetch direction suggested by the user. The driver ignores the suggested fetch direction on sensitive result sets.

JDBC Programming Sample

The following example creates a connection to the database named “DB” on server “MYSERVER.” It then creates a statement object on that connection that is sensitive and updateable. Using the statement object a “SELECT” query is performed. Once the result set object is obtained a call to “absolute” is made in order to move to the fifth row. Once on the fifth row the second column is updated with an integer value of 101. Then a call to “updateRow” is made to actually make the update.

```
Class.forName("com.pervasive.jdbc.v2.Driver");
Connection conn=
DriverManager.getConnection("jdbc:pervasive://
MYSERVER:1583/DB");

Statement stmt =
conn.createStatement(ResultSet.TYPE_SCROLL_SENSITI
VE, ResultSet.CONCUR_UPDATABLE);

ResultSet rs =
m_stmt.executeQuery("SELECT * FROM mytable");


rs.absolute(5);
rs.updateInt(2, 101);
rs.updateRow();

rs.close();
stmt.close();
conn.close();
```

chapter

3

JDBC API Reference

The JDBC API is a standard interface to databases using the Java programming language.

This chapter discusses the following topics:

- JDBC API Reference
- JDBC Samples

JDBC API Reference

JDBC is a standard API that is documented on Sun Microsystem's web site. See the JDBC and the JDBC documentation content at java.sun.com, noting the API limitations of Pervasive's driver in Pervasive JDBC Driver Limitations.

Other useful sites for JDBC programming include Tomcat information at jakarta.apache.org and Apache information at www.apache.org.

For conceptual information on programming with Pervasive's JDBC driver, see the following topics:

- Introduction to the Pervasive JDBC Driver
- Programming with the Pervasive JDBC 2 Driver

JDBC Samples

There are JDBC samples included with the Pervasive PSQL SDK. Please see the SAMPLES directory under your Pervasive PSQL SDK installation directory. If you installed to the default location, this would be *file_path*\PSQL\SDK\SAMPLES\JDBC.

For default locations of Pervasive PSQL files, see Where are the Pervasive PSQL files installed? in *Getting Started With Pervasive PSQL*.

Index

L

O

P

R

S

T

U

W