

Progress
JDBC Driver Guide

© 2001 Progress Software Corporation. All rights reserved.

Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. This manual is also copyrighted and all rights are reserved. This manual may not, in whole or in part, be copied, photocopied, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Progress Software Corporation.

The information in this manual is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear in this document.

The references in this manual to specific platforms supported are subject to change.

Progress, Progress Results, Provision and WebSpeed are registered trademarks of Progress Software Corporation in the United States and other countries. Apptivity, AppServer, ProVision Plus, SmartObjects, IntelliStream, and other Progress product names are trademarks of Progress Software Corporation.

SonicMQ is a trademark of Sonic Software Corporation in the United States and other countries.

Progress Software Corporation acknowledges the use of Raster Imaging Technology copyrighted by Snowbound Software 1993-1997 and the IBM XML Parser for Java Edition.

© IBM Corporation 1998-1999. All rights reserved. U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Progress is a registered trademark of Progress Software Corporation and is used by IBM Corporation in the mark Progress/400 under license. Progress/400 AND 400® are trademarks of IBM Corporation and are used by Progress Software Corporation under license.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Any other trademarks and/or service marks contained herein are the property of their respective owners.

.

May 2001



Product Code: 4509
Item Number: 81078;9.1C

Contents

Preface . . . . . **vii**
Purpose vii
Audience vii
Organization of This Manual vii
How to Use This Manual viii
Typographical Conventions viii
Syntax Notation ix
Other Useful Documentation xii
Getting Started xii
4GL xiii
Database xiii
DataServers. xiii
SQL-92 xiv
Java and JDBC Reference xiv
SQL-92 Reference. xv

1. Introduction **1–1**
1.1 JDBC Overview 1–2
1.1.1 JDBC Architecture 1–2
1.1.2 Native-method Drivers 1–3
1.2 JDBC Compared to ODBC 1–4

2. Basic JDBC Driver Operations **2–1**
2.1 Configuring the JDBC Driver 2–2
2.1.1 Setting Environment Variables 2–2
2.2 Running a JDBC Application 2–3
2.2.1 Connecting to a Database 2–3
2.3 Running the Sample Program ProgressTest.class 2–6

3. JDBC Conformance Notes 3–1
3.1 Supported Data Types 3–2
3.2 Return Values for DatabaseMetaData 3–4
3.3 Error Messages 3–19

A. Sample Program Source Code A–1
A.1 ProgressTest.java A–2

Glossary Glossary–1

Index Index–1

Figures

Figure 1–1: Components of a JDBC Environment 1–3

Tables

Table 2–1:	JDBC Driver Libraries and Executable Files	2–2
Table 2–2:	JDBC Networking Library Environment Variables	2–3
Table 2–3:	Runtime System Shared Library Path Settings	2–3
Table 3–1:	Mapping Between Java and JDBC Data Types	3–2
Table 3–2:	Mapping Between JDBC and Java Data Types	3–3
Table 3–3:	Mapping Between SQL-92 and Java Data Types	3–3
Table 3–4:	Return Values for DatabaseMetaData Methods	3–4

Preface

Purpose

This guide provides an overview of the Java Database Connectivity (JDBC) interface and the Progress SQL-92 JDBC Driver, which provides access to the Progress database from applications that support JDBC. It describes how to set up and use the driver and details the driver's support for the JDBC interface. The Progress SQL-92 JDBC Driver is supplied by MERANT Inc., formerly known as INTERSOLV.

Audience

This book is designed as a guide for anyone who wants to use the Progress SQL-92 JDBC driver to access a Progress Version 9 database from a Java application.

Organization of This Manual

Chapter 1, "Introduction"

Introduces the Progress SQL-92 JDBC driver and describes how it works.

Chapter 2, "Basic JDBC Driver Operations"

Describes basic JDBC Driver operations: setup, required software, using the sample program, connecting to databases, and managing transactions.

Chapter 3, "JDBC Conformance Notes"

Details the information returned by the JDBC driver to the DatabaseMetadata methods and lists the supported SQL and corresponding Java data types.

Appendix A, "Sample Program Source Code"

Provides a source-code listing of the ProgressTest.java sample Java application that uses the JDBC Driver.

"Glossary"

Defines terms introduced in this guide.

How to Use This Manual

This book assumes that you have successfully installed the Progress database as described in the *Installation Notes Progress Version 9 for Windows* or the *Installation Notes Progress Version 9 for UNIX*.

Typographical Conventions

This manual uses the following typographical conventions:

- **Bold typeface** indicates:
  - Commands or characters that the user types
  - That a word carries particular weight or emphasis
- *Italic typeface* indicates:
  - Progress variable information that the user supplies
  - New terms
  - Titles of complete publications
- `Monospaced typeface` indicates:
  - Code examples
  - System output
  - Operating system filenames and pathnames

The following typographical conventions are used to represent keystrokes:

- Small capitals are used for Progress key functions and generic keyboard keys.

  END-ERROR, GET, GO
  ALT, CTRL, SPACEBAR, TAB

- When you have to press a combination of keys, they are joined by a dash. You press and hold down the first key, then press the second key.

  CTRL-X

- When you have to press and release one key, then press another key, the key names are separated with a space.

  ESCAPE H
  ESCAPE CURSOR-LEFT

Syntax Notation

The syntax for each SQL-92 component follows a set of conventions:

- Uppercase words are keywords. Although they are always shown in uppercase, you can use either uppercase or lowercase when using them in an expression or a statement.

  In this example, GRANT, RESOURCE, DBA, and TO are keywords:

  SYNTAX

  ```
  GRANT { RESOURCE, DBA } TO user_name [, user_name] ... ;
  ```

- Italics identify options or arguments that you must supply. These options can be defined as part of the syntax or in a separate syntax identified by the name in italics. In the GRANT statement you must supply at least one *user_name*.

- Square brackets (**[ ]**) around an item indicate that the item, or a choice of one of the enclosed items, is optional.

In this syntax example the first *user_name* is required, and additional *user_name* specifications are optional:

SYNTAX

```
GRANT { RESOURCE, DBA } TO user_name [, user_name] ... ;
```

NOTE: The ellipsis (**...**) indicates repetition, as shown in a following description.

- Braces (**{ }**) around an item indicate that the item, or a choice of one of the enclosed items, is required.

  In the GRANT example, you must specify either RESOURCE or DBA or both, and at least one *user_name*. Any *user_name* specifications after the first are optional:

  SYNTAX

  ```
  GRANT { RESOURCE, DBA } TO user_name [, user_name] ... ;
  ```

  In some cases, braces are not a syntax notation, but part of the language.

  For example, this excerpt from an ODBC application invokes a stored procedure using the ODBC syntax { call *procedure_name* (*param*) }, where braces and parentheses are part of the language:

  SYNTAX

  ```
  proc1( param, "{ call proc2 (param) }", param);
  ```

- A vertical bar (**|**) indicates a choice. In the CREATE SYNONYM example you must specify a *table_name*, *view_name*, or *synonym* but you can only choose one. Note that in all SQL-92 syntax, if you specify the optional *owner_name* qualifier, there must not be a blank between the period separator and the *table_name* or *view_name* or *synonym*:

  SYNTAX

  ```
  CREATE [ PUBLIC ] SYNONYM synonym
  FOR [ owner_name. ] { table_name | view_name | synonym } ;
  ```

In the DELETE FROM example, you must specify one of *table_name* or *view_name*:

SYNTAX

```
DELETE FROM [ owner_name. ] { table_name | view_name }
  [ WHERE search_condition ] ;
```

- Ellipses (**. . .**) indicate that you can choose one or more of the preceding items. If a group of items is enclosed in braces and followed by ellipses, you must choose one or more of those items. If a group of items is enclosed in brackets and followed by ellipses, you can optionally choose one or more of those items.

  In the ORDER BY example, you must include one expression *(expr)* or column position *(posn)*, and you can optionally specify the sort order as ascending (ASC) or descending (DESC). You can specify additional expressions or column positions for sorting within a sorted result set. The SQL-92 engine orders the rows on the basis of the first *expr* or *posn*. If the values are the same, the second *expr* or *posn* is used in the ordering:

  SYNTAX

  ```
  ORDER BY { expr | posn } [ ASC | DESC ]
    [ , [ { expr | posn } [ ASC | DESC ] ] ... ]
  ```

  In the GRANT example, you must include one *user_name*, but you can optionally include more. Note that each subsequent *user_name* must be preceded by a comma:

  SYNTAX

  ```
  GRANT { RESOURCE, DBA } TO user_name [, user_name] ... ;
  ```

- In many examples, the syntax is too long to place in one horizontal row. In such cases, **optional** items appear individually bracketed in multiple rows in order, left-to-right and top-to-bottom. This order generally applies, unless otherwise specified. **Required** items also appear on multiple rows in the required order, left-to-right and top-to-bottom. In cases where grouping and order might otherwise be ambiguous, braced (required) or bracketed (optional) groups clarify the groupings.

In this example, CREATE VIEW is followed by several optional items:

SYNTAX

```
CREATE VIEW [ owner_name. ] view_name
 [ ( column_name [, column_name ] ... ) ]
  AS [ ( ] query_expression [ ) ] [ WITH CHECK OPTION ] ;
```

```
install-dir/dlc/bin/proenv
```

Other Useful Documentation

This section lists Progress Software Corporation documentation that you might find useful. Unless otherwise specified, these manuals support both Windows and Character platforms and are provided in electronic documentation format on CD-ROM.

Getting Started

Progress Installation and Configuration Guide Version 9 for UNIX

A manual that describes how to install and set up Progress Version 9.1 for the UNIX operating system.

Progress Installation and Configuration Guide Version 9 for Windows

A manual that describes how to install and set up Progress Version 9.1 for all supported Windows and Citrix MetaFrame operating systems.

Progress Version 9 Product Update Bulletin

A guide that provides a brief description of each new feature of the release. The booklet also explains where to find more detailed information in the documentation set about each new feature.

Progress Master Index and Glossary for Windows and *Progress Master Index and Glossary for Character* (Hard copy only)

Platform-specific master indexes and glossaries for the Progress hard-copy documentation set.

Progress Startup Command and Parameter Reference

A reference manual that describes the Progress startup commands and parameters in alphabetical order.

Welcome to Progress (Hard copy only)

A booklet that explains how Progress software and media are packaged. An icon-based map groups the documentation by functionality, providing an overall view of the documentation set. *Welcome to Progress* also provides descriptions of the various services Progress Software Corporation offers.

4GL

Progress Internationalization Guide

A guide to developing Progress applications for markets worldwide. The guide covers both internationalization—writing an application so that it adapts readily to different locales (languages, cultures, or regions)—and localization—adapting an application to different locales.

Database

Progress Database Design Guide

A guide that uses a sample database and the Progress Data Dictionary to illustrate the fundamental principles of relational database design. Topics include relationships, normalization, indexing, and database triggers.

Progress Database Administration Guide and Reference

This guide describes Progress database administration concepts and procedures. The procedures allow you to create and maintain your Progress databases and manage their performance.

DataServers

Progress DataServer Guides

These guides describe how to use the DataServers to access non-Progress databases. They provide instructions for building the DataServer modules, a discussion of programming considerations, and a tutorial. Each DataServer has its own guide, such as: the *Progress DataServer for Microsoft SQL Server Guide*, the *Progress DataServer for ODBC Guide*, the *Progress DataServer for ORACLE Guide*, or the *Progress/400 Product Guide*.

SQL-92

Progress Embedded SQL-92 Guide and Reference

A guide to Progress Embedded SQL-92 for C, including step-by-step instructions for building ESQL-92 applications and reference information about all Embedded SQL-92 Preprocessor statements and supporting function calls. This guide also describes the relationship between ESQL-92 and the ANSI standards upon which it is based.

Progress ODBC Driver Guide

A guide to the ODBC interface and the Progress SQL-92 ODBC driver. It describes how to set up and use the driver and details the driver's support for the ODBC interface.

Progress SQL-92 Guide and Reference

A user guide and reference for programmers who use Progress SQL-92. It includes information on all supported SQL-92 statements, SQL-92 Data Manipulation Language components, SQL-92 Data Definition Language components, and Progress functions. The guide describes how to use the Progress SQL-92 Java classes and how to create and use Java stored procedures and triggers.

Java and JDBC Reference

The Java™ Tutorial

Mary Campione and Kathy Walrath
`http://java.sun.com/docs/books/tutorial/index.html`

The Java™ Language Environment: A White Paper

James Gosling & Henry McGilton
`http://java.sun.com/docs/white/langenv/`

The JDBC™ Database Access API

`http://www.javasoft.com/products/jdbc/index.html`

Hamilton, Graham, Rick Cattell, Maydene Fisher. 1997. *JDBC™ Database Access with Java™*. Reading, MA: Addison Wesley.

SQL-92 Reference

A Guide to the SQL Standard

Date, C.J., with Hugh Darwen. 1997. Reading, MA: Addison Wesley.

Understanding the New SQL: A Complete Guide

Melton, Jim (Digital Equipment Corporation) and Alan R. Simon. 1993. San Francisco: Morgan Kaufmann Publishers.

1

Introduction

The Progress SQL-92 JDBC (Java Database Connectivity) Driver is supplied by MERANT Inc., formerly known as Intersolv. It runs on Windows, Window NT, and UNIX environments. It is installed when you install the Progress SQL-92 Client Access product.

The Progress SQL-92 JDBC Driver provides access to Progress Version 9.1 databases from Java applications. It is a type 2 driver that implements the JDBC 2.0 core API and extended 2.0 features, including batch updates, connection pooling, and updateable result sets.

This chapter provides a brief introduction to JDBC. For more information, please refer to the Java and JDBC Reference section in the Preface of this book.

1.1 JDBC Overview

JDBC allows applications to connect to any database using the same set of Java interfaces. Those interfaces allow applications to embed standard Structured Query Language (SQL) statements that update and retrieve data in the database.

Because the Java interfaces and SQL syntax are independent of any particular database implementation, JDBC makes feasible applications that can connect to different database environments without any modification.

1.1.1 JDBC Architecture

JDBC insulates Java applications from variations in database-access implementations through the JDBC API. Instead of using calls to vendor-specific interfaces, JDBC applications use the JDBC API. The JDBC 2.0 API is distributed as two packages. *java.sql* contains the Core API of JDBC 2.0 and *javax.sql* contains the Standard Extension API to support JDBC 2.0 functionality. These packages are included with the JavaSoft JDK (Java Developer Kit).

The Progress SQL-92 JDBC Driver consists of a series of Java class files contained in the Java package *com.progress.sql.jdbc* :

- *com.progress.sql.jdbx* contains the optional JDBC 2.0 extension package for use with applications developed for the JDBC 2.0 API that use a data source interface to establish database connections.
- Operating system dependent libraries contain the class files required to invoke Progress SQL-92 specific CLI calls from within Java.

Java database applications that are developed with JDBC 2.0 make calls to the JDBC API that are then managed by the JDBC driver manager. The JDBC driver manager can support multiple drivers connecting to different databases. When an application tries to connect to a particular database, the driver manager loads the appropriate JDBC driver and routes subsequent calls through the driver. Figure 1–1 shows the different components of a JDBC environment.

Java database applications that are developed using the JDBC 2.0 Standard Extension API, can locate and connect to a database using Java Naming and Directory Interface (JNDI) and DataSource objects.

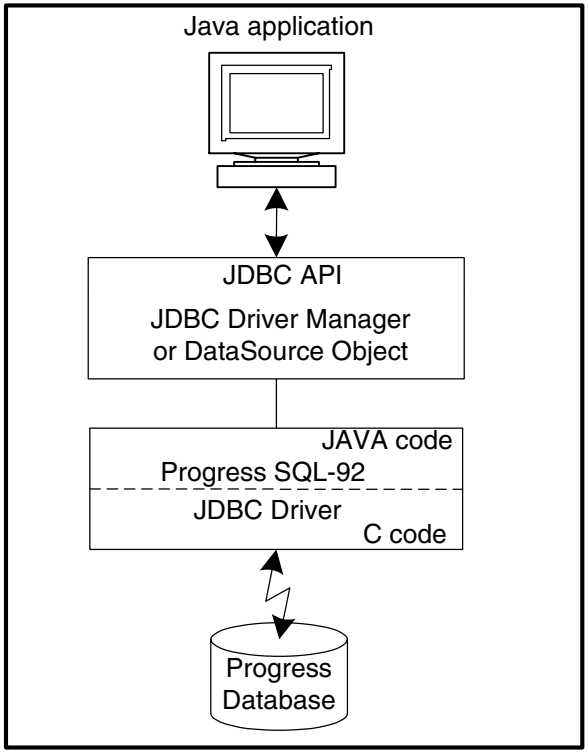

Figure 1–1: Components of a JDBC Environment

1.1.2 Native-method Drivers

Native-method, Type 2 drivers contain Java code that calls native C++ methods already implemented by database vendors. The Progress SQL-92 JDBC Driver is a native-method , Type 2 JDBC driver that enhances existing networking code installed on Java clients.

1.2 JDBC Compared to ODBC

JDBC is to Java what Microsoft's Open Database Connectivity (ODBC) interface is to the C language. Both JDBC and ODBC:

DOCUMENTATION COMMENT: Rephrase this paragraph.

- Provide a vendor-independent API that allows the same application to connect to different vendors' databases and retrieve and update data using standard SQL statements.
- Adopt the architecture of imposing a driver manager between applications and vendor-supplied drivers that translate between the standard API and a vendor's proprietary implementation.
- Are based on the X/Open SQL call-level interface specification.

2

Basic JDBC Driver Operations

This chapter describes how to configure the Progress SQL-92 JDBC Driver.

2.1 Configuring the JDBC Driver

In order to configure the JDBC Driver, your system must have access to a system (UNIX or Windows) where the Progress SQL-92 JDBC networking libraries and executable files have been installed, as described in the *Progress Installation and Configuration Guide Version 9 for UNIX* or the *Progress Installation and Configuration Guide Version 9 for Windows*. Table 2–1 lists the name and location of the Progress SQL-92 JDBC networking libraries and executable files by platform.

Table 2–1: JDBC Driver Libraries and Executable Files

Platform	File Name and Location
Windows	`%DLC%\bin\JdbcProgress.dll.`
Solaris and Compaq Tru64	`$DLC/lib/libJdbcProgress.so`
IBM AIX	`$DLC/lib/libJdbcProgress.a`
Hewlett Packard	`$DLC/lib/libJdbcProgress.sl`

NOTE: $DLC (on UNIX) and %DLC% (on Windows) are environment variables that point to the directory in which Progress is installed. Using $DLC and %DLC% allows you to use the same script to set environment variables relative to where Progress is installed, regardless of where it is installed on a particular system.

When you install Progress with a development license the class file `progress.zip` is installed in the `java` subdirectory of the directory where Progress is installed. However, when you install Progress with a deployment license the class file `jdbc.zip` is installed in the `java` subdirectory of the directory where Progress is installed. In order to configure the JDBC driver your system must have access to a system where one of these class files has been installed.

2.1.1 Setting Environment Variables

To configure the JDBC driver set the following environment variables:

- On both Windows and UNIX, set the CLASSPATH environment variable to point to the directory containing `progress.zip` or `jdbc.zip`.
- Table 2–2 lists the environment variables you must set and the location to which you must set them (the directory containing the JDBC Driver networking library), by platform.

Table 2–2: JDBC Networking Library Environment Variables

Platform	Environment Variable	Location
Windows, NT	PATH	%DLC%\bin.
Solaris and Compaq Tru64	LD_LIBRARY_PATH	$DLC/lib
IBM AIX	LIBPATH	$DLC/lib
Hewlett Packard	SHLIB_PATH	$DLC/lib

2.2 Running a JDBC Application

In order to run a JDBC application on UNIX you must set the system shared library path to include the runtime shared library DLC/lib as follows:

Table 2–3: Runtime System Shared Library Path Settings

Platform	System Shared Library Path Setting
Solaris and Compaq Tru64	LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$DLC/lib
IBM AIX	LIBPATH=$LIBPATH:$DLC/lib
Hewlett Packard	SHLIB_PATH=$SHLIB_PATH:DLC/lib

2.2.1 Connecting to a Database

JDBC applications must perform two steps to connect to a database:

1. Load the JDBC Driver.
2. Connect to the Driver.

Loading the JDBC Driver Using Class.forName

The Class.forName method takes as its argument the fully qualified class name of the JDBC Driver. If it finds the class, the method loads and links the class, and returns the Class object representing the class. The fully qualified class name for the Progress SQL-92 JDBC Driver is com.progress.sql.jdbc.JdbcProgressDriver.

To load the JDBC Driver, use it as the argument to the Class.forName method:

SYNTAX

```
// Load the driver
Class.forName ("com.progress.sql.jdbc.JdbcProgressDriver");
```

Using DriverManager.GetConnection

To connect to a Progress database through the JDBC Driver, an application specifies:

- A database connection string in the form of a JDBC URL
- User authentication detail (User ID and password)

Applications specify this information as arguments to the DriverManager.GetConnection method.

Java URL Connection String

DriverManager.GetConnection requires at least one argument, a character string specifying a database connection URL. For the Progress SQL-92 JDBC Driver, the URL has the following syntax:

SYNTAX

```
jdbc:JdbcProgress:T:host-name:port#:database-name [User-ID password]
```

The URL string has the following components:

`jdbc:JdbcProgress:T`

An identifying subprotocol string for the JDBC Driver.

`host-name`

The name of the system where the Progress SQL-92 JDBC data source resides.

`port#`

Port number or service name to be used for the connection.

`database-name`

The physical file name of the Progress database.

User-ID

The User ID for connecting to the database.

Password

The password for connecting to the database.

EXAMPLE

When passed to DriverManager.GetConnection, the URL `jdbc.JdbcProgress:T:isis:2000:testdb` specifies that the JDBC Driver be used to connect to the database testdb on the server named isis.

User Authentication Detail

DriverManager.GetConnection accepts three variants of user authentication detail:

- User ID and password passed as two character-string arguments:

```
Connection con = DriverManager.getConnection (url, "fred",
"fredspasswd" );
```

- User ID and password passed as a single Properties object:

```
Connection con = DriverManager.getConnection (url, prop );
```

NOTE: The JDBC Driver expects the keys of the Properties object to be named user and password when it processes the object. Application code must use those names when it populates the Properties object:

```
prop.put("user", userid);
prop.put("password", passwd);
```

- Password omitted. On Windows only, the JDBC Driver can connect to the database with a null password:

```
Connection con = DriverManager.getConnection (url);
```

EXAMPLE

The following code sample loads the driver and connects to the database testdb, using the form of DriverManager.GetConnection that takes authentication information as a single Properties object:

```
String url = "jdbc:JdbcProgress:T:isis:2000:testdb";
String userid = "fred";
String passwd = "fredspasswd";

// Load the driver
Class.forName ("com.progress.sql.JdbcProgressDriver");

// Attempt to connect to a driver.  Each one
// of the registered drivers will be loaded until
// one is found that can process this URL.
java.util.Properties  prop = new java.util.Properties();
prop.put("user", userid);
prop.put("password", passwd);

Connection con = DriverManager.getConnection (url, prop);
```

2.3 Running the Sample Program ProgressTest.class

The Progress SQL-92 .class files include a test Java application, ProgressTest.class, that exercises the JDBC interfaces implemented by the JDBC Driver. The application connects to the specified database (or creates a default connection hard-coded in the application) and presents a limited interactive SQL interface.

ProgressTest.class is part of the package *com.progress.sql.jdbc* . The source file is `ProgressTest.java`. See Appendix A, “Sample Program Source Code,” for a listing of `ProgressTest.java`.

To run the sample program, qualify the class name with the package name as the argument to the Java interpreter.

EXAMPLE

For example, on Windows, the following command line supplies a connection URL to run ProgressTest on a host named ktwo, port-number 2000, connected to the database mydb. In this example no User ID or password is required:

```
C:\>java com.progress.sql.jdbc.ProgressTest
jdbc:JdbcProgress:T:ktwo:2000:mydb
```

3

JDBC Conformance Notes

This chapter details the Progress SQL-92 JDBC driver's support for the JDBC standard.

3.1 Supported Data Types

The Progress SQL-92 JDBC Driver supports standard JDBC mapping of JDBC data types to corresponding Java data types.

In the JDBC methods CallableStatement.getXXX and PreparedStatement.setXXX methods, XXX is a Java type:

- For setXXX methods, the driver converts the Java data type to the JDBC data type shown in Table 3–1 before sending it to the database.

Table 3–1: Mapping Between Java and JDBC Data Types

Java Data Type	JDBC Data Type
String	VARCHAR
java.math.BigDecimal	NUMERIC
Boolean	BIT
Byte	TINYINT
Short	SMALLINT
Int	INTEGER
float	REAL
double	DOUBLE
byte [ ]	VARBINARY
byte [ ]	LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

- For getXXX methods, the driver converts the JDBC data type returned by the database to the Java data type shown in Table 3–2 before returning it to the getXXX method.

Table 3–2: Mapping Between JDBC and Java Data Types

JDBC Data Type	Java Data Type
CHAR	String
VARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int

Table 3–3: Mapping Between SQL-92 and Java Data Types *(1 of 2)*

SQL-92 Data Type	Java Data Type
CHAR, VARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte[ ]
SMALLINT	Integer
INTEGER	Integer
REAL	Float
FLOAT	Float
DOUBLE PRECISION	Double

Table 3–3: Mapping Between SQL-92 and Java Data Types *(2 of 2)*

SQL-92 Data Type	Java Data Type
BINARY	byte[ ]
VARBINARY	byte[ ]
LONGVARBINARY	byte[ ]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

3.2 Return Values for DatabaseMetaData

Applications call methods of the DatabaseMetaData class to retrieve details about the JDBC support provided by the Progress SQL-92 JDBC driver.

Table 3–4 lists each method of the DatabaseMetaData class and shows what the Progress SQL-92 JDBC driver returns when an application calls the method. For details on the format and usage of each method, see the Java Core API documentation for your platform. Many of the methods return lists of information as an object of type ResultSet. Use the normal ResultSet methods, such as getString and getInt, to retrieve the data from the result sets.

Table 3–4: Return Values for DatabaseMetaData Methods *(1 of 15)*

Method	Description	Returns
allProceduresAreCallable()	Can all the procedures returned by getProcedures be called by the current user?	False
allTablesAreSelectable()	Can all the tables returned by getTable be SELECTed by the current user?	False
dataDefinitionCausesTransactionCommit()	Does a data definition statement within a transaction force the transaction to commit?	True

Table 3–4: Return Values for DatabaseMetaData Methods *(2 of 15)*

Method	Description	Returns
dataDefinitionIgnoredInTransactions()	Is a data definition statement within a transaction ignored?	False
doesMaxRowSizeIncludeBlobs()	Did getMaxRowSize() include LONGVARCHAR and LONGVARBINARY blobs?	False
getBestRowIdentifier (String, String, String, int, boolean)	Get a description of a table's optimal set of columns that uniquely identifies a row.	(result set)
getCatalogs()	Get the catalog names available in this database.	(result set)
getCatalogSeparator()	What is the separator between catalog and table names?	. (period)
getCatalogTerm()	What is the database vendor's preferred term for "catalog"?	database
getColumnPrivileges (String, String, String, String)	Get a description of the access rights for a table's columns.	(result set)
getColumns(String, String, String, String)	Get a description of table columns available in a catalog.	(result set)

Table 3–4: Return Values for DatabaseMetaData Methods *(3 of 15)*

Method	Description	Returns
getCrossReference (String, String, String, String, String, String)	Get a description of the foreign key columns in the foreign key table that reference the primary key columns of the primary key table (describe how one table imports another's key). This should normally return a single foreign key/primary key pair (most tables only import a foreign key from a table once). They are ordered by FKTABLE_CAT, FKTABLE_SCHEM, FKTABLE_NAME, and KEY_SEQ.	(result set)
getDatabaseProductName()	What is the name of this database product?	PROGRESS
getDatabaseProductVersion()	What is the version of this database product?	09.1C
getDefaultTransactionIsolation()	What is the database's default transaction isolation level? The values are defined in java.sql.Connection.	TRANSACTION _REPEATABLE_READ
getDriverMajorVersion()	What is the version of this JDBC driver?	1
getDriverMinorVersion()	What is the minor version of this JDBC driver?	1000

Table 3–4: Return Values for DatabaseMetaData Methods *(4 of 15)*

Method	Description	Returns
getDriverName()	What is the name of this JDBC driver?	JDBC-PROGRESS Windows: JdbcProgress.dll Solaris and Compaq Tru64: libJdbcProgressjdbc.so IBM AIX: libJdbcProgressjdbc.a Hewlett Packard: libJdbcProgressjdbc.sl
getDriverVersion()	What is the version of this JDBC driver?	1.1000 (03.60.0006)
getExportedKeys(String, String, String)	Get a description of the foreign key columns that reference a table's primary key columns (the foreign keys exported by a table).	(result set)
getExtraNameCharacters()	Get all the "extra" characters that can be used in unquoted identifier names (those beyond a–z, A–Z, 0–9 and _)	"_", "%"
getIdentifierQuoteString ()	What is the string used to quote SQL identifiers? This returns a space " " if identifier quoting is not supported.	" "
getImportedKeys(String, String, String)	Get a description of the primary key columns that reference a table's foreign key columns (the primary keys imported by a table).	(result set)
getIndexInfo (String, String, String, boolean, boolean)	Get a description of a table's indices and statistics.	(result set)
getMaxBinaryLiteralLength()	How many hex characters can you have in an inline binary literal?	31995

Table 3–4: Return Values for DatabaseMetaData Methods *(5 of 15)*

Method	Description	Returns
getMaxCatalogNameLength()	What is the maximum length of a catalog name?	32
getMaxCharLiteralLength()	What is the maximum length for a character literal?	31995
getMaxColumnNameLength()	What is the limit on column name length?	32
getMaxColumnsInGroupBy()	What is the maximum number of columns in a "GROUP BY" clause?	499
getMaxColumnsInIndex()	What is the maximum number of columns allowed in an index?	16
getMaxColumnsInOrderBy()	What is the maximum number of columns in an "ORDER BY" clause?	0
getMaxColumnsInSelect()	What is the maximum number of columns in a "SELECT" list?	500
getMaxColumnsInTable()	What is the maximum number of columns in a table?	500
getMaxConnections()	How many active connections can we have at a time to this database?	0
getMaxCursorNameLength()	What is the maximum cursor name length?	18
getMaxIndexLength()	What is the maximum length of an index (in bytes)?	113
getMaxProcedureNameLength()	What is the maximum length of a procedure name?	32

Table 3–4: Return Values for DatabaseMetaData Methods *(6 of 15)*

Method	Description	Returns
getMaxRowSize()	What is the maximum length of a single row?	31995 bytes
getMaxSchemaNameLength()	What is the maximum length allowed for a schema name?	32
getMaxStatementLength()	What is the maximum length of a SQL statement?	131000
getMaxStatements()	How many active statements can we have open at one time to this database?	100
getMaxTableNameLength()	What is the maximum length of a table name?	32
getMaxTablesInSelect()	What is the maximum number of tables in a SELECT?	250
getMaxUserNameLength()	What is the maximum length of a user name?	32
getNumericFunctions()	Get a comma-separated list of math functions.	ABS, ACOS, ASIN, ATAN, ATAN2, CEILING, COS, DEGREES, EXP, FLOOR, LOG10, MOD, PI, POWER, RADIANS, RAND, ROUND, SIGN,SIN, SQRT, TAN
getPrimaryKeys(String, String, String)	Get a description of a table's primary key columns.	(result set)
getProcedureColumns (String, String, String, String)	Get a description of a catalog's stored procedure parameters and result columns.	(result set)
getProcedures(String, String, String)	Get a description of stored procedures available in a catalog.	(result set)

Table 3–4: Return Values for DatabaseMetaData Methods *(7 of 15)*

Method	Description	Returns
getProcedureTerm()	What is the database vendor's preferred term for "procedure?"	"procedure"
getSchemas()	Get the schema names available in this database.	(result set)
getSchemaTerm()	What is the database vendor's preferred term for "schema?"	Owner
getSearchStringEscape()	This is the string that can be used to escape '_' or '%' in the string pattern style catalog search parameters.	"\"
getSQLKeywords()	Get a comma-separated list of all a database's SQL keywords that are NOT also SQL-92 keywords.	NAME, AREA
getStringFunctions()	Get a comma-separated list of string functions.	ASCII, CHAR, CONCAT, DIFFERENCE, INSERT, LCASE, LEFT, LENGTH, LOCATE, LOCATE2, LTRIM, REPEAT, REPLACE, RIGHT, RTRIM, SPACE, SUBSTRING, UCASE
getSystemFunctions()	Get a comma-separated list of system functions.	USERNAME, IFNULL, DBNAME
getTablePrivileges(String, String, String)	Get a description of the access rights for each table available in a catalog.	(result set)
getTables(String, String, String, String [])	Get a description of tables available in a catalog.	(result set)
getTableTypes()	Get the table types available in this database.	(result set)

Table 3–4: Return Values for DatabaseMetaData Methods *(8 of 15)*

Method	Description	Returns
getTimeDateFunctions()	Get a comma-separated list of time and date functions.	CURDATE, CURTIME, DAYNAME, DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, MONTH, QUARTER, WEEK, YEAR, HOUR, MINUTE, SECOND, MONTHNAME, NOW, TIMESTAMPADD, TIMESTAMPDIFF
getTypeInfo()	Get a description of all the standard SQL types supported by this database.	(result set)
getURL()	What is the URL for this database?	(the URL)
getUserName()	What is our user name as known to the database?	(user name)
getVersionColumns(String, String, String)	Get a description of a table's columns that are automatically updated when any value in a row is updated.	(result set)
isCatalogAtStart()	Does a catalog appear at the start of a qualified table name? Otherwise it appears at the end.	False
isReadOnly()	Is the database in read-only mode?	False
nullPlusNonNullIsNull()	Are concatenations between NULL and non-NULL values NULL? A JDBC-Compliant driver always returns true.	True

Table 3–4: Return Values for DatabaseMetaData Methods *(9 of 15)*

Method	Description	Returns
nullsAreSortedAtEnd()	Are NULL values sorted at the end regardless of sort order?	False
nullsAreSortedAtStart()	Are NULL values sorted at the start regardless of sort order?	False
nullsAreSortedHigh()	Are NULL values sorted high?	False
nullsAreSortedLow()	Are NULL values sorted low?	True
storesLowerCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in lowercase?	False
storesLowerCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in lowercase?	False
storesMixedCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in mixed case?	False
storesMixedCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in mixed case?	True
storesUpperCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case insensitive and store them in uppercase?	True

Table 3–4: Return Values for DatabaseMetaData Methods *(10 of 15)*

Method	Description	Returns
storesUpperCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case insensitive and store them in uppercase?	False
supportsAlterTableWithAddColumn()	Is "ALTER TABLE" with add column supported?	False
supportsAlterTableWithDropColumn()	Is "ALTER TABLE" with drop column supported?	False
supportsANSI92EntryLevelSQL()	Is the ANSI92 entry level SQL grammar supported? All JDBC-compliant drivers must return true.	True
supportsANSI92FullSQL()	Is the ANSI92 full SQL grammar supported?	False
supportsANSI92IntermediateSQL()	Is the ANSI92 intermediate SQL grammar supported?	False
supportsCatalogsInDataManipulation()	Can a catalog name be used in a data manipulation statement?	False
supportsCatalogsInIndexDefinitions()	Can a catalog name be used in an index definition statement?	False
supportsCatalogsInPrivilegeDefinitions()	Can a catalog name be used in a privilege definition statement?	False
supportsCatalogsInProcedureCalls()	Can a catalog name be used in a procedure call statement?	False
supportsCatalogsInTableDefinitions()	Can a catalog name be used in a table definition statement?	False

Table 3–4: Return Values for DatabaseMetaData Methods *(11 of 15)*

Method	Description	Returns
supportsColumnAliasing()	Is column aliasing supported? If so, the SQL AS clause can be used to provide names for computed columns or to provide alias names for columns as required.	True
supportsConvert()	Is the CONVERT function between SQL types supported?	True
supportsConvert(int, int)	Is CONVERT between the given SQL types supported?	True
supportsCoreSQLGrammar()	Is the ODBC Core SQL grammar supported?	True
supportsCorrelatedSubqueries()	Are correlated subqueries supported? A JDBC-compliant driver always returns true.	False
supportsDataDefinitionAndData ManipulationTransactions ()	Are both data definition and data manipulation statements within a transaction supported?	True
supportsDataManipulationTransactions Only()	Are only data manipulation statements within a transaction supported?	False
supportsDifferentTableCorrelationNames()	If table correlation names are supported, are they restricted to be different from the names of the tables?	True
supportsExpressionsInOrderBy()	Are expressions in "ORDER BY" lists supported?	True
supportsExtendedSQLGrammar()	Is the ODBC Extended SQL grammar supported?	True

Table 3–4: Return Values for DatabaseMetaData Methods *(12 of 15)*

Method	Description	Returns
supportsFullOuterJoins()	Are full nested outer joins supported?	False
supportsGroupBy()	Is some form of "GROUP BY" clause supported?	True
supportsGroupByBeyondSelect()	Can a "GROUP BY" clause add columns not in the SELECT provided it specifies all the columns in the SELECT?	True
supportsGroupByUnrelated()	Can a "GROUP BY" clause use columns not in the SELECT?	False
supportsIntegrityEnhancementFacility()	Is the SQL Integrity Enhancement Facility supported?	True
supportsLikeEscapeClause()	Is the escape character in "LIKE" clauses supported? A JDBC-compliant driver always returns true.	True
supportsLimitedOuterJoins()	Is there limited support for outer joins? (This will be true if supportFullOuterJoins is true.)	False
supportsMinimumSQLGrammar()	Is the ODBC Minimum SQL grammar supported? All JDBC-compliant drivers must return true.	True
supportsMixedCaseIdentifiers()	Does the database treat mixed case unquoted SQL identifiers as case sensitive and as a result store them in mixed case? A JDBC-compliant driver will always return false.	False

Table 3–4: Return Values for DatabaseMetaData Methods *(13 of 15)*

Method	Description	Returns
supportsMixedCaseQuotedIdentifiers()	Does the database treat mixed case quoted SQL identifiers as case sensitive and as a result store them in mixed case? A JDBC-compliant driver will always return true.	False
supportsMultipleResultSets()	Are multiple ResultSets from a single execute supported?	False
supportsMultipleTransactions()	Can multiple transactions be open at once (on different connections)?	True
supportsNonNullableColumns()	Can columns be defined as non-nullable? A JDBC-compliant driver always returns true.	True
supportsOpenCursorsAcrossCommit()	Can cursors remain open across commits?	True
supportsOpenCursorsAcrossRollback()	Can cursors remain open across rollbacks?	True
supportsOpenStatementsAcrossCommit()	Can statements remain open across commits?	True
supportsOpenStatementsAcrossRollback()	Can statements remain open across rollbacks?	True
supportsOrderByUnrelated()	Can an "ORDER BY" clause use columns not in the SELECT?	False
supportsOuterJoins()	Is some form of outer join supported?	True
supportsPositionedDelete()	Is positioned DELETE supported?	True

Table 3–4: Return Values for DatabaseMetaData Methods *(14 of 15)*

Method	Description	Returns
supportsPositionedUpdate()	Is positioned UPDATE supported?	True
supportsSchemasInDataManipulation()	Can a schema name be used in a data manipulation statement?	True
supportsSchemasInIndexDefinitions()	Can a schema name be used in an index definition statement?	True
supportsSchemasInPrivilegeDefinitions()	Can a schema name be used in a privilege definition statement?	True
supportsSchemasInProcedureCalls()	Can a schema name be used in a procedure call statement?	True
supportsSchemasInTableDefinitions()	Can a schema name be used in a table definition statement?	True
supportsSelectForUpdate()	Is SELECT for UPDATE supported?	True
supportsStoredProcedures()	Are stored procedure calls using the stored procedure escape syntax supported?	True
supportsSubqueriesInComparisons()	Are subqueries in comparison expressions supported? A JDBC-compliant driver always returns true.	True
supportsSubqueriesInExists()	Are subqueries in 'exists' expressions supported? A JDBC-compliant driver always returns true.	True

Table 3–4: Return Values for DatabaseMetaData Methods *(15 of 15)*

Method	Description	Returns
supportsSubqueriesInIns()	Are subqueries in 'in' statements supported? A JDBC-compliant driver always returns true.	True
supportsSubqueriesInQuantifieds()	Are subqueries in quantified expressions supported? A JDBC-compliant driver always returns true.	True
supportsTableCorrelationNames()	Are table correlation names supported? A JDBC-compliant driver always returns true.	True
supportsTransactionIsolationLevel(int)	Does the database support the given transaction isolation level?	True (for all four transaction levels)
supportsTransactions ()	Are transactions supported? If not, commit is a no-op and the isolation level is TRANSACTION_NONE.	True
supportsUnion()	Is SQL UNION supported?	True
supportsUnionAll()	Is SQL UNION ALL supported?	True
usesLocalFilePerTable()	Does the database use a file for each table?	False
usesLocalFiles()	Does the database store tables in a local file?	False

EXAMPLE

The following example shows an excerpt from ProgressTest.java that illustrates calling methods of DatabaseMetadata:

```
Connection con = DriverManager.getConnection ( url, prop);

                                .
                                .
                                .

            // Get the DatabaseMetaData object and display
            // some information about the connection

            DatabaseMetaData dma = con.getMetaData ();

            o.println("\nConnected to " + dma.getURL());
            o.println("Driver       " +
             dma.getDriverName());
            o.println("Version      " +
              dma.getDriverVersion());
```

3.3 Error Messages

The error messages generated by the driver, along with associated SQLSTATE and Progress SQL-92 error code values, are documented in the *Progress SQL-92 Guide and Reference*.

A

Sample Program Source Code

This appendix provides the Java source code for the sample program ProgressTest.java.

A.1 ProgressTest.java

You can change the connection URL to match your own Progress data source, then use ProgressTest.java to test your JDBC connection. You can also cut and paste all or part of the code into your own Java application. This sample program is located at $DLC/src/samples/sql92.

ProgressTest.java *(1 of 15)*

```
/*************************************************************/
/* Copyright (c) 1984-2000 by Progress Software Corporation  */
/*                                                           */
/* All rights reserved.  No part of this program or document */
/* may be  reproduced in  any form  or by  any means without */
/* permission in writing from Progress Software Corporation. */
/*************************************************************/

//----------------------------------------------------------------------
//
// Module:ProgressTest.java
//
// Description:Test program for JDBC API interface.  This java application
//  will connect to a Progress JDBC driver, issue a select statement
//  and display all result columns and rows.
//




import java.net.URL;
import java.sql.*;
import java.io.*;


class  ProgressTest
{
    protected static final boolean debugFlag = true;
    protected static final PrintStream o = System.out;
    protected static final String ColString = "COLUMNS";
    protected static final String TblString = "TABLES";
    protected static final String ViewString = "VIEWS";
    protected static final String CallString = "CALL";
    protected static final String AutocommitString = "AUTOCOMMIT";
    protected static final int MAX_SQLSTMTLEN = 1000;

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

ProgressTest.java *(2 of 15)*

```
{
            String url    = null;
            String userid = null;
            String passwd = null;
            String query  = null;

            switch (args.length)
            {
            case 0:
                //url   = "jdbc:JdbcProgress:T:isis:testdb";
                url   = "jdbc:JdbcProgress:T:isis:5000:testdb";
                userid= "sysprogress";
                passwd= "dummy";
                break;
            case 1:
                url   = args[0];
                userid= "sysprogress";
                passwd= "dummy";
                break;
            case 2:
                url   = args[0];
                userid= args[1];
                passwd= "dummy";
                break;
            case 3:
                url   = args[0];
                userid= args[1];
                passwd= args[2];
                break;
            default:
                System.out.println("\nUsage: java/jview ProgressTest " +
                                   " [[[<url>]  <userid>] <passwd>] ");

                System.exit(0);
            }
```

ProgressTest.java *(3 of 15)*

```
// Load the driver

Class.forName ("com.progress.sql.jdbc.JdbcProgressDriver");

 java.io.PrintStream      pStream = null;

 if (debugFlag)
 {
      //  Create  a PrintStream using System.out
     pStream = new java.io.PrintStream(System.out, true);
 }
 else
 {
     //  Create  PrintStream using a file.
     java.io.FileOutputStream outFile   =
         new java.io.FileOutputStream("JdbcProgress.log");
     pStream   = new java.io.PrintStream(outFile, true);
 }

 // Enable JDBC tracing
 DriverManager.setLogStream(pStream);


 // Attempt to connect to a driver.  Each one
 // of the registered drivers will be loaded until
 // one is found that can process this URL

 java.util.Properties  prop = new java.util.Properties();
 prop.put("user", userid);
 prop.put("password", passwd);

 // We have to add any other options as additional
 // properties in the prop argument.
 // e.g., prop.put\("Caller", "ProgressTest"\);

 Connection con = DriverManager.getConnection (
           url, prop);

 // If we were unable to connect, an exception
 // would have been thrown.  So, if we get here,
 // we are successfully connected to the URL
```

ProgressTest.java *(4 of 15)*

```
            // Check for, and display and warnings generated
            // by the connect.

             checkForWarning (con.getWarnings ());


             // o.println("\nConnected to " + url);

             // Get the DatabaseMetaData object and display
             // some information about the connection

             DatabaseMetaData dma = con.getMetaData ();

              o.println("\nConnected to " + dma.getURL());
              o.println("Driver       " +
                  dma.getDriverName());
              o.println("Version      " +
                  dma.getDriverVersion());
              o.println("");
              if (con.getAutoCommit())
                  o.println("Autocommit is on");
                else
                  o.println("Autocommit is off");
                o.println("");

              byte[] bArray = new byte[MAX_SQLSTMTLEN + 1];
              boolean           rs_exists = false;
              ResultSet         rs   = null;
              PreparedStatement pstmt = null;
              CallableStatement callstmt = null;

             while (true)
             {
                 o.print ("ProTest> ");
                 o.flush();
                int l = System.in.read (bArray, 0, MAX_SQLSTMTLEN);

                 query = new String (bArray, 0, 0, l);

                 query = query.trim();

                 if (query.endsWith(";"))
                 {
                     query = query.substring(0, query.length() - 1);
                     query = query.trim();
                 }
```

ProgressTest.java *(5 of 15)*

```
                if (query.length() == 0)
                     continue;

                 if (query.equalsIgnoreCase("quit"))
                     break;

                 // Execute the statement.

                 try
                 {
                     if (tablesCmd(query))
                     {
                         String[] types = {"TABLE","SYSTEM TABLE"};
                       String   t_patrn= query.substring(TblString.length());
                         t_patrn = t_patrn.trim();

                         if (t_patrn.length() == 0)
                             rs = dma.getTables("","","%",types);
                         else
                             rs = dma.getTables("","",t_patrn,types);
                         rs_exists = true;
                     } else if (viewsCmd(query) == true)
                     {
                        String[] types = {"VIEW"};
                          String   v_patrn=
query.substring(ViewString.length());
                         v_patrn = v_patrn.trim();

                         if (v_patrn.length() == 0)
                             rs = dma.getTables("","","%",types);
                         else
                             rs = dma.getTables("","",v_patrn,types);
                         rs_exists = true;
                     } else if (autocommitCmd(query) == true)
```

ProgressTest.java *(6 of 15)*

```
{
                        String[] types = {"VIEW"};
                        String   a_patrn= query.substring(
AutocommitString.length());
                        a_patrn = a_patrn.trim();

                        if (a_patrn.equalsIgnoreCase("off"))
                            con.setAutoCommit( false );
                        else if (a_patrn.equalsIgnoreCase("on"))
                            con.setAutoCommit( true );
                        else o.println( "Form is autocommit ( on | off )");
                        if (con.getAutoCommit())
                            o.println("Autocommit is on");
                        else
                            o.println("Autocommit is off");

                    } else if (columnsCmd(query) == true)
                    {
                           String   t_patrn=
query.substring(ColString.length());
                           t_patrn = t_patrn.trim();

                        if (t_patrn.length() == 0)
                        {
                            //rs = dma.getColumns("","","%","%");
                            o.println("Specify a table pattern");
                            continue;
                        }
                        else
                            rs = dma.getColumns("","",t_patrn,"%");
                        rs_exists = true;
                    } else if (callStmt(query) == true)
                    {
                        callstmt = con.prepareCall(query);
                        setCallParams(callstmt);
                        rs_exists = callstmt.execute();
                        if (rs_exists == true)
                        {
                            rs = callstmt.getResultSet();
                        } else
                        {
                            int updCount = callstmt.getUpdateCount();
                            o.println( "RowCount is " + updCount);
                        }
```

ProgressTest.java *(7 of 15)*

```
          } else if (query.equalsIgnoreCase("types"))
                    {
                        rs = dma.getTypeInfo();
                        rs_exists = true;
                    } else if (query.equalsIgnoreCase("commit"))
                    {
                        con.commit();
                        rs_exists = false;
                    } else if (query.equalsIgnoreCase("rollback"))
                    {
                        con.rollback();
                        rs_exists = false;
                    } else if (query.equalsIgnoreCase("?"))
                    {
                          o.println( "Commands are: ");
                          o.println( "        TABLES  [name]");
                          o.println( "        VIEWS   [name]");
                          o.println( "        COLUMNS [name]");
                          o.println( "        CALL    [name]");
                          o.println( "        types");
                          o.println( "        commit");
                          o.println( "        rollback");
                          o.println( "        autocommit on | off");
                          o.println( "        quit");
                          o.println( "        ?");
                          o.println( "        SQL statement to prepare,
execute");
                          o.println( "            -- statement can have
parms");

                    }else
                    {
                        pstmt = con.prepareStatement(query);
                        setParams(pstmt);
                        rs_exists = pstmt.execute();
                        if (rs_exists == true)
                        {
                            rs = pstmt.getResultSet();
                        } else
                        {
                            int updCount = pstmt.getUpdateCount();
                            o.println( "RowCount is " + updCount);
                        }
                    }
                    if (rs_exists == true)
```

ProgressTest.java *(8 of 15)*

```
            {
                        // Display all columns and rows from the result set
                        dispResultSet (rs);
                        rs.close();
                        rs_exists = false;
                    }
                }
                catch (SQLException ex)
                {
                    o.println(ex.getMessage());
                    continue;
                }
                // Close the statement
                if (pstmt != null)
                    pstmt.close();
                pstmt = null;

                if (callstmt != null)
                    callstmt.close();
                callstmt = null;
            }
            // Close the connection
            con.close();
        }
        catch (SQLException ex)
        {

            // A SQLException was generated.  Catch it and
            // display the error information.  Note that there
            // could be multiple error objects chained
            // together
          o.println ("*** (debugging) in sql exception block...");

            while (ex != null)
            {
                o.println ("SQLState: " +
                                    ex.getSQLState ());
                o.println ("Message:  " +
                                    ex.getMessage ());
                o.println ("VendorCode:   " +
                                    ex.getErrorCode ());
                ex = ex.getNextException ();
                o.println ("");
            }
```

ProgressTest.java *(9 of 15)*

```
          }
        catch (java.lang.Exception ex)
        {
            o.println ("*** (debugging) in general exception block...");
            // Got some other type of exception.  Dump it.
            ex.printStackTrace ();
        }

        finally
        {
            o.println ("@ProgressTest:finally");
        }

    }

    //---------------------------------------------------------------------
    // checkForWarning
    // Checks for and displays warnings.  Returns true if a warning
    // existed
    //---------------------------------------------------------------------

    private static boolean checkForWarning (SQLWarning warn)
        throws SQLException
        {
            boolean rc = false;
            // If a SQLWarning object was given, display the
            // warning messages.  Note that there could be
            // multiple warnings chained together

            if (warn != null)
            {
                o.println ("\n *** Warning ***\n");
                rc = true;
                while (warn != null)
                {
                    o.println ("SQLState: " +
         warn.getSQLState ());
                    o.println ("Message:  " +
         warn.getMessage ());
                    o.println ("Vendor:   " +
         warn.getErrorCode ());
                    o.println ("");
                    warn = warn.getNextWarning ();
                }
            }
            return rc;
        }
```

ProgressTest.java *(10 of 15)*

```
 //---------------------------------------------------------------------
 // dispResultSet
 // Displays all columns and rows in the given result set
 //---------------------------------------------------------------------

private static void dispResultSet (ResultSet rs)
    throws SQLException
{
        int i,j;

        // Get the ResultSetMetaData.  This will be used for
        // the column headings

        ResultSetMetaData rsmd = rs.getMetaData ();

        // Get the number of columns in the result set

        int numCols = rsmd.getColumnCount ();

 // Display column headings

        for (i=1; i<=numCols; i++)
        {
            if (i > 1)
                o.print(",");

            String label  = rsmd.getColumnLabel(i);
            o.print(label);
        }
        o.println();

        for (i=1; i<=numCols; i++)
        {
            if (i > 1)
                o.print("-");
            String label  = rsmd.getColumnLabel(i);
            for (j = 0; j < label.length(); j++)
                o.print("-");
        }
        o.println();
```

ProgressTest.java *(11 of 15)*

```
            // Display data, fetching until end of the result set

         while (rs.next ())
         {
             // Loop through each column, getting the
             // column data and displaying

             for (i=1; i<=numCols; i++)
             {
                 if (i > 1) System.out.print(",");
                 o.print(rs.getString(i));
             }
             o.println("");

             // Fetch the next result set row

         }
    }

     //---------------------------------------------------------------------
     // tablesCmd returns true if its a Tables Command
     // else false
     //---------------------------------------------------------------------
    private static boolean tablesCmd (String query)
    {

        String u_query = query.toUpperCase();
        if ((u_query.startsWith(TblString)) ||
             (u_query.startsWith(TblString + " ")))
            return true;
        return  false;
    }

      //---------------------------------------------------------------------
      // ViewsCmd returns true if its a Views Command
      // else false
      //---------------------------------------------------------------------
    private static boolean viewsCmd (String query)
    {

        String u_query = query.toUpperCase();
        if ((u_query.startsWith(ViewString)) ||
            (u_query.startsWith(ViewString + " ")))
            return true;
        return  false;
    }
```

ProgressTest.java *(12 of 15)*

```
      //--------------------------------------------------------------------
      // columnsCmd returns true if its a Columns Command
      // else false

//--------------------------------------------------------------------
    private static boolean columnsCmd (String query)
    {

        String u_query = query.toUpperCase();
        if ((u_query.startsWith(ColString)) ||
            (u_query.startsWith(ColString + " ")))
            return true;
        return  false;
    }

   //--------------------------------------------------------------------
   // CallStmt returns true if its a Procedure Call Statement
   // else false
   //--------------------------------------------------------------------
    private static boolean callStmt (String query)
    {

        String u_query = query.toUpperCase();
        if ((u_query.startsWith(CallString)) ||
            (u_query.startsWith(CallString + " ")))
            return true;
        return  false;
    }


     //--------------------------------------------------------------------
     // autocommitCmd returns true if its an autocommit Command
     // else false
     //--------------------------------------------------------------------
    private static boolean autocommitCmd (String query)
    {

        String u_query = query.toUpperCase();
        if ((u_query.startsWith(AutocommitString)) ||
            (u_query.startsWith(AutocommitString + " ")))
            return true;
        return  false;
    }
```

ProgressTest.java *(13 of 15)*

```
//--------------------------------------------------------------------
// setParams prompts for parameters and sets them.
//--------------------------------------------------------------------
 private static void  setParams (PreparedStatement stmt)
     throws SQLException
     {
         byte[] bArray  = new byte[MAX_SQLSTMTLEN + 1];
         String param = null;

        int paramCount = ((ProgressPreparedStatement)stmt).getNparams();

         if (paramCount > 0)
             o.println("Parameters Required = " + paramCount );

         try
         {
             for (int i = 0; i < paramCount; i++)
             {
                 o.print ("Type parameter no " + i +" : ");
                 int l = System.in.read (bArray, 0, MAX_SQLSTMTLEN);

                 param = new String (bArray, 0, 0, l);
                 param = param.trim();

                 if (param.equalsIgnoreCase("null"))
                     stmt.setNull(i+1, java.sql.Types.CHAR);
                 else
                     stmt.setString(i+1, param);
             }
         }

         catch (java.lang.Exception ex)
         {
             // Got some other type of exception.  Dump it.
             ex.printStackTrace ();
         }
     }
```

ProgressTest.java *(14 of 15)*

```
//--------------------------------------------------------------------
// setCallParams prompts for parameters and sets them.
//--------------------------------------------------------------------
 private static void  setCallParams (CallableStatement stmt)
     throws SQLException
     {
         byte[] bArray  = new byte[MAX_SQLSTMTLEN + 1];
         String param = null;

        int paramCount = ((ProgressCallableStatement)stmt).getNparams();

         if (paramCount > 0)
             o.println("Parameters Required = " + paramCount );

         try
         {
             for (int i = 0; i < paramCount; i++)
             {
                 o.println ("Parameter " + i );

                 while (true)
                 {
                     o.print ("    Type IN , OUT or INOUT : ");

                     int l = System.in.read (bArray, 0, MAX_SQLSTMTLEN);

                     String paramtype = new String (bArray, 0, 0, l);
                     paramtype = paramtype.trim();
                     paramtype = paramtype.toUpperCase();

                     if (paramtype.equalsIgnoreCase("IN"))
                     {
                         o.print ("    Type ParamValue : ");

                         l = System.in.read (bArray, 0, MAX_SQLSTMTLEN);

                        String paramvalue = new String (bArray, 0, 0, l);
                         paramvalue = paramvalue.trim();

                         if (paramvalue.equalsIgnoreCase("null"))
                             stmt.setNull(i+1, java.sql.Types.CHAR);
                         else
                             stmt.setString(i+1, paramvalue);

                         break;
```

ProgressTest.java *(15 of 15)*

```
   } else if (paramtype.equalsIgnoreCase("INOUT"))
                          {
                              o.print ("    Type ParamValue : ");

                              l = System.in.read (bArray, 0, MAX_SQLSTMTLEN);

                             String paramvalue = new String (bArray, 0, 0, l);
                              paramvalue = paramvalue.trim();

                              if (paramvalue.equalsIgnoreCase("null"))
                                  stmt.setNull(i+1, java.sql.Types.CHAR);
                              else
                                  stmt.setString(i+1, paramvalue);

                              stmt.registerOutParameter (i+1,

java.sql.Types.CHAR);
                              break;
                          } else if (paramtype.equalsIgnoreCase("OUT"))
                          {
                              stmt.registerOutParameter (i+1,

java.sql.Types.CHAR);
                              break;
                          }
                      }
                  }
              }

              catch (java.lang.Exception ex)
              {
                  // Got some other type of exception.  Dump it.
                  ex.printStackTrace ();
              }
          }
```

Glossary

Applet

A special kind of Java program whose compiled class files a Java-enabled browser can download from the Internet and run.

ASCII

American Standard Code for Information Interchange. A seven-bit character set that provides 128 character combinations.

Bytecode

Machine-independent code generated by the Java compiler and executed by the Java interpreter.

Client

In client/server systems, the part of the system that sends requests to servers and processes the results of those requests.

Collation

The rules used to control how character strings in a character set compare with each other. Each character set specifies a collating sequence that defines relative values of each character for comparing, merging, and sorting character strings. In addition, storage systems can define additional collations that SQL statements specify with the COLLATE clause in column definitions, column references, and character-string references.

Data Dictionary

Another term for **System Catalog**.

Data Source

See **ODBC Data Source.**

Driver Manager

See **JDBC Driver Manager**.

Form of Use

The storage format for characters in a character set. Some character sets, such as ASCII, require one byte (Octet) for each character. Others, such as **Unicode**, use two bytes, and are called multi-octet character sets.

Input Parameter

In a stored procedure specification, an argument that an application must pass when it calls the stored procedure. In an SQL statement, a **Parameter Marker** in the statement string that acts as a placeholder for a value that will be substituted when the statement executes.

Interface

In Java, a definition of a set of methods that one or more objects will implement. Interfaces declare only methods and constants, not variables. Interfaces provide multiple-inheritance capabilities.

Java Snippet

See **Snippet**.

JDBC

Java Database Connectivity. A part of the Java language that allows applications to embed standard SQL statements and access any database that implements a JDBC driver.

JDBC Driver

Database-specific software that receives calls from the JDBC driver manager, translates them into a form that the database can process, and returns data to the application.

JDBC Driver Manager

A Java class that implements methods to route calls from a JDBC application to the appropriate JDBC driver for a particular JDBC URL.

Metadata

Data that details the structure of tables and indexes in the database. The SQL engine stores metadata in the system catalog.

Octet

A group of eight bits. Synonymous with byte and often used in descriptions of character-set encoding format.

ODBC Data Source

In ODBC terminology, a specific combination of a database system, the operating system it uses, and any network software required to access it. Before applications can access a database through ODBC, you use the ODBC Administrator to add a data source—register

information about the database and an ODBC driver that can connect to it—for that database. More than one data source name can refer to the same database, and deleting a data source does not delete the associated database.

ODBC Driver

Vendor-supplied software that processes ODBC function calls for a specific data source. The driver connects to the data source, translates the standard SQL statements into syntax the data source can process, and returns data to the application. Progress SQL-92 includes an ODBC driver that provides access to proprietary storage systems underlying the ODBC server.

Output Parameter

In a stored procedure specification, an argument in which the stored procedure returns a value after it executes.

Package

A group of related Java classes and interfaces, like a class library in C++. The Java development environment includes many packages of classes that procedures can import. The Java run-time system automatically imports the *java.lang* package. Stored procedures must explicitly import other classes by specifying them in the IMPORT clause of a CREATE PROCEDURE statement.

Parameter Marker

A question mark (?) in a procedure call or SQL statement string that acts as a placeholder for an input or output parameter supplied at run time when the procedure executes. The CALL statement (or corresponding ODBC or JDBC escape clause) uses parameter markers to pass parameters to stored procedures, and the SQLIStatement, SQLPStatement, and SQLCursor objects use them within procedures.

Procedure Body

In a stored procedure, the Java code between the BEGIN and END keywords of a CREATE PROCEDURE statement.

Procedure Result Set

In a stored procedure, a set of data rows returned to the calling application. The number and data types of columns in the procedure result set are specified in the RESULT clause of the CREATE PROCEDURE statement. The procedure can transfer data from an **SQL Result Set** to the procedure result set or it can store data generated internally. A stored procedure can have only one procedure result set.

Procedure Specification

In a CREATE PROCEDURE statement, the clauses preceding the procedure body that specify the procedure name, any input and output parameters, any result set columns, and any Java packages to import.

Procedure Variable

A Java variable declared within the body of a stored procedure, as compared to a procedure **Input Parameter** or **Output Parameter**, which are declared outside the procedure body and are visible to the application that calls the stored procedure.

Repertoire

The set of characters allowed in a character set.

Result Set

In a **Stored Procedure**, either an **SQL Result Set** or a **Procedure Result Set**.

More generally, another term for **Result Table**.

Result Table

A virtual table of values derived from columns and rows of one or more tables that meet conditions specified by an SQL query expression.

Server

In client/server systems, the part of the system that receives requests from clients and responds with results to those requests.

Snippet

In a stored procedure, the sequence of Java statements between the BEGIN and END keywords in the CREATE PROCEDURE (or CREATE TRIGGER) statement. The Java statements become a method in a class the SQL engine creates and submits to the Java compiler.

SQLCA

SQL Communications area. A data structure that contains information about the execution status (success, error, or warning conditions) of the most recent SQL statement. The SQLCA includes an SQLCODE field. The SQLCA provides the same information as the SQL diagnostics area, but is not compliant with the SQL-92 standard. Progress SQL-92 supports both the SQLCA and the SQL diagnostics area.

SQLCODE

An integer status parameter whose value indicates the condition status returned by the most recent SQL statement. An SQLCODE value of zero means success, a positive value

means warning, and a negative value means an error status. SQLCODE is superseded by SQLSTATE in the SQL-92 standard. Applications declare either SQLSTATE or SQLCODE, or both. SQL returns the status to SQLSTATE or SQLCODE after execution of each SQL statement.

SQLSTATE

A five-character status parameter whose value indicates the condition status returned by the most recent SQL statement. SQLSTATE is specified by the SQL-92 standard as a replacement for the SQLCODE status parameter (which was part of SQL-89). SQLSTATE defines many more specific error conditions than SQLCODE, which allows applications to implement more portable error handling. Applications declare either SQLSTATE or SQLCODE, or both. SQL returns the status to SQLSTATE or SQLCODE after execution of each SQL statement.

SQL Engine

The core component of the Progress SQL-92 environment. The SQL engine receives requests from applications, processes them, and returns results.

SQL Result Set

In a stored procedure, the set of data rows generated by an SQL statement (SELECT and, in some cases, CALL).

Stored Procedure

A snippet of Java source code embedded in an SQL CREATE PROCEDURE statement. The source code can use all standard Java features as well as use Progress SQL-92-supplied Java classes for processing any number of SQL statements.

System Catalog

Tables created by the SQL engine that store information about tables, columns, and indexes that make up the database.

System Tables

Another term for **System Catalog**.

Transaction

A group of operations whose changes can be made permanent or undone only as a unit. Once implementations add the ability to change data in the proprietary storage system, they must also implement transaction management to protect against data corruption.

Trigger

A special type of stored procedure that helps ensure referential integrity for a database. Like stored procedures, triggers also contain Java source code (embedded in a CREATE

TRIGGER statement) and use Progress SQL-92 Java classes. However, triggers are automatically invoked (fired) by certain SQL operations (an insert, update, or delete operation) on the triggers' target table.

Trigger Action Time

The BEFORE or AFTER keywords in a CREATE TRIGGER statement. The trigger action time specifies whether the actions implemented by the trigger execute before or after the triggering INSERT, UPDATE, or DELETE statement.

Trigger Event

The statement that causes a trigger to execute. Trigger events can be SQL INSERT, UPDATE, or DELETE statements that affect the table for which a trigger is defined.

Triggered Action

The Java code within the BEGIN END clause of a CREATE TRIGGER statement. The code implements actions to be completed when a triggering statement specifies the target table.

Unicode

A superset of the ASCII character set that uses two bytes for each character rather than ASCII's seven-bit representation. Able to handle 65,536 character combinations instead of ASCII's 128, Unicode includes alphabets for many of the world's languages. The first 128 codes of Unicode are identical to ASCII, with a second-byte value of zero.

URL

In general, a Universal Resource Locator. Used to specify protocols and locations of items on the Internet. In JDBC, a database connection string in the form jdbc:*subprotocol:subname*.

Virtual Machine

The Java specification for a hardware-independent and portable language environment. Java language compilers generate code that can execute on a virtual machine. Implementations of the Java virtual machine for specific hardware and software platforms allow the same compiled code to execute without modification.

Index

A

API calls 1–2

C

class files Glossary–1

Class.forName 2–3

CLASSPATH 2–2

connecting to a database
  Class.forName 2–3
  URL 2–3

D

Data Types 3–2
  CallableStatement.getXXX 3–2
  Java 3–2
  JDBC 3–2
  PreparedStatement.setXXX 3–2

DatabaseMetaData 3–4

DatabaseMetadata
  methods 3–19

%DLC% 2–2

DLC/lib 2–3

driver manager 1–2

E

environment variables
  LD_LIBRARY_PATH 2–3
  PATH 2–2
  progress.zip 2–2

Error Messages 3–19

J

Java 3–2

Java compiler Glossary–1

JDBC 1–2, 3–1, 3–2, Glossary–2
  API calls 1–2
  architecture 1–2
  compared to ODBC 1–4
  conformance notes 3–1
  driver manager 1–2, 1–3
  JDBC driver Glossary–2
  JDBC driver manager Glossary–2
  overview 1–2

JDBC Driver 2–2
  configuring 2–2

JdbcProgress.dll 2–2, 2–3
libJdbcProgress_g.a 2–2, 2–3
libJdbcProgress_g.sl 2–2, 2–3
libJdbcProgress_g.so 2–2, 2–3

JdbcProgress.dll 2–2

JDK 1–2

L

LD_LIBRARY_PATH 2–3

libJdbcProgress_g.a 2–2, 2–3

libJdbcProgress_g.sl 2–2, 2–3

libJdbcProgress_g.so 2–2, 2–3

M

metadata Glossary–2

methods 3–19
DatabaseMetadata 3–19

N

native-method drivers 1–3

P

package Glossary–3

Password 2–5

progress.zip 2–2

ProgressTest.class 2–6

ProgressTest.java 3–19, A–2
Source Code A–1

R

Return Values 3–4
DatabaseMetaData 3–4

S

Source Code A–1

SQLCA Glossary–4

SQLCODE Glossary–4

SQLSTATE Glossary–5
error messages 3–19

stored procedure Glossary–5
input parameter Glossary–2
output parameter Glossary–3
procedure body Glossary–3
procedure specification Glossary–4
procedure variable Glossary–4

system catalog Glossary–5

system tables Glossary–5

T

transaction Glossary–5

trigger Glossary–5
trigger action time Glossary–6
trigger event Glossary–6
triggered action Glossary–6

U

Unicode Glossary–6

URL 2–6, Glossary–6
connecting to a database 2–6

User ID 2–5

V

virtual machine Glossary–6

virtual table result table Glossary–4