DBMaker

JDBC Programmer's Guide

CASEMaker®

CASEMaker Inc./Corporate Headquarters

1680 Civic Center Drive

Santa Clara, CA 95050, U.S.A.

www.casemaker.com

www.casemaker.com/support

Document No. 645049-235169/DBM53-M12302012-JDBC

Publication Date: 2012-12-30

For a description of updated functions that do not appear in this manual, read the file named README.TXT after installing the CASEMaker DBMaker software.

Trademarks

CASEMaker, the CASEMaker logo, and DBMaker are registered trademarks of CASEMaker Inc. Microsoft, MS-DOS, Windows, and Windows NT are registered trademarks of Microsoft Corp. UNIX is a registered trademark of The Open Group. ANSI is a registered trademark of American National Standards Institute, Inc.

Other product names mentioned herein may be trademarks of their respective holders and are mentioned only form information purposes. SQL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

Notices

The software described in this manual is covered by the license agreement supplied with the software.

Contact your dealer for warranty details. Your dealer makes no representations or warranties with respect to the merchantability or fitness of this computer product for any particular purpose. Your dealer is not responsible for any damage caused to this computer product by external forces including sudden shock, excess heat, cold, or humidity, nor for any loss or damage caused by incorrect voltage or incompatible hardware and/or software.

Information in this manual has been carefully checked for reliability; however, no responsibility is assumed for inaccuracies. This manual is subject to change without notice.

Contents

1 Introduction

Welcome to the JDBC User's Guide. This Guide will provide some instruction and samples for users to use the DBMaker JDBC. As all known, JDBC API Standard provides the solution to access database and to manipulate data from database in Java program. It was born with many advantages over ODBC and widely accepted by DBMS vendors and Java programmers; meanwhile, it becomes the standard of database access by Java. And now, DBMaker JDBC Driver supports most useful interfaces and their methods for JDBC2.0 and JDBC 3.0 standard.

1.1 Additional Resources

DBMaker provides complete sets of DBMS manuals in addition to this one. For more detailed information on a particular subject, consult one of the books listed below:

- For an introduction to DBMaker's capabilities and functions, refer to the DBMaker Tutorial.
- For more information on designing, administering, and maintaining a DBMaker database, refers to the Database Administrator's Guide.
- For more information on database server management, refer to the JServer Manager User's Guide.
- For more information on configuring DBMaker, refer to the JConfiguration Tool Reference.
- For more information on the native ODBC API and JDBC API, refer to the ODBC Programmer's Guide and JDBC Programmer's Guide.

- For more information on the dmSQL interface tool, refer to the dmSQL User's Guide.
- For more information on the SQL language used in dmSQL, refer to the SQL Command and Function Reference.
- For more information on the ESQL/C programming, refer to the ESQL/C User's Guide.
- For more information on error and warning messages, refer to the Error and Message Reference.
- For more information on the DBMaker COBOL Interface, refer to the DCI User's Guide.
- For more information on the Stored Procedure, refer to the Stored Procedure User's Guide.
- For more information on the Lock, refer to the Lock User's Guide.
- For more information on the DBMaker bundle version, refer to the DBMaker bundle Instruction.
- For more details on the performance of the database, refer to the Performance Tuning Guide.

1.2 Technical Support

CASEMaker provides thirty days of complimentary email and phone support during the evaluation period. When software is registered, the support period is extending an additional thirty days for a total of sixty days. However, CASEMaker will continue to provide email support (free of charges) for bugs reported after the complimentary support or registered support expires.

For most products, support is available beyond sixty days and may be purchased for twenty percent of the retail price of the product. Please contact sales@casemaker.com for details and prices.

CASEMaker support contact information, by post mail, phone, or email, for your area () is at: www.casemaker.com/support. We recommend searching the most current database of FAQ's before contacting CASEMaker support staff.

Please have the following information available when phoning support for a troubleshooting enquiry or include this information in your correspondence:

- Product's name and version number
- Registration number
- Registered customer's name and address
- Supplier/distributor where the product was purchased
- Platform and computer system configuration
- Specific action(s) performed before error(s) occurred
- Error message and number, if any
- Any additional information deemed pertinent

1.3 Document Conventions

A standard set of typographical conventions is used in the book for clarity and readable. The following table shows the conventions.

CONVENTION	DESCRIPTION
Italics	Italics is used to name or mark the graphic in the context.
Boldface	Boldface indicates the name of the interface or the class, and sometimes the name of the object of database. It will be appear in the context. Such as table name, stored procedure name and so on.
KEYWORDS	All keywords used by the SQL language appear in uppercase when used in normal paragraph text.

CONVENTION	DESCRIPTION
NOTE	Contains important information and it reminds user to pay attention for something.
Example	Examples are given to clarify descriptions, and commonly include text, as it will appear on the screen. Other forms of this convention include Prototype and Syntax.
CommandLine	This format is commonly used to show the JDBC code or the other code that the procedure needed

Table1-1DocumentConvention

2 Getting Started

This chapter will guide you to quickly start using the JDBC Driver. It illustrates how to install the DBMaker JDBC Driver, how to register and how to connect. There are some samples for user to understand easily.

2.1 Getting Started with JDBC Type II Driver

Get and Install DBMaker JDBC Driver

DBMaker JDBC Driver is distributed with the DBMaker product directly. In the directory of %DBMAKER_HOME\bin, dmjdbc20.jar, dmjdbc20xa.jar and dmjdbc30.jar are the binary files for DBMaker JDBC Driver. Interfaces introduced in JDBC API 2.0 are implemented in dmjdbc20.jar and dmjdbc20xa.jar, where dmjdbc20xa.jar contains the XA support interfaces besides the interfaces implemented in dmjdbc20.jar. In dmjdbc30.jar, these interfaces introduced in JDBC API 3.0 are implemented and at the same time, interfaces implemented in dmjdbc20.jar and dmjdbc20xa.jar included.

Register the DBMaker JDBC Driver

DBMaker JDBC Driver supports Type II JDBC driver, so some native binary files need to be used. For the Windows platform, the native file“dmjdbcXX.dll” is copied into the directory of %winnt\system32 during the process of installation of DBMaker, For the Unix/Linux/Solaris platform, user must specify the path

"/DBMAKER_HOME/lib/so" that contains the file libdmjdbcXX.so by the environment variable "LD_LIBRARY_PATH". User can set the LD_LIBRARY_PATH in his shell initial file like these examples below:

For sh (.profile), bash (.bashrc), add the following command in the initial file

```
export LD_LIBRARY_PATH=/DBMAKER_HOME/lib/so:$LD_LIBRARY_PATH
```

For csh/tcsh (.cshrc), add the following command in the initial file

```
setenv LD_LIBRARY_PATH /DBMAKER_HOME/lib/so
```

In addition, to compile and run the Java program using DBMaker JDBC Driver, you should set the CLASSPATH environment with the directory that contains .jar files for DBMaker JDBC Driver.

In Java program, if the connection is obtained by DriverManager, you should invoke the following command to register DBMaker JDBC Driver for your application before the connection is got by DriverManager.

```
 Class.forName("DBMaker.sql.JdbcOdbcDriver"). newInstance();
```

Connect to Database by DBMaker JDBC Driver

DBMaker JDBC Driver supports to get the connection of database by both DriverManager and DataSource. Firstly, we introduce the usage of DriverManager here.

The simplest case is:

```
Connection conn =
DriverManager.getConnection("jdbc:DBMaker:dbname","SYSADM","abc123");
```

As has indicated in the code, we got the connection to the database 'dbname' as the identity "SYSADM" with the password "abc123".

And also, we can get the connection by Properties showed below:

```
Properties connect property = new Properties();

connect property.setProperty("user","SYSADM");

connect property..setProperty("password", "abc123");

Connection conn = DriverManager.getConnection("jdbc:DBMaker:dbname",
connect_property);
```

If you want to connect to the database server by the URL that specify by server address and port number，like "jdbc:DBMaker://SERVER_ADDRESS:PORT_NUMBER/DATABASE_NAME"，You can do like below:

```
Connection conn =
DriverManager.getConnection("jdbc:DBMaker://127.0.0.1 :2345/dbsmaple4","
SYSADM","abc123");
```

This method for connection can avoid the necessarily of specifying the database section in dmconfig.ini.

On the other hand, DBMaker JDBC implements the interface DataSource and XADataSource to get the connection. Please see the Using DBMaker with J2EE Application Server Manual for the details.

2.2 Getting Started with JDBC Type III Driver

Get and Install DBMaker JDBC Driver

Type III JDBC driver which is a pure java JDBC driver makes use of a middle tier between the calling program and the database. The middle-tier (application server) converts JDBCcalls directly or indirectly into the vendor-specific database protocol. To use Type II JDBC, users must at least have the DBMaker client installed. To use Type III JDBC, users can connect to the database's application code with the Type III JDBC jar file dmjdbct3c.jar. Type III JDBC is designed for users who don't want to install the DBMaker client.

Register the DBMaker JDBC Driver

DBMaker JDBC Driver supports Type III JDBC driver. To make connection to DBMaker server with Type III JDBC driver, users only need a Type III JDBC jar file.

In Java program, if the connection is obtained by DriverManager, you should invoke the following command to register DBMaker JDBC Driver for your application before the connection is got by DriverManager.

```
Class.forName("dbmaker.jdbc.ws.client.Driver"). newInstance();
```

Connect to Database by DBMaker JDBC Driver

Type III server that is a standalone middle tier server is not bound with any specific database. Its life cycle is different from database server. Type III server comes with DBMaker server installation, but database server and Type III server don't need to be on the same site. Users should start up both database server and Type III server. The database setting must be properly written in dmconfig.ini where Type III server can read. Type III server can not be used to set session timeout, but users can use datebase server to handle session timeout. Only databases with same lcode can be accessed via Type III server at the same time. For example, db1 (lcode=1) and db2 (lcode=2). If Type III server has already connected to db1, then an error will be returned if users try to make connect Type III server with db2. The workaround would be starting 2 jetty server in different ports to serve different lcode databases.

Type III server is implemented with hessian and deployed under jetty 7 server. Type III client and server both require jre 1.6 environment.

To start Type III server on Windows, users need to click **DBMakter 5.3** and then click **JDBCT3Server Start** in the extended meun. To start Type III server on Linux/Unix, users can use the following command:

```
type ~dbmaker/5.3/bin/t3svr
```

The default port number for Type III server is 8083, but users can change it.

On Windows, open the **shortcut** tab in **JDBCT3Server Start Properties** page and you will see the **target** field. The following is the value in the **target** filed:

```
C:\DBMaker\5.3\jre\bin\java.exe -Djava.library.path="C:\DBMaker\5.3\bin"
-Djetty.port=8083 -DSTOP.PORT=8082 -DSTOP.KEY=stopme -jar start.jar
```

Lastly, users can simply change 8083 to another number with editing "Djetty.port=8083".

On Linux/Unix, edit the following **t3svr** script:

```
#!/bin/sh
#
PROG=`basename $0`
```

```
case ${PROG} in
  t3svr stop) ARGS="--stop" ;;
  *) ARGS="--daemon" ;;
esac

cd /home/dbmaker/5.3/jetty
exec /home/dbmaker/5.3/jre/bin/java \
          -Djava.library.path=/home/dbmaker/5.3/lib/so \
          -Djetty.port=8083 \
          -DSTOP.PORT=8082 \
          -DSTOP.KEY=stopme \
          -jar /home/dbmaker/5.3/jetty/start.jar ${ARGS}
```

Note *The dll/so library path (dmjdbc53.dll/libdmjdbc53.so) is already set in the shortcut/script (-Djava.library.path=xxxx).There is no need to add dbmaker/5.3/bin to library path.*

Type III server provides corresponding servlets, which are wrapped classes for dmjdbc (Type II) and follow HTTP protocol.The classes and methods supported by Type III JDBC jar file dmjdbct3c.jar provides JDBC interface. The supported classes/methods are similar to dmjdbc (type II). The major difference is that users must specify the host name and port number to connect to Type III middle server.

To make connection with dmjdbc (Type II):

- Before coding, users need to have dmjdbc30.jar in classpath. There are two ways to specify the java classpath. One is to set the environment variable CLASSPATH. The other is to set the java command argument –classpath. Type II JDBC needs libdmjdbc53.so/dmjdbc53.dll in PATH or LD_LIBRARY_PATH
- In the coding, load class DBMaker.sql.JdbcOdbcDriver
- The url is like jdbc:DBMaker:<dbname>

Code example:

```
import java.sql.*;
```

```
public class SimpleTest
  {
   public static void main(String[] args)
     {
      try
        {
         Class.forName("dbmaker.sql.JdbcOdbcDriver");
         Connection conn =
DriverManager.getConnection("jdbc:dbmaker:JDBCTEST", "SYSADM",
         "abc");
         DatabaseMetaData dbmd= conn.getMetaData();
         ResultSet rs = dbmd.getTables(null, null, null, null);
         while (rs.next())
           {
            System.out.print(rs.getString(1)+" ");
            System.out.print(rs.getString(2)+" ");
            System.out.println(rs.getString(3)+" ");
           }
         rs.close();
         conn.close();
        }
      catch (Exception e)
        {
         e.printStackTrace();
        }
     }
  }
```

To make connection with Type III JDBC:

- Before coding, users need to have the type 3 jdbc jar file **dmjdbct3c.jar** in classpath, but jdbc so/dll in PATH or LD_LIBRAY_PATH is not needed.
- In the coding, load class DBMaker.jdbc.ws.client.Driver
- The url is like jdbc:DBMaker:type3://<type3server ip>:<port number>/<dbname>

Code example:

```
import java.sql.*;
public class SimpleTest
  {
   public static void main(String[] args)
     {
      try
        {
         Class.forName("dbmaker.jdbc.ws.client.Driver");
         Connection conn =
DriverManager.getConnection("jdbc:dbmaker:type3://127.0.0.1:8083/JDBCTES
T", "SYSADM", abc");
         DatabaseMetaData dbmd= conn.getMetaData();
         ResultSet rs = dbmd.getTables(null, null, null, null);
         while (rs.next())
           {
            System.out.print(rs.getString(1)+" ");
            System.out.print(rs.getString(2)+" ");
            System.out.println(rs.getString(3)+" ");
           }
         rs.close();
         conn.close();
        }
     catch (Exception e)
       {
        e.printStackTrace();
       }
    }
  }
```

3 Sample Programs

This chapter gives some detailed samples for using JDBC. And give you these steps for running them in the DBMaker.

3.1 Run the Sample Programs in the DBMaker

With Type II Driver

In the following description, we use DBMAKER_HOME to indicate the directory where DBMaker is installed. For example, DBMAKER_HOME maybe indicate the directory like '/home/DBMaker/5.3' for DBMaker 5.3 on Unix platform or for the directory like 'C:\DBMaker\5.3' for DBMaker 5.3 on Windows platform.

With the distribution of DBMaker, there are three JDBC sample programs named as ex_Resultset.java, ex_ExecuteParam.java and ex_Resultset_update.java separately in the directory %DBMAKER_HOME%\samples\JDBC. Those files demonstrate the common usage of DBMaker JDBC Driver. In the ex_Resultset.java, the usage of inserting data and retrieving data is shown. In the ex_ExecuteParams.java, the usage of updating/inserting data with parameters is introduced. And finally, we demonstrate the usage of updating/inserting data with ResultSet and the newly API that introduced by JDBC Standard since JDBC2.0 in ex_Resultset_update.java.

To compile and run the sample programs of JDBC, firstly, DBSAMPLE5, the database that was created at the process of installation of DBMaker, should be started.

And also, the JDK1.2 or later should be installed in your OS and make sure that the JRE works well for your OS.

Please refer to the section of Register the DBMaker JDBC Driver of Chapter 2.1 to register the driver properly. And then you can compile and run the sample programs by command of java and javac separately.

If you register the dmjdbcXX.jar into the environment variable of CLASSPATH of your OS, the sample programs, ex_ExecuteParam.java, can be compiled and run by the following command for both UNIX and Windows:

```
javac ex ExecuteParam.java
java ex_ExecuteParam
```

If the dmjdbcXX.jar is not registered into the CLASSPATH, the sample programs, ex_ExecuteParam.java, can be compiled and run by specifying the CLASSPATH with the arguments -classpath of javac and java.

The following is command for UNIX:

```
javac -classpath /DBMAKER HOME/lib/java/dmjdbcxx.jar:./
ex ExecuteParam.java
java -classpath /DBMAKER_HOME/lib/java/dmjdbcxx.jar:./ ex_ExecuteParam
```

The following is command for Windows:

```
javac -classpath \DBMAKER HOME\bin\dmjdbcxx.jar;.\ ex ExecuteParam.java
java -classpath \DBMAKER_HOME\bin\dmjdbcxx.jar;.\ ex_ExecuteParam
```

With Type III Driver

With the distribution of DBMaker, there is Employee.java in the directory of %DBMAKER_HOME%\samples\JDBC . This sample demonstrates how to use type 3 jdbc to retrieve data from database.

To compile and run the sample programs of JDBC, firstly, DBSAMPLE5, the database that was created at the process of installation of DBMaker, should be started. And also, the JDK1.6 or later should be installed in your OS and make sure that the JRE1.6 works well for your OS.

Please refer to the section of Register the DBMaker JDBC Driver of Chapter 2.2 to register the driver properly. And then you can compile and run the sample programs by command of java and javac separately.

If you register the dmjdbct3c.jar into the environment variable of CLASSPATH of your OS, the sample programs, say Employee.java, can be compiled and run by the following syntax for both UNIX and Windows:

```
javac Employee.java
java Employee
```

If the dmjdbct3c.jar is not registered into the CLASSPATH, the sample programs, say Employee.java, can be compiled and run by specifying the CLASSPATH with the arguments -classpath of javac and java, namely,

The following is command for UNIX:

```
javac -classpath /DBMAKER HOME/lib/java/dmjdbct3c.jar:./ Employee.java
java -classpath /DBMAKER_HOME/lib/java/dmjdbct3c.jar:./ Employee
```

The following is command for Windows:

```
javac -classpath \DBMAKER HOME\bin\dmjdbct3c.jar;.\ Employee.java
java -classpath \DBMAKER_HOME\bin\dmjdbct3c.jar;.\ Employee
```

3.2 The example for general JDBC programs

How to execute a SQL command by JDBC?

Here we introduce the general steps using Statement and PreparedStatement to manipulate the database object, like CREATE TABLE, INSERT TUPLES and so on.

The conn object is connected to the database server. For connection to database server by DBMaker JDBC, please refer to the section 2.

To create one Statement for one connection:

```
Statement stmt = conn.createStatement();
```

To execute DDL by the Statement:

```
stmt.executeUpdate("create table jdbc employee (id int, name char(20),
 salay float, hired_date date)");
```

To execute DML by the Statement:

```
stmt.executeUpdate("insert into jdbc employee values (1, 'Charles
Brown', 555.32, '1999/01/01')");
```

To prepare the **PreparedStatement** with host variables:

```
PreparedStatement pstmt = conn.prepareStatement("insert into
jdbc_employee values(?,?,?,?)");
```

Please note that the call of method of PreparedStatement is a time consuming job, so please reuse the PreparedStatement for the same DML with the same host variable as possible as you can.

If we want to insert the information for an employee: the employee id is 4, employee name is “"Mickey Mouse", his salary is “30000.00” and hired date is “1950-01-01”, these data can be inserted by one **PreparedStatement** with host variables:

```
....
pstmt.setInt(1, 4);
    pstmt.setString(2, "Mickey Mouse");
     pstmt.setFloat((3, 30000.00) ;
 pstmt.setDate(4, new Date(50, 0, 1)) ;
pstmt.executeUpdate();
.....
```

If we want to insert some information for three employees “John”, “Mary” and “Eric” with the same **PreparedStatement** prepared before, the code could be like this:

```
// 1. insert the information for employee  "John"
 pstmt . setInt (1, 3045) ;
pstmt.setString(2, "John" ) ;
pstmt.setFloat( 3, 10000.00) ;
pstmt.setDate( 4, new  Date ( 28, 1,3)) ;
// To execute the insert commnad without prepare time because the
prepare work had been finished while calling "conn.prepareStatement "
pstmt.executeUpdate() ;
// 2. insert the information for employee  "Mary"
 pstmt . setInt (1, 3200) ;
```

```
pstmt.setString(2, "Mary" ) ;
pstmt.setFloat( 3, 15000.00) ;
pstmt.setDate( 4, new  Date ( 28, 1,3)) ;
// To execute the insert commnad without prepare time because the
prepare work had been finished while calling "conn.prepareStatement "
pstmt.executeUpdate() ;
// 3. insert the information for employee  "Eric"
 pstmt . setInt (1, 3011) ;
pstmt.setString(2, "Eric" ) ;
pstmt.setFloat( 3, 40000.00) ;
pstmt.setDate( 4, new  Date ( 28 1,3)) ;
// To execute the insert commnad without prepare time because the
prepare work had been finished while calling "conn.prepareStatement "
pstmt.executeUpdate() ;
}
```

So, we insert three tuples by one time “prepare” and three times “execution”.

How to retrieve data from database by JDBC?

For the convenience of demo, we suppose the table schema like this:

```
create table jdbc employee (id int, name char(20), salary float,
hired_date date) ;
```

Firstly, a Statement object is created by the connection connected to the DBMaker database server:

```
Statement stmt = conn.createStatement();
```

And then, you can execute the query command by the method executeQuery of the object of Statement and get the result set:

```
ResultSet rs = stmt.executeQuery(“select id,name,salary,hired_date from
jdbc_employee");
```

Next, you can iterate the result set obtained by the query of Statement:

```
While(rs.next())
{
System.out.println(“ID: “+rs.getInt(1) +
   “NAME: “+rs.getString(2) +
        “SALARY: “+ rs.getFloat(3) +
```

```
                "HIRED DATE: " + rs.getDate(4) );
}
```

To use cursor update in the object of **ResultSet**, you should, firstly, create the object of Statement with the **ResultSet** Type ResultSet.CONCUR_UPDATABLE:

```
Statement stmt = con.createStatement (ResultSet.TYPE SCROLL SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

For the detailed information of **ResultSet** type, please refer to the JDBC Specification.

And now, you can use the same method as before to execute query command and obtain the **ResultSet**:

```
ResultSet rs = stmt.executeQuery("select * from jdbc_employee");
```

However, this time, the **ResultSet** is updateable with cursor:

```
While (rs.next())
{
     float salary = rs.getFloat(2);
     salary = salary + 1000.00f;
     rs.updateFloat(3, salary);
     rs.updateRow();
}
```

To make more efficiency memory allocation when retrieving data and avoid some duplicate checking, for example checking if need trace or need column remapping. There are two major changes in dmjdbc performance enhancement.

- In earlier version of dmjdbc, in JdbcOdbcResultSet getXXX would allocate buffer every time in java, pass it to c, the c code would assign retrieved value into the buffer. The allocated buffer was not reused. The enhancement is trying to reuse the allocated buffer by avoiding frequently allocate/free memory.
- The debugging mechanism of JDBC is to call DriverManager.setLogStream or DirverManager.setLogWriter. In earlier version of JdbcOdbcXXX classes, the first line of those JDBC implemented methods would check for log stream or log writer and determine if need to print debug message. This checking operation had quite significant performance impact. So, the improvement is to keep a flag (a variable) needTrace in JdbcOdbc.java, the ancestor of all JdbcOdbcXXX

classes. So all JdbcOdbcXXX would inherit this variable and have it assigned when the object is created. In each method, it would check for the flag needTrace only, and not call DriverManager.getLogStream or getLogWriter.

Please note that there is one issue about needTrace in dmjdbc performance enhancement. This value is set when the object is created and users can't alter it after then. So, if users do something like:

```
Statement stmt = conn.createStatement();
DriverManager.setLogWriter(new java.io.PrintWriter("c:\temp\jdbc.log"));
ResultSet rs = stmt.executeQuery("xxx");
```

Then, only trace messages from rs will be printed to log but not the ones from stmt. Because the log writer is set after stmt has been created.

How to handle blob data by JDBC?

DBMaker provides powerful functions for the manipulation of large object, including data type of Blob, Clob and FO. DBMaker JDBC Driver implements the interface **java.sql.Blob** and **java.sql.Clob** for Blob and Clob data type in Java program.

We will introduce the usage of Blob and Clob data type in Java program by DBMaker JDBC Driver.

Please see Chapter 4.2 for the APIs implemented by DBMaker JDBC Driver in details. Here we give some samples of JDBC codes for Blob and Clob data and the FO.

USAGE OF CLOB SAMPLE

This sample demonstrates the usage of Clob data with DBMaker JDBC Driver.

In this sample we store the content of the text file demo.txt into database and then retrieve it out and print it to the console.

Firstly, we create the table **jdbc_clob_demo** with a column with long varchar type to store the Clob data.

```
stmt.execute ("CREATE TABLE jdbc clob demo (id INT, content LONG
VARCHAR)");
```

And then, we store the content of the text file 'demo.txt' into this table like this:

```
PreparedStatement pstmt= conn.prepareStatement( "INSERT INTO
jdbc clob demo(id,content) VALUES(?,?)");
FileInputStream fis= new FileInputStream ("demo.txt");
Buff len = fis.available();
pstmt.setInt(1,1);
pstmt.setBinaryStream(2,fis,buff len);
pstmt.execute();
pstmt.close();
fis.close();
```

Now, we have completed the job to store the content of text file 'demo.txt' into database. Next, we will retrieve it out and print the content to the console. In practice, you maybe do something more useful, not just print it out. Following is the code for retrieving and printing:

```
ResultSet rs = stmt.executeQuery("SELECT ID,content FROM
jdbc clob demo");
If (rs.next())
{
Int id =rs.getInt(1);
Clob clob = rs.getClob(2);
/*Get the bytes as ASCII stream */
InputStream astream = clob. getAsciiStream();
    Int len = astream.available();
byte[] bytes = new byte [len+1];
astream.read(bytes);
astream.close();
/* we construct the String instance from bytes with ASCII charset */
String str = new String(bytes, 0, len, "ascii");
System.out.printIn(str);
}
```

For the complete code of this sample, please refer to Appendix for Sample 1.

At the same time, the Clob retrieved by **ResultSet** can also be used to insert some new rows like this:

```
If (rs.next())
{
int id = rs.getInt(1);
Clob content = rs.getClob(2);
        ......
/* Here we insert a new tuple with the ID=2 and the same Clob content as
the tuple with ID = 1 */
 pstmt.setInt(1,2);
     pstmt.setClob(2,content);
     pstmt.execute();
}
```

USAGE OF BLOB SAMPLE

This sample illustrates the usage of Blob data with DBMaker JDBC Driver.

In this sample, we store the picture demo.gif into database and then retrieve it out and save it as a gif file.

Firstly, we create the table **jdbc_blob_demo** with a column with the type of long varbinary to store the picture file:

```
stmt.execute("CREATE TABLE jdbc_blob_demo(id INT, photo LONG VARCHAR)");
```

And then, we read the picture file demo.gif and restore it into **jdbc_blob_demo**:

```
PreparedStatement pstmt = c.prepareStatemnt( "INSERT INTO
jdbc blob demo(id,photo) VALUES(?,?)");
FileInputStream fis = new FileInputStream("demo.qif");
Int buff len = fis.available();
pstmt.setInt(1,1);
pstmt.setBinaryStream(2,fis,buff len);
pstmt.execute();
pstmt.close();
fis.close();
```

Now the picture 'demo.gif' is stored into database as a binary data. Next, we will retrieve the binary data out from **jdbc_blob_demo** and save it as a gif file named 'dup-demo.gif'. For your application, you maybe show the picture on some control. The following is the code for this job:

```
ResultSet rs = stmt.executeQuery ("SELECT ID, PHOTO FROM
jdbc blob test");
If (rs.next())
{
        int id = rs.getInt(1);
        Blob blob data = rs.getBlob(2);
        byte[ ] buffer = new byte[buff len + 1];z
        InputStream bs = blob data.getBinaryStream();
        FileOutputStream fos = new FileOutputStream("dup-demo.gif");
        bs.read(buffer);
        fos.write (buffer);
        bs.close ();
        fos.close ();
}
```

For the complete code of this sample, please refer to Appendix for Sample 2.

Meanwhile, the Blob retrieved by ResultSet can also be used to insert some new rows like this:

```
If (rs.next())
 {
int id = rs.getInt(1);
 Blob photo = rs.getBlob(2);
 /* Here we insert a new tuple with ID = 2 but the same photo as the
tuple with ID = 1*/
     pstmt.setInt(1,2);
     pstmt.setBlob(2,photo);
     pstmt.execute();
 }
```

USAGE OF FILE OBJECT SAMPLE

DBMaker also supports the FILE data type which is the useful feature of DBMaker. The FILE type column can store the binary data as a file outside the database blob file (*.bb). The FILE data type can be used just the same as LONG VARBINARY or LONG VARCHAR for JDBC programmer if the keyword DB_FoTyp is set to "1" in dmconfig.ini, please refer to the DBA Manual (dba.chm) for the keyword in detail. In

Appendix, the Sample 3 provides a complete demo for the usage of FILE data type in JDBC.

How to call a stored RPOCEDURE?

We can write a Stored Procedure with the grammar of EC and create the Stored Procedure in dmSQL from the EC program file, please see the Stored Procedures section in dba.chm manual for details.

The Stored Procedure can be called in JDBC with the syntax:

```
{CALL procedure_name[(?,?)]} or {?=CALL procedure_name[(?,?)]}.
```

The following sample demonstrates the usage of Stored Procedure with DBMaker JDBC Driver.

The table schema used in the sample is shown as the following:

```
Create table SYSADM.JDBC SP DEMO(
        ID INTEGER not null,
        NAME VARCHAR(12) default null,
        BIRTHDAY DATE default null )
        in DEFTABLESPACE lock mode page fillfactor 100;
ALTER TABLE SYSADM.JDBC_SP_DEMO primary key (ID) in DEFTABLESPACE;
```

The stored procedure **UPDATE_OR_INSERT** here is used to insert the record into the table **JDBC_SP_DEMO** if the record does not exist in the table, and to update the record with the same ID in the table **JDBC_SP_DEMO** if the record has already existed in the table. Please see the file update_or_insert.ec in the Sample 4 in Appendix.

Before we call the procedure by JDBC, we have to create this procedure into database first. DBMaker provides the command 'CREATE PROCEDURE FROM …' to create the stored procedure into database from the file that contains the EC program for the stored procedure. Please see the Stored Procedures section in DBA manual dba.chm for details.

Next, we will introduce the general steps of calling Stored Procedure. For the convenience of illustrations, we just suppose that the procedure we are going to call is

named by 'demo_sp', with two parameters. The first one, with the type of String, is used as the INPUT and the second one, with the type of Integer, is used as OUTPUT parameter. And also, the store procedure 'demo_sp' will return the ResultSet of query, containing two columns with the type of Integer and String, separately.

1. To prepare the CallableStatement by java.sql.Connection.prepareCall().

The stored procedure should be prepared by the JDBC escape syntax. The following code is the sample:

```
// prepare to call the stored procedure demo sp with two parameters.
// variable conn is an available connection to the destination database.
CallableStatement cstmt = conn. prepareCall("{CALL demo_sp(?,?)}");
```

2. To register the output variable by **CallableStatement.registerOutputParameter**(), If any.

DBMaker JDBC Driver supports output parameter for Stored Procedure. If there is any output parameter, it should be registered like this:

```
// Register the output parameter, i.e.,  the second parameter named by
'age' with
// integer datatype. Variable cstmt is prepared by calling
Connection.prepareCall().
cstmt.registerOutputParameter(2,Types,INTEGER);
```

Please note thar both the index (1-base started) and name for column are supported to register the output parameter.

3. To set the input parameter by CallableStatement.setXXX().

Like the PreparedStatement, both the index (1-base started) and name for column are supported to set the input parameter. The input parameter can be set like this:

```
// Set the input parameter, say, the first parameter named by 'id' with
varchar(10)
// datatype.  Variable cstmt is prepared by calling
Connection.prepareCall().
cstmt.setString(1,'21935265');
```

4. Execute and retrieve the query result by cstmt.executeQuery()

If the Stored Procedure do something for query and returns the result set, the result set can be accessed by the instance of class java.sql.ResultSet returned by cstmt.executeQuery() .

```
//  Iterate the result set returned by CallableStatement.executeQuery().
//  Variable cstmt is prepared by calling Connection.prepareCall().
     int id ;
     String name ;
boolean brs = cstmt.execute() ;
// brs is true means the cstmt.execute() will product a Resultset
If (brs == true )
 {
     ResultSet rs = cstmt.getResultSet() ;
While(rs.next())
{
      id = rs. GetInt(1) ;
      name = rs.getString(2) ;
}
```

Please note that both the index-based and name-based columns are supported to retrieve the output parameter.

How to get metadata for Database/ResultSet/Parameter?

The metadata of JDBC Specification plays an important role in the advanced programming.

DatabaseMetaData interface contains all the method relating to the specialization of the database server, such as whether the transaction, stored procedure, outer join, etc, are supported and some other information about the target database server.

The **DatabaseMetaData** can be obtained by the following code:

```
DatabaseMetaData dbmd = conn.getMetaData();
```

the **conn** is a Connection object connected to the DBMaker database server.

For example, if you do not know whether the DBMS support transaction, we can get the metadata through the method supportsTransactions() of interface **DatabaseMetaData**.

```
  If (dbmd.supportsTransactions())
   {
       conn.setAutoCommit(false);
    }
 else {
     System.out.println(" transaction is not supported ");
        }
```

Likewise, if you want to know whether the DBMS support Stored Procedure, we can get the metadata through the method supportsStoredProcedure() of interface **DatabaseMetaData**.

```
If (dbmd.supportsStoredProcedure() )
       {
CallableStatement cstmt = conn.preprareCall( "{ call demo sp(? , ?) }" )
        }
else {
     System.out.println(" Stored Procedure is not supported ");
     }
```

ResultSetMetaData interface contains all the method relating to the information of the **ResultSet** returned by a query, such as the number of column, the name and type of each column and so on. Programmer can use that information to achieve some generic programming.

You can obtain the **ResultSetMetaData** by the following code:

```
ResultSetMetaData rsmd = rs.getMetaData();
```

the rs is a **ResultSet** object returned by some query statement. For example, if you know nothing about the **ResultSet** retrieved by some Query, you can iterate the **ResultSet** with the help of **ResultSetMetaData** like this:

```
/* Get the metadata of the ResultSet 'rs' */
ResultSetMetaData rsmd = rs.getMetaData();
/* Get the count of columns of this ResultSet by ResultSetMetaData*/
```

```
int colCount = rsmd.getColumnCount();
/* Print the name of each column by ResultSetMetaData */
for(int i = 1;i <= colCount;++i) {
    System.out.print(rsmd.getColumnName(i)+"\t");
}
/* Iterate the result set */
while(rs.next()) {
for(int i = 1;i < colCount;++i) {
/* Get the type of each column*/
int type = rsmd.getColumnType(i);
/* Check the type to use the corresponding getXXX method*/
     switch(type) {
       case Types.INTEGER :
            System.out.print(rs.getInt(i));
case Types.CHAR :
        case Types.VARCHAR :
              System.out.print(rs.getString(i));
        case …..:   /* And many other cases omitted here */
          }
}
 System.out.println();
}
```

ParameterMetaData interface contains all the method relating to the information of the parameters in the statement with host variables, such as the JDBC Type and precision of those variables.

The **ParameterMetaData** can be obtained by the following code:

```
ParameterMetaData pmd = pstmt.getMetaData();
```

the **pstmt** is a **PreparedStatement** object prepared by some connection object.

We can also get the metadata for parameters of the stored procedure executed by **CallableStatement**.

Here, as a sample, we list the code section of using **ParameterMetaData** to retrieve the metadata information of parameters in the **CallableStatement**.

```
ParameterMetaData pmd = cstmt.getParameterMetaData();

int paramsCount = pmd.getParameterCount();

int type=0;

int mode=0;

for (int i = 1;i <= paramsCount; i++ )

{

/* Get the JDBC type of the Parameter,defined in java.sql.Types */

type = pmd.getParameterType(i);

/* Get the mode of the Parameter,i.e,INPUT, OUTPUT or INPUTOUTPUT */

/* All of the avaliable MODE defined as constant  in ParameterMetaData
*/

imode = pmd.getParameterMode(i);

   /* We can use the type and mode of the parameters to program the
general code , but  we just  print the type and mode of the parameters
here. */

System.out.println( "Parameter "+i+"Type = "+type+", Mode = "+mode);

}
```

4 Reference of DBMaker JDBC Driver

This chapter presents some reference of DBMaker JDBC Driver, including the data types supported by DBMaker JDBC Driver, the APIs that implemented by DBMaker JDBC Driver and the implemented system function.

4.1 Data Types supported

The following table 5-1 shows the data types that DBMaker JDBC Driver supported.

SQL Type	JDBC Type
CHAR	Types.CHAR
INTEGER	Types.INTEGER
SMALLINT	Types.SMALLINT
FLOAT	Types.REAL
DOUBLE	Types.DOUBLE
VARCHAR	Types.VARCHAR
LONG VARCHAR	Types.LONGVARCHAR
BINARY	Types.BINARY
LONG VARBINARY	Types.LONGVARBINARY
DATE	Types.DATE

TIME	Types.TIME
DECIMAL	Types.DECIMAL
TIMESTAMP	Types.TIMESTAMP
DOUBLE PRECISION	Types.FLOAT
NCHAR	Types.CHAR
NVARCHAR	Types.VARCHAR
NCLOB	Types.LONGVARCHAR
CLOB	Types.LONGVARCHAR
BLOB	Types.LONGVARBINARY

Table 4-1 Supported Datatypes

4.2 JDBC API Implemented

JDBC Type II Driver

At present, DBMaker JDBC Type II Driver supports most useful interfaces and their methods for JDBC 2.0 and JDBC 3.0 standard. These interface and methods are described below.

JAVA.SQL.BLOB

Methods	Version
getBytes(long pos,int length) throws java.sql.SQLException;	2.0
getBinaryStream() throws java.sql.SQLException;	2.0
Length() throws java.sql.SQLException;	2.0

Table 4-2 java.sql.Blob

JAVA.SQL.CALLABLESTATEMENT

Methods	Version
getObject(int parameterIndex) throws java.sql.SQLException;	1.0
getBoolean(java.lang.int parameterIndex) throws java.sql.SQLException;	1.0
getByte(int parameterIndex) throws java.sql.SQLException;	1.0
getShort(int parameterIndex) throws java.sql.SQLException;	1.0
getInt(int parameterIndex) throws java.sql.SQLException;	1.0
getFloat(int parameterIndex) throws java.sql.SQLException;	3.0
getDouble(int parameterIndex) throws java.sql.SQLException;	1.0
getBytes(int parameterIndex) throws java.sql.SQLException;	1.0
getDate(int parameterIndex) throws java.sql.SQLException;	1.0
getDate(int parameterIndex,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
getString(int parameterIndex) throws java.sql.SQLException;	1.0
getTime(int parameterIndex,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
getTime(int parameterIndex) throws java.sql.SQLException;	1.0
getTimestamp(int parameterIndex,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
getTimestamp(int parameterIndex) throws java.sql.SQLException;	1.0
setNull(java.lang.String parameterName,int jdbcType,,java.lang.String typeName) throws java.sql.SQLException;	3.0
registerOutParameter(int parameterIndex,int jdbcType,int sacle) throws java.sql.SQLException;	1.0

registerOutParameter(int parameterIndex,int jdbcType) throws java.sql.SQLException;	1.0
setByte(int parameterIndex,byte x) throws java.sql.SQLException;	1.0
setDouble(int parameterIndex,double x) throws java.sql.SQLException;	1.0
setFloat(int parameterIndex,float x) throws java.sql.SQLException;	1.0
setInt(int parameterIndex,int x) throws java.sql.SQLException;	1.0
setShort(int parameterIndex,short x) throws java.sql.SQLException;	1.0
setTimestamp(int parameterIndex,,java.sql.Timestamp x,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
setTimestamp(int parameterIndex,,java.sql.Timestamp x) throws java.sql.SQLException;	1.0
execute() throws java.sql.SQLException;	1.0
getMetaData() throws java.sql.SQLException;	2.0
clearParameters() throws java.sql.SQLException;	1.0
setAsciiStream(int parameterIndex,,java.io.InputStream fin,int length) throws java.sql.SQLException;	1.0
setBinaryStream(int parameterIndex,,java.io.InputStream fin,int length) throws java.sql.SQLException;	1.0
setBlob(int parameterIndex,,java.sql.Blob x) throws java.sql.SQLException;	2.0
setBytes(int parameterIndex,Byte x) throws java.sql.SQLException;	1.0

setCharacterStream(int parameterIndex,,java.io.Reader reader,int length) throws java.sql.SQLException;	2.0
setClob(int parameterIndex,,java.sql.Clob x) throws java.sql.SQLException;	2.0
setDate(int parameterIndex,,java.sql.Date x,java.util.Calendar cal) throws java.sql.SQLException;	2.0
setDate(int parameterIndex,,java.sql.Date x) throws java.sql.SQLException;	1.0
setNull(int parameterIndex,int jdbcType) throws java.sql.SQLException;	1.0
setNull(int parameterIndex ,int jdbcType,,java.lang.String typeName) throws java.sql.SQLException;	1.0
setObject(int parameterIndex,,java.lang.Object x) throws java.sql.SQLException;	1.0
setObject(int parameterIndex,,java.lang.Object x,int targetJdbcType,,int scale) throws java.sql.SQLException;	1.0
setObject(int parameterIndex,,java.lang.Object x,int targetJdbcType) throws java.sql.SQLException;	1.0
setString(int parameterIndex,,java.lang.String x) throws java.sql.SQLException;	1.0
setTime(int parameterIndex,,java.sql.Time x) throws java.sql.SQLException;	1.0
executeQuery() throws java.sql.SQLException;	1.0
executeUpdate() throws java.sql.SQLException;	1.0
getParameterMetaData() throws java.sql.SQLException;	3.0
close() throws java.sql.SQLException;	1.0

execute(java.lang.String sql) throws java.sql.SQLException;	3.0
getConnection() throws java.sql.SQLException;	2.0
clearWarnings() throws java.sql.SQLException;	1.0
getWarnings() throws java.sql.SQLException;	1.0
getFetchDirection() throws java.sql.SQLException;	2.0
getFetchSize() throws java.sql.SQLException;	2.0
setFetchDirection(int direction) throws java.sql.SQLException;	2.0
setFetchSize(int rows) throws java.sql.SQLException;	2.0
getMaxRows() throws java.sql.SQLException;	1.0
setMaxRows(int max) throws java.sql.SQLException;	1.0
getResultSet() throws java.sql.SQLException;	1.0
cancel() throws java.sql.SQLException;	1.0
executeQuery(java.lang.String sql) throws java.sql.SQLException;	1.0
executeUpdate(java.lang.String sql) throws java.sql.SQLException;	1.0
getResultSetConcurrency() throws java.sql.SQLException;	2.0
getResultSetHoldability() throws java.sql.SQLException;	3.0
getResultSetType() throws java.sql.SQLException;	2.0
getUpdateCount() throws java.sql.SQLException;	1.0
setCursorName(java.lang.String name) throws java.sql.SQLException;	1.0

Table 4-3 java.sql.CallableStatement

JAVA.SQL.CLOB

Methods	Version
length() throws java.sql.SQLException;	2.0
getAsciiStream() throws java.sql.SQLException;	2.0
getSubString(long pos,int length) throws java.sql.SQLException;	3.0

Table 4-4 java.sql.Clob

JAVA.SQL.CONNECTION

Methods	Version
setReadOnly(boolean readOnly) throws java.sql.SQLException;	1.0
isReadOnly() throws java.sql.SQLException;	1.0
close() throws java.sql.SQLException;	1.0
clearWarnings() throws java.sql.SQLException;	1.0
commit() throws java.sql.SQLException;	1.0
createStatement() throws java.sql.SQLException;	1.0
createStatement(int resultSetType,int resultSetConcurrency) throws java.sql.SQLException;	2.0
getAutoCommit() throws java.sql.SQLException;	1.0
getCatalog() throws java.sql.SQLException;	1.0
getHoldability() throws java.sql.SQLException;	3.0
getMetaData() throws java.sql.SQLException;	1.0
getTransactionIsolation() throws java.sql.SQLException;	1.0
getWarnings() throws java.sql.SQLException;	1.0
isClosed() throws java.sql.SQLException;	1.0

prepareCall(java.lang.String sql) throws java.sql.SQLException;	1.0
prepareCall(java.lang.String sql,int resultSetType,int resultSetConcurrency) throws java.sql.SQLException;	2.0
prepareStatement(java.lang.String sql) throws java.sql.SQLException;	1.0
prepareStatement(java.lang.String sql,int resultSetType,int resultSetConcurrency,int resultSetHoldability) throws java.sql.SQLException;	3.0
rollback() throws java.sql.SQLException;	1.0
setAutoCommit(boolean enableAutoCommit) throws java.sql.SQLException;	1.0
setHoldability(int holdability) throws java.sql.SQLException;	3.0
setTransactionIsolation(int level) throws java.sql.SQLException;	1.0

Table 4-5 java.sql.Connection

JAVA.SQL.DATABASEMETADATA

Methods	Version
allProceduresAreCallable()	1.0
allTablesAreSelectable() throws SQLException	1.0
dataDefinitionCausesTransactionCommit() throws SQLException	2.0
dataDefinitionIgnoredInTransactions() throws SQLException	3.0
deletesAreDetected(int type) throws SQLException	1.0
doesMaxRowSizeIncludeBlobs() throws SQLException	1.0

getAttributes(String catalog, String schemaPattern,String typeNamePattern, String attributeNamePattern) throws SQLException	1.0
getBestRowIdentifier(String catalog, String schema,String table, int scope, boolean nullable) throws SQLException	1.0
getCatalogSeparator() throws SQLException	2.0
getCatalogTerm() throws SQLException	1.0
getCatalogs() throws SQLException	1.0
getColumnPrivileges(String catalog, String schema,String table, String columnNamePattern) throws SQLException	1.0
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern) throws SQLException	1.0
getConnection() throws SQLException	1.0
getCrossReference(String parentCatalog, String parentSchema,String parentTable, String foreignCatalog, String foreignSchema,String foreignTable) throws SQLException	1.0
getDatabaseMajorVersion() throws SQLException	1.0
getDatabaseMinorVersion() throws SQLException	3.0
getDatabaseProductName() throws SQLException	3.0
getDatabaseProductVersion() throws SQLException	1.0
getDefaultTransactionIsolation() throws SQLException	1.0
getDriverMajorVersion()	1.0
getDriverMinorVersion()	1.0

getDriverName() throws SQLException	1.0
getDriverVersion() throws SQLException	1.0
getExportedKeys(String catalog, String schema, String table)throws SQLException	1.0
getExtraNameCharacters() throws SQLException	1.0
getIdentifierQuoteString() throws SQLException	1.0
getImportedKeys(String catalog, String schema, String table) throws SQLException	1.0
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate) throws SQLException	1.0
getJDBCMajorVersion() throws SQLException	1.0
getJDBCMinorVersion() throws SQLException	3.0
getMaxBinaryLiteralLength() throws SQLException	3.0
getMaxCatalogNameLength() throws SQLException	1.0
getMaxCharLiteralLength() throws SQLException	1.0
getMaxColumnNameLength() throws SQLException	1.0
getMaxColumnsInGroupBy() throws SQLException	1.0
getMaxColumnsInIndex() throws SQLException	1.0
getMaxColumnsInOrderBy() throws SQLException	1.0
getMaxColumnsInSelect() throws SQLException	1.0
getMaxColumnsInTable() throws SQLException	1.0
getMaxConnections() throws SQLException	1.0

getMaxCursorNameLength() throws SQLException	1.0
getMaxIndexLength() throws SQLException	1.0
getMaxProcedureNameLength() throws SQLException	1.0
getMaxRowSize() throws SQLException	1.0
getMaxSchemaNameLength() throws SQLException	1.0
getMaxStatementLength() throws SQLException	1.0
getMaxStatements() throws SQLException	1.0
getMaxTableNameLength() throws SQLException	1.0
getMaxTablesInSelect() throws SQLException	1.0
getMaxTablesInSelect() throws java.sql.SQLException;	1.0
getMaxUserNameLength() throws java.sql.SQLException;	1.0
getPrimaryKeys(java.lang.String catalog,,java.lang.String schema,,java.lang.String table) throws java.sql.SQLException;	1.0
getProcedureColumns(java.lang.String catalog,,java.lang.String schemaPattern,,java.lang.String procedureNamePattern,,java.lang.String columnNamePattern) throws java.sql.SQLException;	1.0
getProcedureTerm() throws java.sql.SQLException;	1.0
getProcedures(java.lang.String catalog,,java.lang.String schemaPattern,java.lang.String procedureNamePattern) throws java.sql.SQLException;	1.0
getSQLKeywords() throws java.sql.SQLException;	1.0
getSQLStateType() throws java.sql.SQLException;	3.0

getSchemaTerm() throws java.sql.SQLException;	1.0
getSchemas() throws java.sql.SQLException;	1.0
getSearchStringEscape() throws java.sql.SQLException;	1.0
getStringFunctions() throws java.sql.SQLException;	1.0
getSystemFunctions() throws java.sql.SQLException;	1.0
getTableTypes() throws java.sql.SQLException;	1.0
getTables(java.lang.String catalog,,java.lang.String schemaPattern,,java.lang.String tableNamePattern,,java.lang.String type[]) throws java.sql.SQLException;	1.0
getTimeDateFunctions() throws java.sql.SQLException;	1.0
getTypeInfo() throws java.sql.SQLException;	1.0
getUserName() throws java.sql.SQLException;	1.0
getVersionColumns(java.lang.String catalog,,java.lang.String schema,,java.lang.String table) throws java.sql.SQLException;	1.0
insertsAreDetected(int type) throws java.sql.SQLException;	2.0
isCatalogAtStart() throws java.sql.SQLException;	1.0
nullPlusNonNullIsNull() throws java.sql.SQLException;	1.0
nullsAreSortedAtEnd() throws java.sql.SQLException;	1.0
nullsAreSortedAtStart() throws java.sql.SQLException;	1.0
nullsAreSortedHigh() throws java.sql.SQLException;	1.0
nullsAreSortedLow() throws java.sql.SQLException;	1.0
othersDeletesAreVisible(int type) throws	2.0

java.sql.SQLException;	
othersInsertsAreVisible(int type) throws java.sql.SQLException;	2.0
othersUpdatesAreVisible(int type) throws java.sql.SQLException;	2.0
ownDeletesAreVisible(int type) throws java.sql.SQLException;	2.0
ownInsertsAreVisible(int type) throws java.sql.SQLException;	2.0
ownUpdatesAreVisible(int type) throws java.sql.SQLException;	2.0
storesLowerCaseIdentifiers() throws java.sql.SQLException;	1.0
storesLowerCaseQuotedIdentifiers() throws java.sql.SQLException;	1.0
storesMixedCaseIdentifiers() throws java.sql.SQLException;	1.0
storesMixedCaseQuotedIdentifiers() throws java.sql.SQLException;	1.0
storesUpperCaseIdentifiers() throws java.sql.SQLException;	1.0
storesUpperCaseQuotedIdentifiers() throws java.sql.SQLException;	1.0
supportsANSI92EntryLevelSQL() throws java.sql.SQLException;	1.0
supportsANSI92FullSQL() throws java.sql.SQLException;	1.0
supportsANSI92IntermediateSQL() throws java.sql.SQLException;	1.0
supportsAlterTableWithAddColumn() throws	1.0

java.sql.SQLException;	
supportsAlterTableWithDropColumn() throws java.sql.SQLException;	1.0
supportsBatchUpdates() throws java.sql.SQLException;	2.0
supportsCatalogsInDataManipulation() throws java.sql.SQLException;	1.0
supportsCatalogsInIndexDefinitions() throws java.sql.SQLException;	1.0
supportsCatalogsInPrivilegeDefinitions() throws java.sql.SQLException;	1.0
supportsCatalogsInProcedureCalls() throws java.sql.SQLException;	1.0
supportsCatalogsInTableDefinitions() throws java.sql.SQLException;	1.0
supportsColumnAliasing() throws java.sql.SQLException;	1.0
supportsConvert() throws java.sql.SQLException;	1.0
supportsConvert(int fromType,int toType) throws java.sql.SQLException;	1.0
supportsCoreSQLGrammar() throws java.sql.SQLException;	1.0
supportsCorrelatedSubqueries() throws java.sql.SQLException;	1.0
supportsDataDefinitionAndDataManipulationTransactions( ) throws java.sql.SQLException;	1.0
supportsDataManipulationTransactionsOnly() throws java.sql.SQLException;	1.0

supportsDifferentTableCorrelationNames() throws java.sql.SQLException;	1.0
supportsExpressionsInOrderBy() throws java.sql.SQLException;	1.0
supportsExtendedSQLGrammar() throws java.sql.SQLException;	1.0
supportsFullOuterJoins() throws java.sql.SQLException;	1.0
supportsGetGeneratedKeys() throws java.sql.SQLException;	3.0
supportsGroupBy() throws java.sql.SQLException;	1.0
supportsGroupByBeyondSelect() throws java.sql.SQLException;	1.0
supportsGroupByUnrelated() throws java.sql.SQLException;	1.0
supportsIntegrityEnhancementFacility() throws java.sql.SQLException;	1.0
supportsLikeEscapeClause() throws java.sql.SQLException;	1.0
supportsLimitedOuterJoins() throws java.sql.SQLException;	1.0
supportsMinimumSQLGrammar() throws java.sql.SQLException;	1.0
supportsMixedCaseIdentifiers() throws java.sql.SQLException;	1.0
supportsMixedCaseQuotedIdentifiers() throws java.sql.SQLException;	1.0
supportsMultipleOpenResults() throws java.sql.SQLException;	3.0

supportsMultipleTransactions() throws java.sql.SQLException;	1.0
supportsNamedParameters() throws java.sql.SQLException;	3.0
supportsNonNullableColumns() throws java.sql.SQLException;	1.0
supportsOpenCursorsAcrossCommit() throws java.sql.SQLException;	1.0
supportsOpenCursorsAcrossRollback() throws java.sql.SQLException;	1.0
supportsOpenStatementsAcrossCommit() throws java.sql.SQLException;	1.0
supportsOpenStatementsAcrossRollback() throws java.sql.SQLException;	1.0
supportsOrderByUnrelated() throws java.sql.SQLException;	1.0
supportsOuterJoins() throws java.sql.SQLException;	1.0
supportsPositionedDelete() throws java.sql.SQLException;	1.0
supportsPositionedUpdate() throws java.sql.SQLException;	1.0
supportsResultSetConcurrency(int type,int concurrency) throws java.sql.SQLException;	2.0
supportsResultSetHoldability(int holdability) throws java.sql.SQLException;	3.0
supportsResultSetType(int type) throws java.sql.SQLException;	2.0
supportsSavepoints() throws java.sql.SQLException;	3.0
supportsSchemasInDataManipulation() throws	1.0

java.sql.SQLException;	
supportsSchemasInIndexDefinitions() throws java.sql.SQLException;	1.0
supportsSchemasInPrivilegeDefinitions() throws java.sql.SQLException;	1.0
supportsSchemasInProcedureCalls() throws java.sql.SQLException;	1.0
supportsSchemasInTableDefinitions() throws java.sql.SQLException;	1.0
supportsSelectForUpdate() throws java.sql.SQLException;	1.0
supportsStoredProcedures() throws java.sql.SQLException;	1.0
supportsSubqueriesInComparisons() throws java.sql.SQLException;	1.0
supportsSubqueriesInExists() throws java.sql.SQLException;	1.0
supportsSubqueriesInIns() throws java.sql.SQLException;	1.0
supportsSubqueriesInQuantifieds() throws java.sql.SQLException;	1.0
supportsTableCorrelationNames() throws java.sql.SQLException;	1.0
supportsTransactionIsolationLevel(int level) throws java.sql.SQLException;	1.0
supportsTransactions() throws java.sql.SQLException;	1.0
supportsUnion() throws java.sql.SQLException;	1.0
supportsUnionAll() throws java.sql.SQLException;	1.0
updatesAreDetected(int type) throws	2.0

java.sql.SQLException;	
usesLocalFilePerTable() throws java.sql.SQLException;	1.0
usesLocalFiles() throws java.sql.SQLException;	1.0

Table 4-6 java.sql.DatabaseMetaData

JAVA.SQL.DRVIER

Methods	Version
acceptsURL(java.lang.String url) throws java.sql.SQLException;	1.0
jdbcCompliant() throws ;	1.0
connect(java.lang.String url,,java.util.Properties info) throws java.sql.SQLException;	1.0
getMajorVersion() throws ;	1.0
getMinorVersion() throws ;	1.0

Table 4-7 java.sql.Driver

JAVA.SQL.PARAMETERMETADATA

Methods	Version
getPrecision(int param) throws java.sql.SQLException;	3.0
getScale(int param) throws java.sql.SQLException;	3.0
isNullable(int param) throws java.sql.SQLException;	3.0
isSigned(int param) throws java.sql.SQLException;	3.0
getParameterClassName(int param) throws java.sql.SQLException;	3.0
getParameterCount() throws java.sql.SQLException;	3.0
getParameterMode(int param) throws java.sql.SQLException;	3.0

getParameterType(int param) throws java.sql.SQLException;	3.0
getParameterTypeName(int param) throws java.sql.SQLException;	3.0

Table 4-8 java.sql.ParameterMetaData

JAVA.SQL.PREPAREDSTATEMENT

Methods	Version
setBoolean(int parameterIndex,boolean b) throws java.sql.SQLException;	1.0
setByte(int parameterIndex,byte x) throws java.sql.SQLException;	1.0
setDouble(int parameterIndex,double x) throws java.sql.SQLException;	1.0
setFloat(int parameterIndex,float x) throws java.sql.SQLException;	1.0
setInt(int parameterIndex,int x) throws java.sql.SQLException;	1.0
setShort(int parameterIndex,short x) throws java.sql.SQLException;	1.0
setTimestamp(int parameterIndex,,java.sql.Timestamp x,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
setTimestamp(int parameterIndex,,java.sql.Timestamp x) throws java.sql.SQLException;	1.0
execute() throws java.sql.SQLException;	1.0
getMetaData() throws java.sql.SQLException;	2.0
clearParameters() throws java.sql.SQLException;	1.0
setAsciiStream(int parameterIndex,,java.io.InputStream fin,int length) throws java.sql.SQLException;	1.0
setBinaryStream(int parameterIndex,,java.io.InputStream fin,int	1.0

length) throws java.sql.SQLException;	
setBlob(int parameterIndex,,java.sql.Blob x) throws java.sql.SQLException;	2.0
setBytes(int parameterIndex,byte x[]) throws java.sql.SQLException;	1.0
setCharacterStream(int parameterIndex,,java.io.Reader reader,int length) throws java.sql.SQLException;	2.0
setClob(int parameterIndex,,java.sql.Clob x) throws java.sql.SQLException;	2.0
setDate(int parameterIndex,,java.sql.Date x,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
setDate(int parameterIndex,,java.sql.Date x) throws java.sql.SQLException;	1.0
setNull(int parameterIndex,int jdbcype) throws java.sql.SQLException;	1.0
setObject(int parameterIndex,,java.lang.Object x) throws java.sql.SQLException;	2.0
setObject(int parameterIndex,,java.lang.Object x,int targetJdbcType,int scale) throws java.sql.SQLException;	1.0
setObject(int parameterIndex,,java.lang.Object x,int targetJdbcType) throws java.sql.SQLException;	1.0
setString(int parameterIndex,,java.lang.String x) throws java.sql.SQLException;	1.0
setTime(int parameterIndex,,java.sql.Time x) throws java.sql.SQLException;	1.0
setTime(int parameterIndex,,java.sql.Time x,,java.util.Calendar cal) throws java.sql.SQLException;	2.0

executeQuery() throws java.sql.SQLException;	1.0
executeUpdate() throws java.sql.SQLException;	1.0
getParameterMetaData() throws java.sql.SQLException;	3.0
close() throws java.sql.SQLException;	1.0
execute(java.lang.String sql) throws java.sql.SQLException;	1.0
getConnection() throws java.sql.SQLException;	2.0
clearWarnings() throws java.sql.SQLException;	1.0
getWarnings() throws java.sql.SQLException;	1.0
getFetchDirection() throws java.sql.SQLException;	2.0
getFetchSize() throws java.sql.SQLException;	2.0
setFetchDirection(int direction) throws java.sql.SQLException;	2.0
setFetchSize(int rows) throws java.sql.SQLException;	2.0
getMaxRows() throws java.sql.SQLException;	1.0
setMaxRows(int max) throws java.sql.SQLException;	1.0
getResultSet() throws java.sql.SQLException;	1.0
cancel() throws java.sql.SQLException;	1.0
executeQuery(java.lang.String sql) throws java.sql.SQLException;	1.0
executeUpdate(java.lang.String sql) throws java.sql.SQLException;	1.0
getResultSetConcurrency() throws java.sql.SQLException;	2.0
getResultSetHoldability() throws java.sql.SQLException;	3.0
getResultSetType() throws java.sql.SQLException;	2.0
getUpdateCount() throws java.sql.SQLException;	1.0

setCursorName(java.lang.String name) throws java.sql.SQLException;	1.0

Table 4-9 java.sql.PreparedStatement

JAVA.SQL.RESULTSET

Methods	Version
getObject(int columnIndex) throws java.sql.SQLException;	1.0
getObject(java.lang.String columnName) throws java.sql.SQLException;	1.0
getBoolean(int columnIndex) throws java.sql.SQLException;	1.0
getBoolean(java.lang.String columnName) throws java.sql.SQLException;	1.0
getByte(int columnIndex) throws java.sql.SQLException;	1.0
getByte(java.lang.String columnName) throws java.sql.SQLException;	1.0
getShort(int columnIndex) throws java.sql.SQLException;	1.0
getShort(java.lang.String columnName) throws java.sql.SQLException;	1.0
getInt(int columnIndex) throws java.sql.SQLException;	1.0
getInt(java.lang.String columnName) throws java.sql.SQLException;	1.0
getLong(java.lang.String columnName) throws java.sql.SQLException;	1.0
getLong(int columnIndex) throws java.sql.SQLException;	1.0
getFloat(java.lang.String columnName) throws	1.0

java.sql.SQLException;	
getFloat(int columnIndex) throws java.sql.SQLException;	1.0
getDouble(java.lang.String columnName) throws java.sql.SQLException;	1.0
getDouble(int columnIndex) throws java.sql.SQLException;	1.0
getBytes(java.lang.String columnName) throws java.sql.SQLException;	1.0
getBytes(int columnIndex) throws java.sql.SQLException;	1.0
getArray(int columnIndex) throws java.sql.SQLException;	2.0
next() throws java.sql.SQLException;	1.0
getType() throws java.sql.SQLException;	2.0
previous() throws java.sql.SQLException;	2.0
close() throws java.sql.SQLException;	1.0
getRef(java.lang.String columnName) throws java.sql.SQLException;	2.0
getRef(int columnIndex) throws java.sql.SQLException;	2.0
getDate(int columnIndex,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
getDate(int columnIndex) throws java.sql.SQLException;	1.0
getDate(java.lang.String columnName) throws java.sql.SQLException;	1.0
getDate(java.lang.String columnName,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
first() throws java.sql.SQLException;	2.0

last() throws java.sql.SQLException;	2.0
clearWarnings() throws java.sql.SQLException;	1.0
getMetaData() throws java.sql.SQLException;	1.0
getWarnings() throws java.sql.SQLException;	1.0
absolute(int row) throws java.sql.SQLException;	2.0
afterLast() throws java.sql.SQLException;	2.0
beforeFirst() throws java.sql.SQLException;	2.0
cancelRowUpdates() throws java.sql.SQLException;	2.0
deleteRow() throws java.sql.SQLException;	2.0
findColumn(java.lang.String columnName) throws java.sql.SQLException;	1.0
getAsciiStream(java.lang.String columnName) throws java.sql.SQLException;	1.0
getAsciiStream(int columnIndex) throws java.sql.SQLException;	1.0
getBigDecimal(java.lang.String columnName) throws java.sql.SQLException;	1.0
getBigDecimal(int columnIndex) throws java.sql.SQLException;	1.0
getBigDecimal(int columnIndex,int scale) throws java.sql.SQLException;	1.0
getBigDecimal(java.lang.String columnName,int scale) throws java.sql.SQLException;	1.0
getBinaryStream(java.lang.String columnName) throws java.sql.SQLException;	1.0
getBinaryStream(int columnIndex) throws java.sql.SQLException;	1.0

getBlob(int columnIndex) throws java.sql.SQLException;	2.0
getBlob(java.lang.String columnName) throws java.sql.SQLException;	2.0
getCharacterStream(int columnIndex) throws java.sql.SQLException;	2.0
getCharacterStream(java.lang.String columnName) throws java.sql.SQLException;	2.0
getClob(int columnIndex) throws java.sql.SQLException;	2.0
getClob(java.lang.String columnName) throws java.sql.SQLException;	2.0
getConcurrency() throws java.sql.SQLException;	2.0
getCursorName() throws java.sql.SQLException;	1.0
getFetchDirection() throws java.sql.SQLException;	2.0
getFetchSize() throws java.sql.SQLException;	2.0
getRow() throws java.sql.SQLException;	2.0
getStatement() throws java.sql.SQLException;	2.0
getString(java.lang.String columnName) throws java.sql.SQLException;	1.0
getString(int columnIndex) throws java.sql.SQLException;	1.0
getTime(java.lang.StringcolumnName,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
getTime(int columnIndex,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
getTime(int columnIndex) throws java.sql.SQLException;	1.0

getTime(java.lang.String columnName) throws java.sql.SQLException;	1.0
getTimestamp(java.lang.String columnName,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
getTimestamp(int columnIndex,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
getTimestamp(java.lang.String columnName) throws java.sql.SQLException;	1.0
getTimestamp(int columnIndex) throws java.sql.SQLException;	1.0
getUnicodeStream(java.lang.String columnName) throws java.sql.SQLException;	1.0
getUnicodeStream(int columnIndex) throws java.sql.SQLException;	1.0
insertRow() throws java.sql.SQLException;	2.0
isAfterLast() throws java.sql.SQLException;	2.0
isBeforeFirst() throws java.sql.SQLException;	2.0
isFirst() throws java.sql.SQLException;	2.0
isLast() throws java.sql.SQLException;	2.0
moveToCurrentRow() throws java.sql.SQLException;	2.0
moveToInsertRow() throws java.sql.SQLException;	2.0
refreshRow() throws java.sql.SQLException;	2.0
relative(int rows) throws java.sql.SQLException;	2.0
rowDeleted() throws java.sql.SQLException;	2.0
rowInserted() throws java.sql.SQLException;	2.0

rowUpdated() throws java.sql.SQLException;	2.0
setFetchDirection(int direction) throws java.sql.SQLException;	2.0
setFetchSize(int rows) throws java.sql.SQLException;	2.0
updateArray(int columnIndex,,java.sql.Array x) throws java.sql.SQLException;	3.0
updateArray(java.lang.String columnName,,java.sql.Array x) throws java.sql.SQLException;	3.0
updateAsciiStream(int columnIndex,,java.io.InputStream x,int length) throws java.sql.SQLException;	2.0
updateAsciiStream(java.lang.String columnName,,java.io.InputStream x,int length) throws java.sql.SQLException;	2.0
updateBigDecimal(int columnIndex,java.math.BigDecimal x) throws java.sql.SQLException;	2.0
updateBigDecimal(java.lang.String columnName,,java.math.BigDecimal x) throws java.sql.SQLException;	2.0
updateBinaryStream(int columnIndex,java.io.InputStream x,int length) throws java.sql.SQLException;	2.0
updateBinaryStream(java.lang.String columnName,,java.io.InputStream x,int length) throws java.sql.SQLException;	2.0
updateBlob(int columnIndex,,java.sql.Blob x) throws java.sql.SQLException;	3.0
updateBlob(java.lang.String columnName,,java.sql.Blob x) throws java.sql.SQLException;	3.0

updateBoolean(int columnIndex,boolean x) throws java.sql.SQLException;	2.0
updateBoolean(java.lang.String columnName,boolean x) throws java.sql.SQLException;	2.0
updateByte(int columnIndex,byte x) throws java.sql.SQLException;	2.0
updateByte(java.lang.String columnName,byte x) throws java.sql.SQLException;	2.0
updateBytes(int columnIndex,byte[] x) throws java.sql.SQLException;	2.0
updateBytes(java.lang.String columnName,byte[] x) throws java.sql.SQLException;	2.0
updateCharacterStream(java.lang.String columnName,,java.io.Reader x1,int length) throws java.sql.SQLException;	2.0
updateCharacterStream(int columnIndex,java.io.Reader x,int length) throws java.sql.SQLException;	2.0
updateClob(int columnIndex,java.sql.Clob x) throws java.sql.SQLException;	3.0
updateClob(java.lang.String columnName,,java.sql.Clob x) throws java.sql.SQLException;	3.0
updateDate(int columnIndex,java.sql.Date x) throws java.sql.SQLException;	2.0
updateDate(java.lang.String columnName,,java.sql.Date x) throws java.sql.SQLException;	2.0
updateDouble(int columnIndex,double x) throws java.sql.SQLException;	2.0

updateDouble(java.lang.String columnName,double x) throws java.sql.SQLException;	2.0
updateFloat(int columnIndex,float x) throws java.sql.SQLException;	2.0
updateFloat(java.lang.String columnName,float x) throws java.sql.SQLException;	2.0
updateInt(int columnIndex,int x) throws java.sql.SQLException;	2.0
updateInt(java.lang.String columnName,int x) throws java.sql.SQLException;	2.0
updateLong(java.lang.String columnName,long x) throws java.sql.SQLException;	2.0
updateLong(int columnIndex,long x) throws java.sql.SQLException;	2.0
updateNull(int columnIndex) throws java.sql.SQLException;	2.0
updateNull(java.lang.String columnName) throws java.sql.SQLException;	2.0
updateObject(java.lang.String columnName,,java.lang.Object x,int scale) throws java.sql.SQLException;	2.0
updateObject(java.lang.String columnName,,java.lang.Object x) throws java.sql.SQLException;	2.0
updateObject(int columnIndex,,java.lang.Object x) throws java.sql.SQLException;	2.0
updateObject(int columnIndex,,java.lang.Object x,int scale) throws java.sql.SQLException;	2.0
updateRef(java.lang.String columnName,,java.sql.Ref x) throws java.sql.SQLException;	3.0

updateRef(int columnIndex,java.sql.Ref x) throws java.sql.SQLException;	3.0
updateRow() throws java.sql.SQLException;	2.0
updateShort(java.lang.String columnName,short x) throws java.sql.SQLException;	2.0
updateShort(int columnIndex,short x) throws java.sql.SQLException;	2.0
updateString(int columnIndex,java.lang.String x) throws java.sql.SQLException;	2.0
updateString(java.lang.String columnName,,java.lang.String x) throws java.sql.SQLException;	2.0
updateTime(int columnIndex,,java.sql.Time x) throws java.sql.SQLException;	2.0
updateTime(java.lang.String columnName,,java.sql.Time x) throws java.sql.SQLException;	2.0
updateTimestamp(int columnIndex,,java.sql.Timestamp x) throws java.sql.SQLException;	2.0
updateTimestamp(java.lang.String columnName,,java.sql.Timestamp x) throws java.sql.SQLException;	2.0
wasNull() throws java.sql.SQLException;	1.0

Table 4-10 java.sql.ResultSet

JAVA.SQL.RESULTSETMETADATA

Methods	Version
isReadOnly(int column) throws java.sql.SQLException;	1.0

getCatalogName(int column) throws java.sql.SQLException;	1.0
getColumnClassName(int column) throws java.sql.SQLException;	2.0
getColumnCount() throws java.sql.SQLException;	1.0
getColumnDisplaySize(int column) throws java.sql.SQLException;	1.0
getColumnLabel(int column) throws java.sql.SQLException;	1.0
getColumnName(int column) throws java.sql.SQLException;	1.0
getColumnType(int column) throws java.sql.SQLException;	1.0
getColumnTypeName(int column) throws java.sql.SQLException;	1.0
getPrecision(int column) throws java.sql.SQLException;	1.0
getScale(int column) throws java.sql.SQLException;	1.0
getSchemaName(int column) throws java.sql.SQLException;	1.0
getTableName(int column) throws java.sql.SQLException;	1.0
isAutoIncrement(int column) throws java.sql.SQLException;	1.0
isCaseSensitive(int column) throws java.sql.SQLException;	1.0
isCurrency(int column) throws java.sql.SQLException;	1.0
isDefinitelyWritable(int column) throws	1.0

java.sql.SQLException;	
isNullable(int column) throws java.sql.SQLException;	1.0
isSearchable(int column) throws java.sql.SQLException;	1.0
isSigned(int column) throws java.sql.SQLException;	1.0
isWritable(int column) throws java.sql.SQLException;	1.0

Table 4-11 java.sql.ResultSetMetaData

JAVA.SQL.STATEMENT

Methods	Version
close() throws java.sql.SQLException;	1.0
execute(java.lang.String sql) throws java.sql.SQLException;	1.0
getConnection() throws java.sql.SQLException;	2.0
clearWarnings() throws java.sql.SQLException;	1.0
getWarnings() throws java.sql.SQLException;	1.0
getFetchDirection() throws java.sql.SQLException;	2.0
getFetchSize() throws java.sql.SQLException;	2.0
setFetchDirection(int direction) throws java.sql.SQLException;	2.0
setFetchSize(int rows) throws java.sql.SQLException;	1.0
getMaxRows() throws java.sql.SQLException;	1.0
setMaxRows(int max) throws java.sql.SQLException;	1.0
getResultSet() throws java.sql.SQLException;	1.0
cancel() throws java.sql.SQLException;	1.0
executeQuery(java.lang.String sql) throws	1.0

java.sql.SQLException;	
executeUpdate(java.lang.String sql) throws java.sql.SQLException;	1..0
getResultSetConcurrency() throws java.sql.SQLException;	2.0
getResultSetHoldability() throws java.sql.SQLException;	3.0
getResultSetType() throws java.sql.SQLException;	2.0
getUpdateCount() throws java.sql.SQLException;	1.0
setCursorName(java.lang.String name) throws java.sql.SQLException;	1.0

Table 4-12 java.sql.Statement

JAVAX.SQL.CONNECTIONPOOLDATASOURCE

Methods	Version
getLoginTimeout() throws java.sql.SQLException;	1.0
setLoginTimeout(int seconds) throws java.sql.SQLException;	1.0
getPooledConnection(java.lang.String user, ,java.lang.String password) throws java.sql.SQLException;	1.0
getPooledConnection() throws java.sql.SQLException;	1.0

Table 4-13 javax.sql.ConnectionPoolDataSource

javax.SQL.Datesource

Methods	Version
getConnection(java.lang.String user,,java.lang.String password) throws java.sql.SQLException;	1.0
getConnection() throws java.sql.SQLException;	1.0

getLoginTimeout() throws java.sql.SQLException;	1.0
setLoginTimeout(int seconds) throws java.sql.SQLException;	1.0

Table 4-14 javax.sql.DataSource

javax.sql.Pooledconnection

Methods	Version
close() throws java.sql.SQLException;	1.0
getConnection() throws java.sql.SQLException;	1.0
addConnectionEventListener(javax.sql.ConnectionEventListener Listener) ;	1.0
removeConnectionEventListener(javax.sql.ConnectionEventListener Listener) ;	1.0

Table 4-15 javax.sql.PooledConnection

javax.sql.XAConnection

Methods	Version
getXAResource() throws java.sql.SQLException;	1.0

Table 4-16 javax.sql.XAConnection

javax.transaction.xa.Xid

Methods	Version
getBranchQualifier() ;	1.0
getFormatId() ;	1.0
getGlobalTransactionId() ;	1.0

Table 4-17 javax.transaction.xa.Xid

JDBC Type III Driver

At present, DBMaker JDBC Type III Driver supports most useful interfaces and their methods for JDBC 2.0, JDBC 3.0 and JDBC 4.0 standard. These interface and methods are described below.

java.sql.Blob

Methods	Version
free() throws SQLException	4.0
length() throws SQLException	2.0
getBytes(long pos,int length) throws java.sql.SQLException;	2.0
getBinaryStream(long pos, long length) throws SQLException;	4.0
getBinaryStream() throws java.sql.SQLException;	2.0

Table 4-18 java.sql.Blob

java.sql.CallableStatement

Methods	Version
getBigDecimal(int parameterIndex) throws SQLException	2.0
getBlob(int parameterIndex) throws SQLException	2.0
getBoolean(int parameterIndex) throws SQLException	1.0
getByte(int parameterIndex) throws SQLException	1.0
getBytes(int parameterIndex) throws SQLException	1.0
getCharacterStream(int parameterIndex) throws SQLException	4.0
getClob(int parameterIndex) throws SQLException	2.0
getDate(int parameterIndex) throws SQLException	1.0

getDate(int parameterIndex, Calendar cal) throws SQLException	2.0
getDouble(int parameterIndex) throws SQLException	1.0
getFloat(int parameterIndex) throws SQLException	1.0
getInt(int parameterIndex)	1.0
getLong(int parameterIndex) throws SQLException	1.0
getNCharacterStream(int parameterIndex) throws SQLException	4.0
getNClob(int parameterIndex) throws SQLException	4.0
getNString(int parameterIndex) throws SQLException	4.0
getObject(int parameterIndex) throws SQLException	1.0
getShort(int parameterIndex) throws SQLException	1.0
getString(int parameterIndex) throws SQLException	1.0
getTime(int parameterIndex) throws SQLException	1.0
getTime(int parameterIndex, Calendar cal) throws SQLException	2.0
getTimestamp(int parameterIndex) throws SQLException	1.0
getTimestamp(int parameterIndex, Calendar cal) throws SQLException	2.0
registerOutParameter(int parameterIndex, int sqlType) throws SQLException	1.0
registerOutParameter(int parameterIndex, int sqlType, int scale) throws SQLException	1.0
registerOutParameter(int parameterIndex, int sqlType, String typeName) throws SQLException	2.0
getNClob(int parameterIndex) throws SQLException	4.0

getNString(int parameterIndex) throws SQLException	4.0
wasNull() throws SQLException	1.0

Table 4-19 java.sql.CallableStatement

java.sql.Clob

Methods	Version
free() throws SQLException	4.0
length() throws java.sql.SQLException;	2.0
getBytes(long pos, int length) throws SQLException	2.0
getAsciiStream() throws java.sql.SQLException;	2.0
getCharacterStream() throws SQLException	2.0
getCharacterStream(long pos, long length) throws SQLException	4.0
getSubString(long pos,int length) throws java.sql.SQLException;	2.0

Table 4-20 java.sql.Clob

java.sql.Connection

Methods	Version
clearWarnings() throws SQLException	1.0
close() throws SQLException	1.0
commit() throws SQLException	1.0
createStatement() throws SQLException	1.0
createStatement(int resultSetType, int resultSetConcurrency) throws SQLException	2.0
getAutoCommit() throws SQLException	1.0

getCatalog() throws SQLException	1.0
getHoldability() throws SQLException	3.0
getMetaData() throws SQLException	1.0
getTransactionIsolation() throws SQLException	1.0
getWarnings() throws SQLException	1.0
isClosed() throws SQLException	1.0
isReadOnly() throws SQLException	1.0
prepareCall(String sql) throws SQLException	1.0
prepareCall(String sql, int resultSetType,int resultSetConcurrency) throws SQLException	1.0
prepareStatement(String sql) throws SQLException	1.0
prepareStatement(String sql, int resultSetType,int resultSetConcurrency) throws SQLException	2.0
rollback() throws SQLException	1.0
setAutoCommit(boolean autoCommit) throws SQLException	1.0
setHoldability(int holdability) throws SQLException	3.0
setTransactionIsolation(int level) throws SQLException	1.0

Table 4-21 java.sql.Connection

java.sql.DatabaseMetaData

Methods	Version
allProceduresAreCallable() throws SQLException	1.0
allTablesAreSelectable() throws SQLException	1.0
dataDefinitionCausesTransactionCommit() throws SQLException	1.0

dataDefinitionIgnoredInTransactions() throws SQLException	1.0
deletesAreDetected(int type) throws SQLException	2.0
doesMaxRowSizeIncludeBlobs() throws SQLException	1.0
getAttributes(String catalog, String schemaPattern,String typeNamePattern, String attributeNamePattern) throws SQLException	3.0
getBestRowIdentifier(String catalog, String schema,String table, int scope, boolean nullable) throws SQLException	1.0
getCatalogSeparator() throws SQLException	1.0
getCatalogTerm() throws SQLException	1.0
getCatalogs() throws SQLException	1.0
getColumnPrivileges(String catalog, String schema,String table, String columnNamePattern) throws SQLException	1.0
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern) throws SQLException	1.0
getConnection() throws SQLException	2.0
getCrossReference(String parentCatalog, String parentSchema,String parentTable, String foreignCatalog, String foreignSchema,String foreignTable) throws SQLException	1.0
getDatabaseMajorVersion() throws SQLException	3.0
getDatabaseMinorVersion() throws SQLException	3.0
getDatabaseProductName() throws SQLException	1.0
getDatabaseProductVersion() throws SQLException	1.0
getDefaultTransactionIsolation() throws SQLException	1.0

getDriverMajorVersion()	1.0
getDriverMinorVersion()	1.0
getDriverName() throws SQLException	1.0
getDriverVersion() throws SQLException	1.0
getExportedKeys(String catalog, String schema, String table)throws SQLException	1.0
getExtraNameCharacters() throws SQLException	1.0
getIdentifierQuoteString() throws SQLException	1.0
getImportedKeys(String catalog, String schema, String table) throws SQLException	1.0
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate) throws SQLException	1.0
getJDBCMajorVersion() throws SQLException	3.0
getJDBCMinorVersion() throws SQLException	3.0
getMaxBinaryLiteralLength() throws SQLException	1.0
getMaxCatalogNameLength() throws SQLException	1.0
getMaxCharLiteralLength() throws SQLException	1.0
getMaxColumnNameLength() throws SQLException	1.0
getMaxColumnsInGroupBy() throws SQLException	1.0
getMaxColumnsInIndex() throws SQLException	1.0
getMaxColumnsInOrderBy() throws SQLException	1.0
getMaxColumnsInSelect() throws SQLException	1.0
getMaxColumnsInTable() throws SQLException	1.0

getMaxConnections() throws SQLException	1.0
getMaxCursorNameLength() throws SQLException	1.0
getMaxIndexLength() throws SQLException	1.0
getMaxProcedureNameLength() throws SQLException	1.0
getMaxRowSize() throws SQLException	1.0
getMaxSchemaNameLength() throws SQLException	1.0
getMaxStatementLength() throws SQLException	1.0
getMaxStatements() throws SQLException	1.0
getMaxTableNameLength() throws SQLException	1.0
getMaxTablesInSelect() throws SQLException	1.0
getMaxUserNameLength() throws SQLException	1.0
getNumericFunctions() throws SQLException	1.0
getPrimaryKeys(String catalog, String schema, String table)throws SQLException	1.0
getProcedureColumns(String catalog, String schemaPattern,String procedureNamePattern, String columnNamePattern)throws SQLException	1.0
getProcedureTerm() throws SQLException	1.0
getProcedures(String catalog, String schemaPattern,String procedureNamePattern) throws SQLException	1.0
getResultSetHoldability() throws SQLException	3.0
getSQLKeywords() throws SQLException	1.0
getSQLStateType() throws SQLException	3.0

getSchemaTerm() throws SQLException	1.0
getSchemas() throws SQLException;	1.0
getSearchStringEscape() throws SQLException	1.0
getStringFunctions() throws SQLException	1.0
getSuperTables(String catalog, String schemaPattern, String tableNamePattern) throws SQLException	3.0
getSuperTypes(String catalog, String schemaPattern,String typeNamePattern) throws SQLException	3.0
getSystemFunctions() throws SQLException	1.0
getTablePrivileges(String catalog, String schemaPattern,String tableNamePattern) throws SQLException	1.0
getTableTypes() throws SQLException	1.0
getTables(String catalog, String schemaPattern,String tableNamePattern, String[] types) throws SQLException	1.0
getTimeDateFunctions() throws SQLException	1.0
getTypeInfo() throws SQLException	1.0
getUDTs(String catalog, String schemaPattern,String typeNamePattern, int[] types) throws SQLException	2.0
getURL() throws SQLException	1.0
getUserName() throws SQLException	1.0
getVersionColumns(String catalog, String schema, String table) throws SQLException	1.0
insertsAreDetected(int type) throws SQLException	2.0
isCatalogAtStart() throws SQLException	1.0

isReadOnly() throws SQLException	1.0
locatorsUpdateCopy() throws SQLException	3.0
nullPlusNonNullIsNull() throws SQLException	1.0
nullsAreSortedAtEnd() throws SQLException	1.0
nullsAreSortedAtStart() throws SQLException	1.0
nullsAreSortedHigh() throws SQLException	1.0
nullsAreSortedLow() throws SQLException	1.0
othersDeletesAreVisible(int type) throws SQLException	2.0
othersInsertsAreVisible(int type) throws SQLException	2.0
othersUpdatesAreVisible(int type) throws SQLException	2.0
ownDeletesAreVisible(int type) throws SQLException	2.0
ownInsertsAreVisible(int type) throws SQLException	2.0
ownUpdatesAreVisible(int type) throws SQLException	2.0
storesLowerCaseIdentifiers() throws SQLException	2.0
storesLowerCaseQuotedIdentifiers() throws SQLException	1.0
storesMixedCaseIdentifiers() throws SQLException	1.0
storesMixedCaseQuotedIdentifiers() throws SQLException	1.0
storesUpperCaseIdentifiers() throws SQLException	1.0
storesUpperCaseQuotedIdentifiers() throws SQLException	1.0
supportsANSI92EntryLevelSQL() throws SQLException	1.0
supportsANSI92FullSQL() throws SQLException	1.0
supportsANSI92IntermediateSQL() throws SQLException	1.0

supportsAlterTableWithAddColumn() throws SQLException	1.0
supportsAlterTableWithDropColumn() throws SQLException	1.0
supportsBatchUpdates() throws SQLException	2.0
supportsCatalogsInDataManipulation() throws SQLException	1.0
supportsCatalogsInIndexDefinitions() throws SQLException	1.0
supportsCatalogsInPrivilegeDefinitions() throws SQLException	1.0
supportsCatalogsInProcedureCalls() throws SQLException	1.0
supportsCatalogsInTableDefinitions() throws SQLException	1.0
supportsColumnAliasing() throws SQLException	1.0
supportsConvert() throws SQLException	1.0
supportsConvert(int fromType, int toType) throws SQLException	1.0
supportsCoreSQLGrammar() throws SQLException	1.0
supportsCorrelatedSubqueries() throws SQLException	1.0
supportsDataDefinitionAndDataManipulationTransactions()throws SQLException	1.0
supportsDataManipulationTransactionsOnly() throws SQLException	1.0
supportsDifferentTableCorrelationNames() throws SQLException	1.0
supportsExpressionsInOrderBy() throws SQLException	1.0
supportsExtendedSQLGrammar() throws SQLException	1.0
supportsFullOuterJoins() throws SQLException	1.0
supportsGetGeneratedKeys() throws SQLException	3.0

supportsGroupBy() throws SQLException	1.0
supportsGroupByBeyondSelect() throws SQLException	1.0
supportsGroupByUnrelated() throws SQLException	1.0
supportsIntegrityEnhancementFacility() throws SQLException	1.0
supportsLikeEscapeClause() throws SQLException	1.0
supportsLimitedOuterJoins() throws SQLException	1.0
supportsMinimumSQLGrammar() throws SQLException	1.0
supportsMixedCaseIdentifiers() throws SQLException	1.0
supportsMixedCaseQuotedIdentifiers() throws SQLException	1.0
supportsMultipleOpenResults() throws SQLException	3.0
supportsMultipleResultSets() throws SQLException	1.0
supportsMultipleTransactions() throws SQLException	1.0
supportsNamedParameters() throws SQLException	3.0
supportsNonNullableColumns() throws SQLException	1.0
supportsOpenCursorsAcrossCommit() throws SQLException	1.0
supportsOpenCursorsAcrossRollback() throws SQLException	1.0
supportsOpenStatementsAcrossCommit() throws SQLException	1.0
supportsOpenStatementsAcrossRollback() throws SQLException	1.0
supportsOrderByUnrelated() throws SQLException	1.0
supportsOuterJoins() throws SQLException	1.0
supportsPositionedDelete() throws SQLException	1.0
supportsPositionedUpdate() throws SQLException	1.0

supportsResultSetConcurrency(int type, int concurrency)throws SQLException	2.0
supportsResultSetHoldability(int holdability)throws SQLException	3.0
supportsResultSetType(int type) throws SQLException	2.0
supportsSavepoints() throws SQLException	3.0
supportsSchemasInDataManipulation() throws SQLExceptionportsSubqueriesInExists() throws java.sql.SQLException;	1.0
supportsSchemasInIndexDefinitions() throws SQLException	1.0
supportsSchemasInPrivilegeDefinitions() throws SQLException	1.0
supportsSchemasInProcedureCalls() throws SQLException	1.0
supportsSchemasInTableDefinitions() throws SQLException	1.0
supportsSelectForUpdate() throws SQLException	1.0
supportsStatementPooling() throws SQLException	3.0
supportsStoredFunctionsUsingCallSyntax() throws SQLException	4.0
supportsStoredProcedures() throws SQLException	1.0
supportsSubqueriesInComparisons() throws SQLException	1.0
supportsSubqueriesInExists() throws SQLException	1.0
supportsSubqueriesInIns() throws SQLException	1.0
supportsSubqueriesInQuantifieds() throws SQLException	1.0
supportsTableCorrelationNames() throws SQLException	1.0
supportsTransactionIsolationLevel(int level) throws SQLException	1.0

supportsTransactions() throws SQLException	1.0
supportsUnion() throws SQLException	1.0
supportsUnionAll() throws SQLException	1.0
updatesAreDetected(int type) throws SQLException	2.0
usesLocalFilePerTable() throws SQLException	1.0
usesLocalFiles() throws SQLException	1.0

Table 4-22 java.sql.DatabaseMetaData

java.sql.Driver

Methods	Version
acceptsURL(String url) throws SQLException	1.0
connect(String url, Properties properties) throws SQLException	1.0
getMajorVersion()	1.0
getMinorVersion()	1.0
jdbcCompliant()	1.0

Table 4-23 java.sql.Driver

java.sql.ParameterMetaData

Methods	Version
getParameterClassName(int param) throws SQLException	3.0
geParameterCount() throws SQLException	3.0
getParameterMode(int param) throws SQLException	3.0
getParameterType(int param) throws SQLException	3.0
getParameterTypeName(int param) throws SQLException	3.0

getPrecision(int param) throws SQLException	3.0
getScale(int param) throws SQLException	3.0
isNullable(int param) throws SQLException	3.0
isSigned(int param) throws SQLException	3.0

Table 4-24 java.sql.ParameterMetaData

java.sql.PreparedStatement

Methods	Version
clearParameters() throws SQLException	1.0
execute() throws SQLException	1.0
executeQuery() throws SQLException	1.0
executeUpdate() throws SQLException	1.0
getMetaData() throws SQLException	2.0
getParameterMetaData() throws SQLException	3.0
setAsciiStream(int parameterIndex, InputStream x) throws SQLException	4.0
setAsciiStream(int parameterIndex, InputStream x, int length) throws SQLException	4.0
setAsciiStream(int parameterIndex, InputStream x, long length) throws SQLException	4.0
setBigDecimal(int parameterIndex, BigDecimal x) throws SQLException	1.0
setBinaryStream(int parameterIndex, InputStream x)throws SQLException	4.0
setBinaryStream(int parameterIndex, InputStream x, int	1.0

length)throws SQLException	
setBinaryStream(int parameterIndex, InputStream x, long length) throws SQLException	4.0
setBlob(int parameterIndex, Blob x) throws SQLException	2.0
setBlob(int parameterIndex, InputStream inputStream)throws SQLException	4.0
setBlob(int parameterIndex, InputStream inputStream, long length) throws SQLException	4.0
setBoolean(int parameterIndex, boolean x) throws SQLException	1.0
setByte(int parameterIndex, byte x) throws SQLException	1.0
setBytes(int parameterIndex, byte[] x) throws SQLException	1.0
setCharacterStream(int parameterIndex, Reader reader) throws SQLException	4.0
setCharacterStream(int parameterIndex, Reader reader, int length) throws SQLException	2.0
setCharacterStream(int parameterIndex, Reader reader, long length)throws SQLException	4.0
setClob(int parameterIndex, Clob x) throws SQLException	2.0
setClob(int parameterIndex, Reader reader) throws SQLException	4.0
setClob(int parameterIndex, Reader reader, long length) throws SQLException	4.0
setDate(int parameterIndex, Date x) throws SQLException	1.0
setDate(int parameterIndex, Date x, Calendar cal) throws SQLException	2.0
setDouble(int parameterIndex, double x) throws SQLException	1.0

setFloat(int parameterIndex, float x) throws SQLException	1.0
setInt(int parameterIndex, int x) throws SQLException	1.0
setLong(int parameterIndex, long x) throws SQLException	1.0
setNCharacterStream(int parameterIndex, Reader value) throws SQLException	4.0
setNCharacterStream(int parameterIndex, Reader value, long length)throws SQLException	4.0
setNClob(int parameterIndex, NClob value) throws SQLException	4.0
setNClob(int parameterIndex, Reader reader) throws SQLException	4.0
setNClob(int parameterIndex, Reader reader, long length)throws SQLException	4.0
setNString(int parameterIndex, String value) throws SQLException	4.0
setNull(int parameterIndex, int sqlType) throws SQLException	1.0
setNull(int parameterIndex, int sqlType, String typeName) throws SQLException	2.0
setObject(int parameterIndex, Object x) throws SQLException	1.0
setObject(int parameterIndex, Object x, int targetSqlType)throws SQLException	1.0
setObject(int parameterIndex, Object x, int targetSqlType,int scaleOrLength) throws SQLException	4.0
setShort(int parameterIndex, short x) throws SQLException	1.0
setString(int parameterIndex, String x) throws SQLException	1.0
setTime(int parameterIndex, Time x) throws SQLException	1.0
setTime(int parameterIndex, Time x, Calendar cal) throws	2.0

SQLException	
setTimestamp(int parameterIndex, Timestamp x) throws SQLException	1.0
setTimestamp(int parameterIndex, Timestamp x, Calendar cal) throws SQLException	2.0
setURL(int parameterIndex, URL x) throws SQLException	3.0
setUnicodeStream(int parameterIndex, InputStream x, int length) throws SQLException	1.0
setNCharacterStream(int parameterIndex, Reader value) throws SQLException	4.0
setNCharacterStream(int parameterIndex, Reader value, long length) throws SQLException	4.0
setNClob(int parameterIndex, NClob value) throws SQLException	4.0
setNClob(int parameterIndex, Reader reader) throws SQLException	4.0
setNClob(int parameterIndex, Reader reader, long length) throws SQLException	4.0
setNString(int parameterIndex, String value) throws SQLException	4.0

Table 4-25 java.sql.PreparedStatement

java.sql.ResultSet

Methods	Version
absolute(int row) throws SQLException	2.0
afterLast() throws SQLException	2.0
beforeFirst() throws SQLException	2.0
cancelRowUpdates() throws SQLException	2.0

clearWarnings() throws SQLException	1.0
close() throws SQLException	1.0
deleteRow() throws SQLException	2.0
findColumn(String columnLabel) throws SQLException	1.0
first() throws SQLException	2.0
getAsciiStream(int columnIndex) throws SQLException	1.0
getAsciiStream(String columnLabel) throws SQLException	1.0
getBigDecimal(int columnIndex) throws SQLException	2.0
getBigDecimal(String columnLabel) throws SQLException	2.0
getBigDecimal(int columnIndex, int scale) throws SQLException	1.0
getBigDecimal(String columnLabel, int scale)throws SQLException	1.0
getBinaryStream(int columnIndex) throws SQLException	1.0
getBinaryStream(String columnLabel) throws SQLException	1.0
getBlob(int columnIndex) throws SQLException	2.0
getBlob(String columnLabel) throws SQLException	2.0
getBoolean(int columnIndex) throws SQLException	1.0
getBoolean(String columnLabel) throws SQLException	1.0
getByte(int columnIndex) throws SQLException	1.0
getByte(String columnLabel) throws SQLException	1.0
getBytes(int columnIndex) throws SQLException;	1.0
getBytes(String columnLabel) throws SQLException	1.0
getCharacterStream(int columnIndex) throws SQLException	2.0

getCharacterStream(String columnLabel) throws SQLException	2.0
getClob(int columnIndex) throws SQLException	2.0
getClob(String columnLabel) throws SQLException	2.0
getConcurrency() throws SQLException	2.0
getCursorName() throws SQLException	1.0
getDate(int columnIndex) throws SQLException	1.0
getDate(String columnLabel) throws SQLException	1.0
getDate(int columnIndex, Calendar cal) throws SQLException	2.0
getDate(String columnLabel, Calendar cal) throws SQLException	2.0
getDouble(int columnIndex) throws SQLException	1.0
getDouble(String columnLabel) throws SQLException	1.0
getFetchDirection() throws SQLException	2.0
getFetchSize() throws SQLException	2.0
getFloat(int columnIndex) throws SQLException	1.0
getFloat(String columnLabel) throws SQLException	1.0
getInt(int columnIndex) throws SQLException	1.0
getInt(String columnLabel) throws SQLException	1.0
getLong(int columnIndex) throws SQLException	1.0
getLong(String columnLabel) throws SQLException	1.0
getMetaData() throws SQLException	1.0
getNCharacterStream(int columnIndex) throws SQLException	4.0
getNCharacterStream(String columnLabel) throws SQLException	4.0

getNClob(int columnIndex) throws SQLException	4.0
getNClob(String columnLabel) throws SQLException	4.0
getNString(int columnIndex) throws SQLException	4.0
getNString(String columnLabel) throws SQLException	4.0
getObject(int columnIndex) throws SQLException	1.0
getObject(String columnLabel) throws SQLException	1.0
getRow() throws SQLException	2.0
getShort(int columnIndex) throws SQLException	1.0
getShort(String columnLabel) throws SQLException	1.0
getStatement() throws SQLException	2.0
getString(int columnIndex) throws SQLException	1.0
getString(String columnLabel) throws SQLException	1.0
getTime(int columnIndex) throws SQLException	1.0
getTime(String columnLabel) throws SQLException	1.0
getTime(int columnIndex, Calendar cal) throws SQLException	2.0
getTime(String columnLabel, Calendar cal) throws SQLException	2.0
getTimestamp(int columnIndex) throws SQLException	1.0
getTimestamp(String columnLabel) throws SQLException	1.0
getTimestamp(int columnIndex, Calendar cal)throws SQLException	2.0
getTimestamp(String columnLabel, Calendar cal) throws SQLException	2.0
getType() throws SQLException	2.0

getWarnings() throws SQLException	1.0
insertRow() throws SQLException	2.0
isAfterLast() throws SQLException	2.0
isBeforeFirst() throws SQLException	2.0
isClosed() throws SQLException	4.0
last() throws SQLException	2.0
moveToCurrentRow() throws SQLException	2.0
moveToInsertRow() throws SQLException	2.0
next() throws SQLException	1.0
previous() throws SQLException	2.0
refreshRow() throws SQLException	2.0
relative(int rows) throws SQLException	2.0
rowDeleted() throws SQLException	2.0
rowInserted() throws SQLException	2.0
rowUpdated() throws SQLException	2.0
setFetchDirection(int direction) throws SQLException	2.0
setFetchSize(int rows) throws SQLException	2.0
updateAsciiStream(int columnIndex, InputStream x) throws SQLException	4.0
updateAsciiStream(String columnLabel, InputStream x)throws SQLException	4.0
updateAsciiStream(int columnIndex, InputStream x, int length) throws SQLException	2.0

updateAsciiStream(String columnLabel, InputStream x, int length)throws SQLException	2.0
updateAsciiStream(int columnIndex, InputStream x, long length)throws SQLException	4.0
updateAsciiStream(String columnLabel, InputStream x, long length)throws SQLException	4.0
updateBigDecimal(int columnIndex, BigDecimal x) throws SQLException	2.0
updateBigDecimal(String columnLabel, BigDecimal x)throws SQLException	2.0
updateBinaryStream(int columnIndex, InputStream x)throws SQLException	4.0
updateBinaryStream(String columnLabel, InputStream x)throws SQLException	4.0
updateBinaryStream(int columnIndex, InputStream x, int length)throws SQLException	2.0
updateBinaryStream(String columnLabel, InputStream x, int length)throws SQLException	2.0
updateBinaryStream(int columnIndex, InputStream x, long length) throws SQLException	4.0
updateBinaryStream(String columnLabel, InputStream x, long length) throws SQLException	4.0
updateBlob(int columnIndex, java.sql.Blob x) throws SQLException	3.0
updateBlob(String columnLabel, java.sql.Blob x) throws SQLException	3.0

updateBlob(int columnIndex, InputStream inputStream)throws SQLException	4.0
updateBlob(String columnLabel, InputStream inputStream) throws SQLException	4.0
updateBlob(int columnIndex, InputStream inputStream, long length) throws SQLException	4.0
updateBlob(String columnLabel, InputStream inputStream, long length) throws SQLException	4.0
updateBoolean(int columnIndex, boolean x) throws SQLException	2.0
updateBoolean(String columnLabel, boolean x)throws SQLException	2.0
updateByte(int columnIndex, byte x) throws SQLException	2.0
updateByte(String columnLabel, byte x) throws SQLException	2.0
updateBytes(int columnIndex, byte[] x) throws SQLException	2.0
updateBytes(String columnLabel, byte[] x) throws SQLException	2.0
updateCharacterStream(int columnIndex, Reader x)throws SQLException	4.0
updateCharacterStream(String columnLabel, Reader reader)throws SQLException	4.0
updateCharacterStream(int columnIndex, Reader x, int length)throws SQLException	2.0
updateCharacterStream(String columnLabel, Reader reader,int length) throws SQLException	2.0
updateCharacterStream(int columnIndex, Reader x, long length) throws SQLException	4.0

updateCharacterStream(String columnLabel, Reader reader,long length) throws SQLException	4.0
updateClob(int columnIndex, java.sql.Clob x) throws SQLException	3.0
updateClob(String columnLabel, java.sql.Clob x) throws SQLException	3.0
updateClob(int columnIndex, Reader reader) throws SQLException	4.0
updateClob(String columnLabel, Reader reader) throws SQLException	4.0
updateClob(int columnIndex, Reader reader, long length) throws SQLException	4.0
updateClob(String columnLabel, Reader reader, long length)throws SQLException	4.0
updateDate(int columnIndex, Date x) throws SQLException	2.0
updateDate(String columnLabel, Date x) throws SQLException	2.0
updateDouble(int columnIndex, double x) throws SQLException	2.0
updateDouble(String columnLabel, double x) throws SQLException	2.0
updateFloat(int columnIndex, float x) throws SQLException	2.0
updateFloat(String columnLabel, float x) throws SQLException	2.0
updateInt(int columnIndex, int x) throws SQLException	2.0
updateInt(String columnLabel, int x) throws SQLException	2.0
updateLong(int columnIndex, long x) throws SQLException	2.0
updateLong(String columnLabel, long x) throws SQLException	2.0

updateNCharacterStream(int columnIndex, Reader x) throws SQLException	4.0
updateNCharacterStream(String columnLabel, Reader reader) throws SQLException	4.0
updateNCharacterStream(int columnIndex, Reader x, long length)throws SQLException	4.0
updateNCharacterStream(String columnLabel, Reader reader,long length) throws SQLException	4.0
updateNClob(int columnIndex, java.sql.NClob nClob) throws SQLException	4.0
updateNClob(String columnLabel, java.sql.NClob nClob) throws SQLException	4.0
updateNClob(int columnIndex, Reader reader) throws SQLException	4.0
updateNClob(String columnLabel, Reader reader)throws SQLException	4.0
updateNClob(int columnIndex, Reader reader, long length)throws SQLException	4.0
updateNClob(String columnLabel, Reader reader, long length) throws SQLException	4.0
updateNString(int columnIndex, String nString)throws SQLException	4.0
updateNString(String columnLabel, String nString) throws SQLException	4.0
updateNull(int columnIndex) throws SQLException	2.0
updateNull(String columnLabel) throws SQLException	2.0

updateObject(int columnIndex, Object x) throws SQLException	2.0
updateObject(String columnLabel, Object x) throws SQLException	2.0
updateRow() throws SQLException	2.0
updateShort(int columnIndex, short x) throws SQLException	2.0
updateShort(String columnLabel, short x) throws SQLException	2.0
updateString(int columnIndex, String x) throws SQLException	2.0
updateString(String columnLabel, String x) throws SQLException	2.0
updateTime(int columnIndex, Time x) throws SQLException	2.0
updateTime(String columnLabel, Time x) throws SQLException	2.0
updateTimestamp(int columnIndex, Timestamp x) throws SQLException	2.0
updateTimestamp(String columnLabel, Timestamp x)throws SQLException	2.0
getNCharacterStream(int columnIndex) throws SQLException	4.0
getNCharacterStream(String columnLabel) throws SQLException	4.0
getNClob(int columnIndex) throws SQLException	4.0
getNClob(String columnLabel) throws SQLException	4.0
getNString(int columnIndex) throws SQLException	4.0
getNString(String columnLabel) throws SQLException	4.0
updateNCharacterStream(int columnIndex, Reader x) throws SQLException	4.0
updateNCharacterStream(String columnLabel, Reader reader) throws SQLException	4.0

updateNCharacterStream(int columnIndex, Reader x, long length)throws SQLException	4.0
updateNCharacterStream(String columnLabel, Reader reader,long length) throws SQLException	4.0
updateNClob(int columnIndex, java.sql.NClob nClob) throws SQLException	4.0
updateNClob(String columnLabel, java.sql.NClob nClob) throws SQLException	4.0
updateNClob(int columnIndex, Reader reader) throws SQLException	4.0
updateNClob(String columnLabel, Reader reader)throws SQLException	4.0
updateNClob(int columnIndex, Reader reader, long length)throws SQLException	4.0
updateNClob(String columnLabel, Reader reader, long length) throws SQLException	4.0
updateNString(int columnIndex, String nString)throws SQLException	4.0
updateNString(String columnLabel, String nString) throws SQLException	4.0
wasNull() throws SQLException	1.0

Table 4-26 java.sql.ResultSet

java.sql.ResultSetMetaData

Methods	Version
getCatalogName(int column) throws SQLException	1.0

getColumnClassName(int column) throws SQLException	2.0
getColumnCount() throws SQLException	1.0
getColumnDisplaySize(int column) throws SQLException	1.0
getColumnLabel(int column) throws SQLException	1.0
getColumnName(int column) throws SQLException	1.0
getColumnType(int column) throws SQLException	1.0
getColumnTypeName(int column) throws SQLException	1.0
getPrecision(int column) throws SQLException	1.0
getScale(int column) throws SQLException	1.0
getSchemaName(int column) throws SQLException	1.0
getTableName(int column) throws SQLException	1.0
isAutoIncrement(int column) throws SQLException	1.0
isCaseSensitive(int column) throws SQLException	1.0
isCurrency(int column) throws SQLException	1.0
isDefinitelyWritable(int column) throws SQLException	1.0
isNullable(int column) throws SQLException	1.0
isReadOnly(int column) throws SQLException	1.0
isSearchable(int column) throws SQLException	1.0
isSigned(int column) throws SQLException	1.0
isWritable(int column) throws SQLException	1.0

Table 4-27 java.sql.ResultSetMetaData

java.sql.Statement

Methods	Version
cancel() throws SQLException	1.0
clearWarnings() throws SQLException	1.0
close() throws SQLException	1.0
execute(String sql) throws SQLException	1.0
executeQuery(String sql) throws SQLException	1.0
executeUpdate(String sql) throws SQLException	1.0
getConnection() throws SQLException	2.0
getFetchDirection() throws SQLException	2.0
getFetchSize() throws SQLException	2.0
getMaxFieldSize() throws SQLException	1.0
getMaxRows() throws SQLException	1.0
getMoreResults() throws SQLException	1.0
getMoreResults(int current) throws SQLException	3.0
getQueryTimeout() throws SQLException	1.0
getResultSet() throws SQLException	1.0
getResultSetConcurrency() throws SQLException	2.0
getResultSetHoldability() throws SQLException	3.0
getResultSetType() throws SQLException	2.0
getUpdateCount() throws SQLException	1.0
getWarnings() throws SQLException	1.0
isClosed() throws SQLException	4.0

setCursorName(String name) throws SQLException	1.0
setFetchDirection(int direction) throws SQLException	2.0
setFetchSize(int rows) throws SQLException	2.0
setMaxFieldSize(int max) throws SQLException	1.0
setMaxRows(int max) throws SQLException	1.0
setQueryTimeout(int seconds) throws SQLException	1.0

Table 4-28 java.sql.Statement

javax.sql.ConnectionPoolDataSource

Methods	Version
getPooledConnection()	3.0
getPooledConnection(String user, String password) throws SQLException	3.0

Table 4-29 javax.sql.ConnectionPoolDataSource

javax.sql.Datesource

Methods	Version
getConnection(java.lang.String user,,java.lang.String password) throws java.sql.SQLException;	3.0
getConnection() throws java.sql.SQLException;	1.0
getLogWriter() throws SQLException	3.0
getLoginTimeout() throws SQLException	3.0
setLoginTimeout(int seconds) throws SQLException	3.0

Table 4-30 javax.sql.DataSource

javax.sql.Pooledconnection

Methods	Version
addConnectionEventListener(ConnectionEventListener listener)	1.0
addStatementEventListener(StatementEventListener listener)	4.0
close() throws SQLException	3.0
getConnection() throws SQLException	3.0
removeConnectionEventListener(ConnectionEventListener listener)	1.0
removeStatementEventListener(StatementEventListener listener)	4.0

Table 4-31 javax.sql.PooledConnection

javax.sql.XAConnection

Methods	Version
addConnectionEventListener(ConnectionEventListener listener)	1.0
addStatementEventListener(StatementEventListener listener)	4.0
close() throws SQLException	3.0
getConnection() throws SQLException	3.0
removeConnectionEventListener(ConnectionEventListener listener)	1.0
removeStatementEventListener(StatementEventListener listener)	4.0
getXAResource() throws java.sql.SQLException;	3.0

Table 4-32 javax.sql.XAConnection

javax.sql.XADataSource

Methods	Version

getXAConnection(String user, String password) throws SQLException	3.0

Table 4-33 javax. sql.XADataSource

javax.transaction.xa.XAResource

Methods	Version
commit(Xid xid, boolean arg1) throws XAException	1.0
end(Xid xid, int arg1) throws XAException	1.0
forget(Xid xid) throws XAException	1.0
getTransactionTimeout() throws XAException	1.0
isSameRM(javax.transaction.xa.XAResource arg0) throws XAException	1.0
prepare(Xid xid) throws XAException	1.0
recover(int arg0) throws XAException	1.0
rollback(Xid xid) throws XAException	1.0
setTransactionTimeout(int arg0) throws XAException	1.0
start(Xid xid, int arg1) throws XAException	1.0

Table 4-34 javax.transaction.xa. XAResource

javax.transaction.xa.Xid

Methods	Version
getBranchQualifier() ;	1.0
getFormatId() ;	1.0
getGlobalTransactionId() ;	1.0

Table 4-35 javax.transaction.xa.Xid

4.3 System Function Implemented

Following table shows the system function that DBMaker JDBC implemented.

Numeric Function	ABS,ACOS,ASIN,ATAN,ATAN2,CEILING,COS,COT, DEGREES,EXP,FLOOR,LOG,LOG10,MOD,PI,POWER, RADIANS,RAND,ROUND,SIGN,SIN,SQRT,TAN
String Function	ASCII,CHAR,CONCAT,DIFFERENCE,INSERT,LCASE,LEFT,LENGTH,LOCATE,LTRIM,REPEAT,REPLACE,RIGHT,RTRIM,SPACE,SUBSTRING,UCASE
System	DBNAME, IFNULL, USERNAME
TimeData Function	CURDATE,CURTIME,DAYNAME,DAYOFMONTH,DAYOFWEEK,DAYOFYEAR,HOUR,MINUTE,MONTH,MONTHNAME,NOW,QUARTER,SECOND,TIMESTAMPADD, TIMESTAMPDIFF,WEEK,YEAR
Example	Examples are given to clarify descriptions, and commonly include text, as it will appear on the screen. Other forms of this convention include Prototype and Syntax.
CommandLine	This format is commonly used to show the JDBC code or the other code that the procedure needed

Table 4-36 Implemented Function

5 Frequently Asked Questions

Q1: Why I face the error “ClassNotFoundException: DBMaker.sql.JdbcOdbcDriver SQLException: No suitable driver” when execute the Java program that access database by DBMaker JDBC Driver?

A1: Please see that the dmjdbc20.jar or dmjdbc30.jar lies in the classpath when you run the java program. See how to run the samples for details.

Q2: Why I face the error “Exception in thread "main" java.lang.UnsatisfiedLinkError: no dmjdbcxx in java.library.path” When I execute the Java program that access database by DBMaker JDBC Driver?

A2: The user needs the DBMaker native Driver to run the Java program to access database by DBMaker JDBC Driver. Please see that the native driver dmjdbcxx.dll (so) lies in the java.library.path java library pass variable. Please see How to run the samples for details.

Q3: Why I face the error “SQLException: fail to establish a connection (8007), [DBMaker] cannot find the specified database section in the configuration file”, When I execute the Java program that access database by DBMaker JDBC Driver?

A3: Please see that the database you plan to access was well configured in the dmconfig.ini file. About the dmconfig.ini, please see the DBA Manual (dba.chm) for details.

Q4: Does DBMaker JDBC Driver support transaction and is there any suggestion for the use of transaction?

A4: DBMaker does support the transaction in JDBC. As any DBMS with transaction, the program should be designed carefully and the long transaction should be avoid by splitting into small ones. If the transaction is used in multithread environment, please be aware of the problem of deadlock when you design the transaction you plan to run.

Q5: Why I got UnsupportedException when I called the method PreparedStatement.setDecimal()?

A5: At present, DBMaker does not implement **java.sql.PreparedStatement**.setDecimal in the interface **java.sql.PreparedStatement**. The workaround is to use **java.sql.PreparedStatement.setString**() for the column with the datatype Decimal. For the details of the implementation of interface **java.sql.PreparedStatement** by DBMaker JDBC Driver, please see the chapter 4.2.8 of this document. All of the interfaces of JDBC standard implemented by DBMaker JDBC Driver have been listed in the prievous chapter 4.

Q6: How can I use DBMaker JDBC Driver in DBMaker bundle version?

A6: Java AP need to load dmjdbcxx.jar at first, this jar will load dmjdbcxx.dll and dmapixx.dll. To find these .dll paths, please add the following path setting to your Java library path when launching Java VM:Java –
Djava.library.path=C:\YourBundlePath

6 Appendix Sample codes

The Appendix gives the completely codes of the samples that used in the previous chapter.

6.1 Sample1 Usage of Clob

This sample can be run in the same way as for the section of RUN SAMPLE PROGRAM.

Please refer to the section RUN SAMPLE PROGRAM for details.

File: UsageOfClob.java

```
import java.io.*;
import java.sql.*;
public class UsageOfClob {
        public static void main(String[] args)
{
Connection conn = null;
int buff len = -1;
          try {
           // Get the connection to Database server
           Class.forName("DBMaker.sql.JdbcOdbcDriver").newInstance();
           conn =
DriverManager.getConnection("jdbc:DBMaker:support","SYSADM","");
```

```
// to get a statemant object
           Statement stmt = conn.createStatement();
           // The table jdbc clob demo will be used later, so we create
it first
           stmt.execute("CREATE TABLE jdbc clob demo(id INT, content
LONG VARCHAR)");
           // Insert Clob data from demo.txt
            PreparedStatement pstmt = conn.prepareStatement(
"INSERT INTO jdbc clob demo(id,content) VALUES(?,?)");
          // Open the file demo.txt which contains the clob data
            FileInputStream fis = null;
   // To insert  three tuples into database
            for (int i = 0;i < 3;++i)
{
               fis = new FileInputStream("demo.txt");
         // Get the available length of the InputStream
         buff len = fis.available();
         // Set the Stream as Clob data for the PreparedStatement
         pstmt.setInt(1,i+1);
         pstmt.setBinaryStream(2,fis,buff len);
         // Execute the INSERT command
         pstmt.execute();
         fis.close();
             }

           // After the INSERT command is executed, close the pstmt we
prepared before
           pstmt.close();
           // Retrieve Clob data from jdbc clob demo
           ResultSet rs = stmt.executeQuery("SELECT ID,content FROM
jdbc clob demo");

           // There is only one Clob tuples here, so we just iterate
the ResultSet once
int id =0, len=0 ;
Clob clob =null;
```

```
String str = null;
byte[] buffer=null ;
InputStream astream=null;
          while (rs.next())
       {
          id = rs.getInt(1);
          // Retrive the Clob data into the instance of Clob
          clob = rs.getClob(2);
          // Get the Clob data as characters encoded with ASCII
          astream = clob.getAsciiStream();
          // Allocate buffer for the stream
          len = astream.available();
buffer= new byte[len+1];
          // Read the characters into the buffer
          astream.read(buffer);
          astream.close();
          // Construct an instance of String by the content of buffer
with ASCII decoder
          str = new String(buffer,0,len,"ascii");
          // Here we just print the Clob characters to the screen
          System.out.println(str);
             }
          rs.close();
         }catch(Exception e){
           e.printStackTrace();
          }finally {
           // Close the connection if it was opened
           if( conn != null)
              try{ conn.close() ;} catch(Exception e){}
          }
          }
}
```

6.2 Sample2 Usage of Blob

This sample can be run in the same way as for the section of RUN SAMPLE PROGRAM.

Please refer to the section RUN SAMPLE PROGRAM for details.

File: UsageOfBlob.java.

```
mport java.io.*;
import java.sql.*;


public class UsageOfBlob {
    public static void main(String[] args) {
        Connection conn = null;
int buff len = -1;
        try {
            // Get the connection to Database server
            Class.forName("DBMaker.sql.JdbcOdbcDriver").newInstance();
            conn =
DriverManager.getConnection("jdbc:DBMaker:support","SYSADM","");


            Statement stmt = conn.createStatement();
            // The table jdbc blob demo will be used later, so we create
it first
            stmt.execute("CREATE TABLE jdbc blob demo(id INT, photo LONG
                                VARBINARY)");


            // Insert Blob data from logo.gif
            PreparedStatement pstmt = conn.prepareStatement(
                                                                "INSERT INTO
jdbc blob demo(id,photo) VALUES(?,?)");
            FileInputStream fis = null;
            For (int i = 0; i < 3; ++i)
{
            // Open the file logo.gif which contains the blob data
            fis = new FileInputStream("logo.gif");
// Get the available length of the InputStream
```

```
         buff len = fis.available();
         // Set the Stream as Blob data for the PreparedStatement
pstmt.setInt(1,i+1);
 pstmt.setBinaryStream(2,fis,buff len);
             // Execute the INSERT command
         pstmt.execute();
fis.close();
}
            // After the INSERT command is executed, close the pstmt we
prepared before
            pstmt.close();

// Retrieve Blob data from jdbc blob test and write to logo-copy.gif
ResultSet rs = stmt.executeQuery("SELECT ID,PHOTO FROM jdbc blob demo");
Blob blob = null;
int id = -1;
InputStream bs=n ull ;
FileOutputStream fos = null;
               byte[] buffer = null;
 while (rs.next())
{
             id = rs.getInt(1);
             // Retrive the Blob data into the instance of Blob
             blob = rs.getBlob(2);
 // Get the Blob data as the binary stream
             bs = blob.getBinaryStream();
                     // Allocate buffer the stream
             buff len = bs.available();
buffer = new byte[buff len+1];
             // Here we just output the Blob to the file system
             fos = new FileOutputStream("logo-copy.gif");
             bs.read(buffer);
             fos.write(buffer);
             // Close the input and output stream
```

```
            bs.close();
            fos.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        finally{
            // Close the connection if it was opened
            if( conn != null)
                try{ conn.close() ;} catch(Exception e){}
        }
    }
}
```

6.3 Sample3 Usage of FO

Also, this sample can be run in the same way as for the section of RUN SAMPLE PROGRAM.

Please refer to the section RUN SAMPLE PROGRAM for details.

FILE: UsageOfFo.java

```
import java.io.*;
import java.sql.*;
public class UsageOfFo {
    public static void main(String[] args) {
        Connection conn = null;
int buff len = -1;
        try {
            // Get the connection to Database server
            Class.forName("DBMaker.sql.JdbcOdbcDriver").newInstance();
            conn =
DriverManager.getConnection("jdbc:DBMaker:support","SYSADM","");
            // create a statement object
            Statement stmt = conn.createStatement();
```

```
            // The table jdbc blob demo will be used later, so we create
it first
            stmt.execute("CREATE TABLE jdbc fo demo(ID INTEGER,PHOTO
FILE)");
            PreparedStatement pstmt = conn.prepareStatement("INSERT INTO
jdbc fo demo                          VALUES(?,?)");
            FileInputStream fis = null;
            for (int i = 0;i < 3; ++i)
{
            pstmt.setInt(1,i+1);
            // Open the file logo.gif which contains the blob data
            fis = new FileInputStream("logo.gif");
            // Get the available length of the InputStream
            buff len = fis.available();
            // Set the Stream as Blob data for the PreparedStatement
            pstmt.setBinaryStream(2,fis,(int)buff len);
            // Execute the INSERT command
            pstmt.execute();
            fis.close();
            }
            // After the INSERT command is executed, close the pstmt we
prepared before
            pstmt.close();
            // Retrieve the ID,filename and filelen from jdbc blob test
and print to the screen
            // Meanwhile, we get the blob data and write it to logo-
copy-fo.gif
            ResultSet rs = stmt.executeQuery("SELECT ID,filename(PHOTO)
AS NAME," +
                        "filelen(PHOTO) AS LENGTH,PHOTO" + " FROM
jdbc fo demo ");
            Blob blob = null;
            InputStream is = null;
            FileOutputStream fos = null;
            int id=0 , len=0;
            String name=null ;
byte[] buffer = null ;
```

```
            while (rs.next())
{
                // Get the ID, filename and filelen columns
                id = rs.getInt(1);
                name = rs.getString(2);
                len = rs.getInt(3);
                // Print the ID, filename and filelen columns to the
screen

System.out.println("ID="+id+"\nName="+name+"\nLength="+len);
                // Get the File Object as a Blob data
                blob = rs.getBlob(4);
                // Get the Blob data as binary stream
                is = blob.getBinaryStream();
                // Allocate buffer for the binary stream
                buff len = is.available();
                buffer = new byte[buff len+1];
                // Create the outer file as the destination of copy
                fos = new FileOutputStream("logo-copy-fo.gif");
                // Output the binary stream to the outer file
                is.read(buffer);
                fos.write(buffer);
                // Close the input and output stream
                is.close();
                fos.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }finally{
            // Close the connection if it was opened
            if( conn != null)
                 try{ conn.close() ;} catch(Exception e){}
        }
    }
}
```

6.4 Sample4 Usage demo of Stored Procedure

The following is the step by step work to run this sample program:

1. Create table schema in dmSQL like:

```
dmSQL> create table SYSADM.JDBC SP DEMO (
       ID  INTEGER not null  ,
       NAME  VARCHAR(12) default null ,
       BIRTHDAY  DATE default null )
       in DEFTABLESPACE  lock mode page  fillfactor 100 ;
       alter table SYSADM.JDBC SP DEMO primary key ( ID) in
DEFTABLESPACE;
```

2. Create the procedure insert_or_update in dmSQL like:

```
dmSQL> create procedure from 'insert_or_update.ec';
```

Please note that copy the file 'insert_or_update.ec' into the directory DBMaker/4.x/bin first.

3. Create the procedure query_data in dmSQL like:

```
dmSQL> create procedure from 'query_data.ec';
```

Please note that copy the file 'queryupdate.ec' into the directory DBMaker/4.x/bin first.

4. Compile the program UsageOfBlob.java by javac and run the program by java with the classpath containing dmjdbc20.jar.

Please note that the database DBNAME should be started before run the program. For the details of running sample program, please refer to the section RUN SAMPLE PROGRAM.

File: insert_or_update.ec

File: insert_or_update.ec

```
/*****************************************
 * This stored procedure insert or update the record
```

```
 * in the table jdbc sp demo.
 * If the record id existed in
 * jdbc sp demo, then this stored procedure will
 * update the name and birthday column of the record with
 * the same id as the old one.
 * If the record id does not exist in jdbc sp demo,
 * then this stored procedure just insert this tuple.
 *
 * Parameters :
 *     INPUT  id        The identifier column of the tuple
 *     INPUT  name      The name column of the tuple
 *     INPUT  birthday  The birthday column of the tuple
 *     OUTPUT inInsert  Return to the caller is the tuple updated
 *                      the old one or is inserted
 ***************************************** /
EXEC SQL CREATE PROCEDURE insert or update(INT id INPUT,VARCHAR(12) name
INPUT,
        DATE birthday INPUT,INT isInsert OUTPUT) RETURNS STATUS;
{
        EXEC SQL BEGIN DECLARE SECTION;
        int  isExist = -1;
        EXEC SQL END DECLARE SECTION;

        EXEC SQL BEGIN CODE SECTION;
        EXEC SQL SELECT COUNT(*) INTO :isExist FROM jdbc sp demo WHERE
ID = :id;
        if(isExist > 0) {
                EXEC SQL UPDATE jdbc sp demo SET NAME = :name,BIRTHDAY
= :birthday
                        WHERE ID = :id;
                isInsert = 0;
        }
        else if(isExist == 0) {
                EXEC SQL INSERT INTO jdbc sp demo(ID, NAME, BIRTHDAY)
                        VALUES(:id,:name,:birthday);
```

```
                isInsert = 1;
                }
        else {
                isInsert = -1;
                }
        EXEC SQL RETURNS STATUS SQLCODE;
        EXEC SQL END CODE SECTION;
}
```

File: query_data.ec

File: query_data.ec

```
/*****************************************
 * This procedure returns the ResultSet of all the tuples in
 * the table jdbc sp demo that the birthday is after the 'in birthday'
 *
 * Paramerters :
 *    INPUT     in birthday        All the tuples with the column of
birthday
 *                                  larger than in birthday will be
selected
 *****************************************/
exec sql create procedure query data(DATE in birthday input) returns int
id,varchar(12) name,date birthday;
{
        exec sql begin code section;
        exec sql RETURNS SELECT ID ,NAME ,BIRTHDAY
                 FROM jdbc sp demo WHERE BIRTHDAY > :in birthday
INTO :id,:name,:birthday ;
        exec sql end code section;
}
```

File: UsageOfStoredProcedure.java

File:UsageofStoredProcedure.java

```
import java.sql.*;
```

```
public class UsageOfStoredProcedure {
public static void main(String[] args)
{
        Connection conn = null;
int isInsert = -1;  // isInsert  means ➔  0 : Update 1 : Insert
        try {
            // Get the connection to Database server
            Class.forName("DBMaker.sql.JdbcOdbcDriver").newInstance();
            conn =
DriverManager.getConnection("jdbc:DBMaker:newmis","SYSADM","");
            // create a statement object
Statement stmt = conn.createStatement();
// Prepare for call StoredProcedure insert or update
CallableStatement pc = conn.prepareCall("{CALL
insert or update(?,?,?,?)}");

        // Insert one tuple into table
        pc.setInt(1,1);    // Set the ID as 1
        pc.setString(2,"John Nash");    // Set the name as 'John Nash'
Date d = new Date(1928-1900,4-1,13);
        pc.setDate(3,d);
     // To register the forth parameters as OUTPUT variable
pc.registerOutParameter(4,Types.INTEGER);
// To call the procedure insert or update
pc.execute();
        // Retrive the OUTPUT variable to see if the INSERT command is
executed
isInsert = pc.getInt(4);
if (isInsert == 1)
{
         System.out.println("Insert Tuple with ID = 1");
         }else if (isInsert == 0)
{
          System.out.println("Uncorrect output variable returned");
```

```
          }


          // Update the tuple with the ID = 1
          pc.setInt(1,1);    // Set the ID as 1
          pc.setString(2,"Nash");   // To modify the name from 'John
Nash' to 'Nash'
          pc.setDate(3,d);
          // To register the forth parameters as OUTPUT variable
          pc.registerOutParameter(4,Types.INTEGER);
          // To call the procedure insert or update
          pc.execute();
          // Retrive the OUTPUT variable to see if the UPDATE command is
executed
          isInsert = pc.getInt(4);
          if (isInsert == 1)
{
             System.out.println("Uncorrect output variable returned");
              }else if (isInsert == 0)
{
             System.out.println("Update Tuple with ID = 1");


             // Here, we close the CallableStatement we preprepared
before
             pc.close();


        // Now we call the procedure query data
        // To prepare a new CallableStatement for query
        pc = conn.prepareCall("{CALL query data(?)}");
        // Set the date , all the tuples with the birthday column larger
than this date will be selected.
        pc.setDate(1,new Date(1901-1900,0,0));


        // To execute the Query
         pc.execute();
        // To get the result set, if it exists
        ResultSet rs = pc.getResultSet();
```

```
        // Print out the result , if the ResultSet is not null
        if  (rs != null)
        {
        System.out.println("ID    NAME    BIRTHDAY");
        //This is just the same as the common iteration of ResultSet
int id =0;
         String name =null ;
         Date birth = null ;;
          while (rs.next())
{
           id = rs.getInt(1);
        name = rs.getString(2);
                birth = rs.getDate(3);
                System.out.println(id+"    "+name+"    "+birth);
                }
}
// At last, we close the CallableStatement we prepared before
pc.close();
        }catch(Exception e) {
             e.printStackTrace();
        }finally {
            // Close the connection if it was opened
            if( conn != null)
                 try{ conn.close() ;} catch(Exception e){}
        }
    }
}
```