DBMaster

JDBC程序员参考手册

SYSCOM

SYSCOM Computer Engineering Co./Corporate Headquarters

B1, 2-7F No. 115 Emei Street, Wanhua District,
Taipei City 108, Taiwan (R.O.C.)

www.dbmaker.com

www.dbmaker.com.tw/service

Document No.645049-237521/DBM54CN-M03312017-JDBC

发行日期: 2017-03-31

对于本手册中没有体现的关于产品最新功能的描述，请在安装完成SYSCOM DBMaster 软件后阅读 README.TXT 文件。

注册商标
SYSCOM，SYSCOM 图标和 DBMaster 是SYSCOM 公司的注册商标。
Microsoft, MS-DOS, Windows 和 Windows NT 是 Microsoft 公司的注册商标。
UNIX 是 The Open Group 的注册商标。
ANSI 是美国国家标准化组织的注册商标。

手册中提到的其他产品名称或许是它们各自持有者的注册商标，仅仅是为提供此信息。
SQL 是行业语言，并不为任何公司或任何组织所有。

注意事项
本手册中有关软件的描述，均以该软件所提供的使用许可为基础。

对于授权许可的详细信息，请与您的经销商联系。关于计算机产品的特殊用途的市场性与适用性，经销商不会给予任何说明和保证。因外界因素如地震、过热、过冷和潮湿而引起产品的任何损坏以及由于使用不正确的电压和不兼容的软硬件而引起的损失和损坏，经销商概不负责。

虽然该手册的内容已经过仔细核对，但错误再所难免。若手册再有改动，不另行通知。还请见谅。

目录

1 简介

欢迎使用JDBC程序员手册，该手册将为用户如何使用DBMaster JDBC提供一些介绍和示例。众所周知，JDBC API标准提供了访问数据库和以java程序操纵数据库的解决方案。它的很多优势都过于ODBC，并且被DBMS供应商和JAVA程序员广泛采用，同时它也成为java访问数据库的标准。目前，DBMaster JDBC驱动程序支持JDBC2.0和JDBC 3.0标准以适应更多更有用的平台。

1.1 其它相关文件

除了本手册外，DBMaster还提供了一整套的数据库管理系统（DBMS）手册，用户可以根据特定需要参考以下手册。

- 有关DBMaster性能和功能的介绍，请参考*DBMaster指南*。
- 有关设计、管理和维护DBMaster数据库的信息，请参考*数据库管理员手册*。
- 有关如何管理DBMaster的信息，请参考*服务器管理工具用户手册*。
- 有关DBMaster的配置信息，请参考*配置管理工具用户手册*。
- 有关DBMaster功能的信息，请参考*数据库管理工具用户手册*。
- 有关JDBC API的信息，请参考*JDBC程序员参考手册*。
- 有关dmSQL命令行工具的使用方法，请参考*dmSQL使用手册*。
- 有关DBMaster SQL语言的语法和使用的相关信息，请参考*SQL命令与函数参考手册*。
- 有关嵌入式ESQL/C语言的语法和使用，请参考*ESQL/C程序员参考手册*。
- 有关DBMaster的错误信息及警告信息，请参考*错误信息参考手册*。
- 有关DCI COBOL接口的详细信息，请参考*DBMasterDCI用户手册*。

1.2 技术支持

在软件试用期间，Syscom Computer Engineering Co.(“Syscom”)将为您提供30天的免费email支持和电话支持。当软件注册后，我们还会再为您提供30天的免费技术支持。如此一来，您就可以获得60天的免费支持。不仅如此，在您购买软件后，Syscom对任何问题都会以email的方式为您提供技术支持。

您除了可以获得免费的技术支持外，还可以以20%的零售价购买其它产品。要想获得更多的详细资料和价格信息，请与sales@dbmaker.com保持联系。

您可以通过任何一种方式(普通信件、电话或email)与Syscom技术支持保持联系，请登录至：www.casemaker.com/support以获取详细信息。在与Syscom技术支持联系之前，请先查询当前数据库的常见问题解答。

无论您以何种方式与Syscom的技术支持联系时，请务必写上以下有效信息：

- 产品名称和版本号
- 注册号
- 注册的用户名和地址
- 供应商/发行者的地址
- 操作平台和计算机系统配置
- 错误发生前执行的动作
- 如果可以，请提供错误信息和编号
- 其它一些相关信息

1.3 文档协定

为方便用户的阅读和使用，本手册使用了一种标准的排版约定：注释、程序、示例和命令行都用缩进排版的方式进行了特别的设置。

协定	说明
斜体字	斜体字表示必须输入的信息占位符，例如用户名和表名。此字符可用实际的名称来替换。有时，文档也会使用斜体字来介绍新的关键字，强调着重点。
黑体字	黑体字表示文件名、数据库名、表名称、字段名、用户名和其它数据库对象。它也用于强调程序执行步骤中的菜单命令。
关键字	文字段落中， SQL语言使用的关键字都是以大写字母出现的。
小符号	文档中出现的小写字符表示键盘上的按键，两个键名之间的加号（+）表示在按住第一个键不放的同时，再按第二个键。两个键名之间的逗号（，）表示释放第一个键以后，再按第二个键。
注意	包含一些重要的信息。
➲ **程序**	表示后面跟随的是程序的执行步骤或连续的项目。很多任务都是通过这种方式描述，给用户提供一个逻辑顺序步骤得以效仿。
➲ **示例**	例子用来阐明描述，通常包括屏幕上出现的文本，用户也可以将这些例子输入到计算机中，通过屏幕看到运行结果。当然，示例还包括一些原型和语法。
命令行	包括文本，这些命令都可以输入计算机中，显示在屏幕上。通常用于显示 SQL 命令的输入输出或 dmconfig.ini 中的内容。

表 1-1 文档协定

2 入门

本章将为用户介绍如何使用JDBC驱动来进行快速启动，并演示如何安装、注册并连接DBMaster JDBC驱动，为了使用户更容易理解，下面将举一些示例进行说明。

2.1 JDBC Type II 驱动入门

获得并安装DBMaster JDBC驱动

DBMaster JDBC驱动直接与DBMaster产品绑定，%DBMASTER%_HOME\bin目录下的dmjdbc20.jar, dmjdbc20xa.jar和dmjdbc30.jar文件都是DBMaster JDBC驱动的二进制文件。JDBC API 2.0的接口由dmjdbc20.jar和dmjdbc20xa.jar实现，dmjdbc20xa.jar包括由dmjdbc20.jar实现的接口和XA支持的接口。JDBC API 3.0的接口由dmjdbc30.jar实现。此外，由dmjdbc20.jar和dmjdbc20xa.jar实现的接口也可由dmjdbc30.jar实现。

注册DBMaster JDBC驱动

DBMaster JDBC驱动支持Type II JDBC驱动，一些二进制文件是必须的。Windows平台下，安装DBMaster进程中，复制本地文件dmjdbcXX.dll到目录%winnt%\system32；Unix/Linux/Solaris平台下，用户可在shell原始文件中设置环境变量“LD_LIBRARY_PATH”，将文件libdmjdbcXX.so包含在目录/DBMASTER_HOME/lib/so下，语法如下所示：

对于sh (.profile)、 bash (.bashrc)，用户可添加如下命令到原始文件中：

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

对于csh/tcsh (.cshrc)，用户可添加如下命令到原始文件中：

```
setenv LD_LIBRARY_PATH /DBMASTER_HOME/lib/so
```

此外，如果要使用DBMake JDBC驱动编译并运行Java程序，用户需在包含JDBC驱动jar文件的目录下设置CLASSPATH环境。

在Java程序中，若通过DriverManager获得连接，则用户需援引如下代码：

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

使用DriverManager获取连接前，用户需为应用程序注册DBMaster JDBC驱动。

使用DBMaster JDBC驱动连接数据库

DBMaster JDBC驱动支持用户通过DriverManager和DataSource连接数据库。这里首先介绍DriverManager的使用方法。

最简单的例子如下：

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

如代码所示，用户“SYSADM”连接到数据库“dbname”，登陆密码是“abc123”。

此外，用户可通过以下属性连接数据库：

```
Properties connect_property = new Properties();
connect_property.setProperty("user","SYSADM");
connect_property..setProperty("password", "abc123");
Connection conn = DriverManager.getConnection("jdbc:DBMaster:dbname",
connect_property);
```

如果用户想通过由服务器地址和端口号指定的URL连接数据库，例如："jdbc:DBMaster://SERVER_ADDRESS:PORT_NUMBER/DATABASE_NAME"，可使用如下代码：

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

使用这种连接方法，用户无需在dmconfig.ini中指定数据库。

另一方面，DBMaster JDBC实现了通过接口DataSource和XADataSource连接数据库。详情请参考*通过J2EE应用服务器使用DBMaster手册*。

2.2 JDBC Type III 驱动入门

获得并安装DBMaster JDBC驱动

Type III JDBC驱动是纯java JDBC驱动，它在主调程序和数据库之间使用一个中间层。该中间层（应用服务器）直接或间接地将JDBCcalls转换成厂商特性的数据库协议。使用Type II JDBC，用户至少需安装DBMaster客户端；使用Type III JDBC，用户可通过Type III JDBC的JAR文件dmjdbct3c.jar连接数据库应用程序代码。Type III JDBC专为不想安装DBMaster客户端的用户设计。

注册DBMaster JDBC驱动

DBMaster JDBC驱动支持Type III JDBC驱动。如果要通过Type III JDBC连接DBMaster服务器，用户仅需要一个Type III JDBC的JAR文件。

在Java程序中，若通过DriverManager获得连接，则用户需援引如下代码：

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

使用DriverManager获取连接前，用户需为应用程序注册DBMaster JDBC驱动。

使用DBMaster JDBC驱动连接数据库

Type III服务器是一个单独的中间层服务器，它不与任何指定数据库绑定，且生命周期不同于数据库服务器。Type III服务器在DBMaster服务器安装过程自动安装，用户可为二者选择不同的安装路径。数据库服务器和Type III服务器应同时启动。数据库设置应写在**dmconfig.ini**文件的适当位置，以便于Type III服务器读取。Type III服务器不设置会话超时，用户可使用数据库服务器处理会话超时。仅拥有相同lcode的数据库才能在同一时间通过Type III服务器互连。例如，db1（lcode=1），db2

（lcode=2），如果db1已连接Type III服务器，则此时若db2也要连接Type III服务器，用户需在不同端口为这两个拥有不同lcode的数据库分别开启一个jetty服务器，否则将返回错误。

Type III服务器通过hessian实现，在jetty 7服务器下配置。Type III客户端和服务器都需要jre 1.6环境。

在Windows平台下启动Type III服务器，用户需先点击**DBMaster 5.4**，接着在其扩展菜单中选择**JDBCT3Server Start**。在Linux/Unix平台下启动Type III服务器，用户可使用如下命令：

```
type ~dbmaster/5.4/bin/t3svr
```

Type III服务器的默认端口号是8083，用户可更改该端口号。

在Windows平台下更改默认端口号，用户需首先打开**JDBCT3Server Start属性**页的**快捷方式**页签，在该页签中找到**目标**字段，该字段取值如下所示：

```
C:\DBMaster\5.4\jre\bin\java.exe -Djava.library.path="C:\DBMaster\5.4\bin" -
Djetty.port=8083 -DSTOP.PORT=8082 -DSTOP.KEY=stopme -jar start.jar
```

最后，通过编辑“Djetty.port=8083”更改默认端口号8083。

在Linux/Unix平台下更改默认端口号，用户需编辑如下**t3svr**脚本：

```
#!/bin/sh
#
PROG=`basename $0`
case ${PROG} in
  t3svr_stop) ARGS="--stop" ;;
  *) ARGS="--daemon" ;;
esac

cd /home/dbmaster/5.4/jetty
exec /home/dbmatster/5.4/jre/bin/java \
          -Djava.library.path=/home/dbmaster/5.4/lib/so \
          -Djetty.port=8083 \
          -DSTOP.PORT=8082 \
          -DSTOP.KEY=stopme \
          -jar /home/dbmaster/5.4/jetty/start.jar ${ARGS}
```

注意 *dll/so 库路径（dmjdbc53.dll/libdmjdbc53.so）已存在于快捷方式/脚本中（-Djava.library.path=xxxx），因此用户无需再添加 dbmaster/5.4/bin 到库路径。*

Type III服务器提供相关Servlet，该Servlet是dmjdbc (Type II)的包裹类，并遵循HTTP协议。Type III JDBC jar文件dmjdbct3c.jar支持的类和方法与dmjdbc （Type II）相似，提供JDBC接口，其主要差异是连接Type III中间服务器时用户需指定主机名和端口号。

使用dmjdbc (Type II)连接数据库：

- 编码前，用户需将dmjdbc30.jar放置于classpath下。指定java classpath有两种方法：一是设置环境变量CLASSPATH；二是设置java命令的参数–classpath。使用Type II JDBC时，用户需添加libdmjdbc53.so、dmjdbc53.dll至PATH或LD_LIBRARY_PATH
- 编码时，下载类DBMaster.sql.JdbcOdbcDriver
- URL地址类似jdbc:DBMaster:<dbname>

编码示例：

```
import java.sql.*;
public class SimpleTest
  {
   public static void main(String[] args)
     {
      try
        {
         Class.forName("dbmaster.sql.JdbcOdbcDriver");
         Connection conn =
DriverManager.getConnection("jdbc:dbmaster: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();
      }
    }
}
```

使用Type III JDBC连接数据库：

- 编码前，用户需将type 3 jdbc jar文件dmjdbct3c.jar放置于classpath下，且无需添加jdbc so/dll至PATH或LD_LIBRAY_PATH
- 编码时，下载类DBMaster.jdbc.ws.client.Driver
- URL地址类似jdbc:DBMaster:type3://<type3server ip>:<port number>/<dbname>

编码示例：

```
import java.sql.*;
public class SimpleTest
  {
   public static void main(String[] args)
    {
     try
       {
        Class.forName("dbmaster.jdbc.ws.client.Driver");
        Connection conn =
DriverManager.getConnection("jdbc:dbmaster:type3://127.0.0.1:8083/JDBCT
EST", "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 样例程序

该章提供JDBC使用的详细样例，并向用户介绍在DBMaster中运行样例的步骤。

3.1 运行 DBMaster 中的样例程序

使用Type II驱动

在以下描述中，DBMASTER_HOME用于表示DBMaster的安装目录。例如，Unix平台下，DBMASTER_HOME用于表示DBMaster 5.4类似/home/DBMaster/5.4的目录；Windows平台下，DBMASTER_HOME用于表示DBMaster 5.4类似C:\DBMaster\5.4的目录。

在目录%DBMASTER_HOME%\samples\JDBC下，存在三个单独样例程序，分别命名为ex_Resultset.java, ex_ExecuteParam.java和ex_Resultset_update.java，这些文件用于演示DBMaster JDBC驱动的常规用法。ex_Resultset.java用于插入数据和检索数据；ex_ExecuteParams.java用于演示更新或插入参数数值；ex_Resultset_update.java用于演示更新或插入ResultSet数据，并介绍JDBC 2.0之后的JDBC标准引入的新API。.

若用户要编译并运行JDBC样例程序，首先要启动安装DBMaster进程时创建的数据库DBSAMPLE5。同时，用户需安装JDK1.2或更高版本，并确定JRE适用于该操作系统。

参考章节2.1注册DBMaster JDBC驱动部分正确注册驱动后，用户可在java和javac命令中任选一个编译运行样例程序。例如，在UNIX和Windows系统下，在环境变量CLASSPATH中注册dmjdbcXX.jar后，用户便可使用如下命令编译运行名为ex_ExecuteParam.java的样例程序：

```
# javac ex_ExecuteParam.java
# java ex_ExecuteParam
```

如果用户要在没有在CLASSPATH中注册dmjdbcXX.jar的情况下编译运行名为ex_ExecuteParam.java的样例程序，需通过javac和java的参数-classpath指定CLASSPATH。

在UNIX系统下，语法如下所示：

```
#javac -classpath /DBMASTER_HOME/lib/java/dmjdbcxx.jar:./ ex_ExecuteParam.java
#java -classpath /DBMASTER_HOME/lib/java/dmjdbcxx.jar:./ ex_ExecuteParam
```

在Windows系统下，语法如下所示：

```
#javac -classpath \DBMASTER_HOME\bin\dmjdbcxx.jar;.\ ex_ExecuteParam.java
#java -classpath \DBMASTER_HOME\bin\dmjdbcxx.jar;.\ ex_ExecuteParam
```

使用Type III驱动

样例Employee.java存在于目录%DBMASTER_HOME%\samples\JDBC下，用于演示如何使用Type III JDBC检索数据库数据。

如果用户要编译并运行JDBC样例程序，首先要启动安装DBMaster进程时创建的数据库DBSAMPLE5。同时，用户需安装JDK1.6或更高版本，并确定JRE 1.6适用于该操作系统。

参考章节2.1注册DBMaster JDBC驱动部分正确注册驱动后，用户可在java和javac命令中任选一个编译运行样例程序。例如，在UNIX和Windows系统下，在环境变量CLASSPATH中注册dmjdbcXX.jar后，用户便可使用如下命令编译运行名为Employee.java的样例程序：

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

如果用户要在没有在CLASSPATH中注册dmjdbct3c.jar的情况下编译运行名为Employee.java的样例程序，需通过javac和java的参数-classpath指定CLASSPATH。

在UNIX系统下，语法如下所示：

```
#javac -classpath /DBMASTER_HOME/lib/java/dmjdbct3c.jar:./ Employee.java
#java -classpath /DBMASTER_HOME/lib/java/dmjdbct3c.jar:./ Employee
```

在Windows系统下，语法如下所示：

```
#javac -classpath \DBMASTER_HOME\bin\dmjdbct3c.jar;.\ Employee.java
#java -classpath \DBMASTER_HOME\bin\dmjdbct3c.jar;.\ Employee
```

3.2 一般的 JDBC 程序例

如何使用JDBC执行SQL命令？

下面介绍使用**Statement**和**PreparedStatement**操作数据库对象的一般步骤，例如，CREATE TABLE、INSERT TUPLES等。

Conn对象已连接至数据库服务器。如何使用DBMaster JDBC连接数据库，请参考章节2。

为连接创建**Statement**：

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

使用**Statement**执行**DDL**：

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

使用**Statement**执行**DML**：

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

准备包含主机变量的**PreparedStatement**：

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

请注意调用**PreparedStatement**的方法很耗费时间，因此对于包含相同主机变量的相同**DML**，用户需尽可能地重复使用**PreparedStatement**。

用户可利用包含主机变量的**PreparedStatement**为**employee**插入以下信息：**employee**的**ID**是4，**name**是“Mickey Mouse”，**salary**是“30000.00”，**hired_date**是“1950-01-01”，如下所示：

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

```
...
```

如果用户想要使用之前准备的同一个**PreparedStatement**为三个employee（“John”、“Mary”、“Eric”）添加信息，可使用如下代码：

```
// 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() ;
}
```

此时，用户通过一次“prepare”和三次“execution”插入了三组数据。

如何使用JDBC检索数据库数据

为演示方便，假设表模式如下：

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

首先，已连接到DBMaster服务器的连接创建一个Statement对象：

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

接下来，用户可通过Statement对象的方法executeQuery执行查询命令并获得结果集：

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

接下来，用户可以显示通过查询语句获取的结果集：

```
While(rs.next())
{
System.out.println("ID: "+rs.getInt(1) +
        "NAME: "+rs.getString(2) +
                "SALARY: "+ rs.getFloat(3) +
                "HIRED_DATE: " + rs.getDate(4) );
}
```

若要使用游标更新对象**ResultSet**，用户首先需创建**ResultSet**类型ResultSet.CONCUR_UPDATABLE的Statement对象：

```
Statement stmt = con.createStatement (ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

ResultSet类型的详细信息请参考JDBC规范。

此时，用户可使用之前的方法执行查询命令并获得**ResultSet**：

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

不同之处在于，此时用户可使用游标更新**ResultSet**：

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

```
    rs.updateRow();
}
```

Dmjdbc性能有两处主要改进，该改进可帮助用户在检索数据时提高内存分配效率，并可在检查是否需要追踪或字段映射时避免重复检查：

- 早期的dmjdbc版本，在JdbcOdbcResultSet中gerXXX每次都会在java中分配缓存，并将其传递给C，C代码将会为每个缓存指定检索值，该缓存无法重复利用。改进后的dmjdbc版本可避免频繁分配/释放内存，从而使得已分配的缓存得到重复使用。
- JDBC的调试机制是调用DriverManager.setLogStream或DirverManager.setLogWriter。早期的JdbcOdbcXXX类中，JDBC实现方法的第一行会检查日志流和日志作者，并决定是否打印调试信息。该检测操作对性能有显著影响。改进后，在所有JdbcOdbcXXX类的原型JdbcOdbc.java中保留标志（一个变量）needTrace，此时所有的JdbcOdbcXXX将会继承这个变量并在创建对象时指定变量值。因此，在每个方法中，将仅检查标志needTrace，而不调用DriverManager.setLogStream或DirverManager.setLogWriter。

dmjdbc版本改进后，needTrace的值只能在创建对象时设置，且设置后无法更改。因此，如果用户使用如下代码：

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

则仅有rs的追踪信息输出至日志中，因为日志作者是在stmt创建后设置的。

如何使用JDBC操纵BLOB数据？

DBMaster用户可使用有效函数操纵Blob、Clob、FO等大数据。DBMaster JDBC驱动为Java程序中的Blob和Clob数据实现接口**java.sql.Clob**和**java.sql.Blob**。

如何使用JDBC操纵Java程序中Blob和Clob数据将会在此进行介绍。

通过DBMaster JDBC驱动实现的API详情请参考章节4.2。下面给出Blob、Clob和FO数据的JDBC代码实例。

CLOB数据的使用样例

该样例用于演示如何使用DBMaster JDBC驱动操纵Clob数据。

在该例中，我们首先将文本文件demo.txt的内容存储在数据库中，然后将其检索出来并将其输出到控制台。

首先，创建表**jdbc_clob_demo**，设定该表某一字段数据类型为long varchar，用于存储Clob数据。

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

接下来，使用如下代码将文本文件demo.txt的内容存储到该表中：

```
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();
```

现在文本文件demo.txt的内容已被存储到数据库中。接下来，检索出该内容并将其输出到控制台。实际上，用户还可执行更多操作，而不仅仅是将检索到的内容输出。以下代码用于检索该文本文件内容并将其输出：

```
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);
}
```

该样例的完整代码请参考附录 样例1。

同时，用户可在新行中插入使用**ResultSet**检索到的Clob数据，代码如下所示：

```
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();
}
```

BLOB数据的使用样例

该样例用于演示如何使用DBMaster JDBC驱动操纵Blob数据。

在该例中，我们首先将图片demo.gif存储在数据库中，然后将其检索出来并将其保存为gif文件。

首先，创建表**jdbc_clob_demo**，设定该表某一字段数据类型为long varbinary，用于存储图片文件。

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

接下来，读出图片文件demo.gif，将其存储到**jdbc_blob_demo**中：

```
PreparedStatement pstmt = c.prepareStatemnt( “INSERT INTO
jdbc_blob_demo(id,photo) VALUES(?,?)”);
FileInputStream fis = new FileInputStream(“demo.gif”);
Int buff_len = fis.available();
pstmt.setInt(1,1);
pstmt.setBinaryStream(2,fis,buff_len);
pstmt.execute();
pstmt.close();
fis.close();
```

现在图片demo.gif已作为二进制数据存储到数据库中。接下来，从jdbc_blob_demo中检索出该二进制数据并将其保存为gif文件dup-demo.gif。用户使用如下代码将该图片输出到控制台：

```
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 ();
}
```

该样例的完整代码请参考附录 样例2。

同时，用户可在新行中插入使用**ResultSet**检索到的Clob数据，代码如下所示：

```
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();
 }
```

FO数据的使用样例

DBMaster支持FILE数据类型，用户可使用FILE类型的字段以文件的方式存储二进制数据，该文件不是Blob文件（*.bb）。Java编程时，如果在dmconfig.ini中设置关键字DB_FoTyp值为1，则FILE类型数据的使用方法与LONG VARBINARY或LONG VARCHAR的使用方法相同，有关关键字的详细信息请参考DBA手册(dba.chm)。在附录中，样例3完整演示了如何操作FILE数据。

如何调用存储过程？

用户可使用EC语法编辑存储过程，并在dmSQL中创建与该EC程序文件对应的存储过程。详情请参考dba.chm中存储过程章节。

用户可使用如下语法调用存储过程：

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

下面的样例演示如何使用DBMaster JDBC驱动调用存储过程

样例的表模式如下所示：

```
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;
```

用户使用存储过程UPDATE_OR_INSERT向表中添加记录时，如果该记录不存在于表中，则在表JDBC_SP_DEMO中添加该记录；如果该记录

已存在于表中，则在表JDBC_SP_DEMO中更新该记录。请参考附录 样例4的文件update_or_insert.ec。

要使用JDBC调用存储过程，用户首先需要在数据库中创建该存储过程。DBMaster用户可使用命令CREATE PROCEDURE FROM …从包含EC程序的文件中创建存储过程。详情请参考*数据库管理员手册存储过程章节*。

接下来介绍调用存储过程的一般步骤。为便于说明，假设存储过程名为demo_sp，含有两个参数：string类型的INPUT参数和integer类型的OUTPUT参数。同时，该存储过程demo_sp将返回查询结果集，该结果集包含两个字段，分别为integer类型和string类型。

使用java.sql.Connection.prepareCall()准备CallableStatement by java.sql.Connection.prepareCall()。

用户应使用JDBC转义语法预编译该存储过程，代码如下所示：

```
// 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(?,?)}");
```

1. 若存储过程包含输出变量，使用 **CallableStatement.registerOutputParameter()**注册该输出变量。

DBMaster JDBC驱动支持存储过程包含输出参数，用户可使用如下代码注册输出参数：

```
// 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);
```

请注意用户可通过索引（1级开始）和字段名注册输出参数。

2. 使用CallableStatement.setXXX()设置输入参数。

类似于PreparedStatement，用户也可使用索引（1级开始）和字段名设置输入参数。设置输入参数代码如下：

```
// 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');
```

3. 使用cstmt.executeQuery()执行并检索查询结果。

如果存储过程执行查询动作并返回结果集，则cstmt.executeQuery()返回类class java.sql.ResultSet的实例可访问该结果集。

```
//  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) ;
}
```

请注意用户可通过基于索引和名称的字段检索输出参数。

如何获取数据库/结果集/参数的元数据？

JDBC规范的元数据在高级编程中起着重要作用。

DatabaseMetaData接口包含数据库服务器规范的所有方法，例如，是都支持事务、存储过程、外部连接等，以及目标数据库服务器的某些信息。

用户可使用如下代码获取**DatabaseMetaData**：

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

Conn是已连接到DBMaster服务器的连接对象。

例如，如果要验证DBMS是否支持事务，用户可通过接口**DatabaseMetaData**的方法supportsTransactions()获取元数据。

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

同样，如果要验证DBMS是否支持存储过程，用户可通过接口**DatabaseMetaData**的方法supportsStoredProcedure()获取元数据。.

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

ResultSetMetaData接口包含与查询所返回结果集的信息相关的所有方法，例如，字段个数、每个字段的名称和类型等。是都支持事务、存储过程、外部连接等，以及目标数据库服务器的某些信息。这些信息可用于编写通用程序。

用户可使用如下代码获取**ResultSetMetaData**：

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

Rs是**ResultSet**对象，该对象由查询语句返回。例如，如果用户对查询检索出的ResultSet一无所知，用户可通过**ResultSetMetaData**重申**ResultSet**，代码如下：

```
/* 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接口包含与含有主机变量的语句的参数信息相关的所有方法，例如，JDBC类型、变量精度。

用户可使用如下代码获取**ParameterMetaData**：

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

Patmt由一些连接对象准备，是**PreparedStatement**对象。

用户也可获得**CallableStatement**执行的存储过程的参数的元数据。

使用**ParameterMetaData**检索**CallableStatement**参数信息元数据的代码如下：

```
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 DBMaster JDBC 驱动参考

本章介绍DBMaster JDBC驱动的参考信息，包括DBMaster JDBC驱动支持的数据类型、实现的API及系统函数。

4.1 支持的数据类型

DBMaster JDBC驱动支持的数据类型如表4-1所示。

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

表 4-1 支持的数据类型

4.2 JDBC 实现的 API

JDBC Type II驱动

目前，DBMaster JDBC Type II 驱动支持符合JDBC2.0和JDBC3.0标准的大部分接口及其方法，该接口及方法描述如下：

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

表 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	3.0

java.sql.SQLException;	
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	1.0

java.sql.SQLException;	
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

表 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

表 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

表 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	1.0

procedureNamePattern,java.lang.String columnNamePattern) throws java.sql.SQLException;	
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 java.sql.SQLException;	2.0
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 java.sql.SQLException;	1.0
supportsAlterTableWithDropColumn() throws java.sql.SQLException;	1.0
supportsBatchUpdates() throws java.sql.SQLException;	2.0
supportsCatalogsInDataManipulation() throws java.sql.SQLException;	1.0
supportsCatalogsInIndexDefinitions() throws	1.0

java.sql.SQLException;	
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	1.0

java.sql.SQLException;	
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	1.0

java.sql.SQLException;	
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 java.sql.SQLException;	1.0
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 java.sql.SQLException;	2.0
usesLocalFilePerTable() throws java.sql.SQLException;	1.0
usesLocalFiles() throws java.sql.SQLException;	1.0

表 4-6 java.sql.DatabaseMetaData

JAVA.SQL.DRIVER

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

表 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

表 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 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 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	1.0

targetJdbcType,int scale) throws java.sql.SQLException;	
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

表 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 java.sql.SQLException;	1.0
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	1.0

java.sql.SQLException;	
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	2.0

columnName,java.io.InputStream x,int length) throws java.sql.SQLException;	
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

表 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	1.0

java.sql.SQLException;	
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 java.sql.SQLException;	1.0
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

表 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

表 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

表 4-13 javax.sql.ConnectionPoolDataSource

JAVAX.SQL.DATESOURCE

Methods	Version
getConnection(java.lang.String user,java.lang.String	1.0

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

表 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

表 4-15 javax.sql.PooledConnection

JAVAX.SQL.XACONNECTION

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

表 4-16 javax.sql.XAConnection

JAVAX.TRANSACTION.XA.XID

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

getGlobalTransactionId();	1.0

表 4-17 javax.transaction.xa.Xid

JDBC Type III 驱动

目前，DBMaster JDBC Type III 驱动支持符合JDBC2.0、JDBC3.0和JDBC4.0标准的大部分接口及其方法，该接口及方法描述如下。

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

表 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	4.0

SQLException;	
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
wasNull() throws SQLException;	1.0

表 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

表 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

表 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,	1.0

String foreignSchema,String foreignTable) throws SQLException;	
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	1.0

columnNamePattern)throws SQLException;	
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

表 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

表 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

表 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	4.0

SQLException;	
setBinaryStream(int parameterIndex, InputStream x, int length)throws SQLException;	1.0
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 SQLException;	2.0
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

表 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	1.0

SQLException;	
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	1.0

SQLException;	
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)	2.0

throws SQLException;	
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
isFirst() throws SQLException;	2.0
isLast() throws SQLException;	2.0
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	4.0

x) throws SQLException;	
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,	4.0

InputStream x, long length) throws SQLException;	
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	4.0

x)throws SQLException;	
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	2.0

SQLException;	
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	4.0

nClob) throws SQLException;	
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	2.0

SQLException;	
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
wasNull() throws SQLException;	1.0

表 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

表 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

表 4-28 java.sql.Statement

JAVAX.SQL.CONNECTIONPOOLDATASOURCE

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

表 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

表 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

表 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

表 4-32 javax.sql.XAConnection

JAVAX.SQL.XADATASOURCE

Methods	Version
getXAConnection(String user, String password) throws	3.0

SQLException;

表 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

表 4-34 javax.transaction.xa. XAResource

JAVAX.TRANSACTION.XA.XID

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

表 4-35 javax.transaction.xa.Xid

4.3 实现的系统函数

DBMaster JDBC实现的系统函数如下表所示。

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

表 4-36 Implemented Function

5 常见问题

Q1: Java程序已通过DBMaster JDBC驱动连接到数据库，执行时为什么产生错误“ClassNotFoundException: DBMaster.sql.JdbcOdbcDriver SQLException：无合适驱动”？

A1: 运行java程序时，请确定dmjdbc20.jar或dmjdbc30.jar位于classpath下。详情请参考如何运行样例。

Q2: Java程序已通过DBMaster JDBC驱动连接到数据库，执行时为什么产生错误“线程”main”异常，java.lang.UnsatisfiedLinkError：java.library.path下无dmjdbcxx”？

A2: 要运行Java程序并使用DBMaster将其连接到数据库上，用户需要使用DBMaster本地驱动。请确定dmjdbc20.jar或dmjdbc30.jar位于classpath下。详情请参考如何运行样例。

Q3: Java程序已通过DBMaster JDBC驱动连接到数据库，执行时为什么产生错误“SQLException：建立连接失败（8007），[DBMaster]无法在配置文件中找到指定数据库”？

A3: 请确定已在dmconfig.ini文件中已配置好要连接的数据库。有关dmconfig.ini文件，详情请参考DBA手册(dba.chm)。

Q4: DBMaster JDBC驱动是否支持事务？关于事务的使用有何建议？

A4: DBMaster JDBC驱动支持事务。和任何支持事务的DBMS一样，用户需仔细设计程序且不要将程序分为几个小程序。如果在多线程唤醒下使用事务，用户在设计将要执行的事务时需注意死锁问题。

Q5: 调用方法PreparedStatement.setDecimal()时，为什么返回UnsupportedException ？

A5: 目前，DBMaster 无法实现在接口 **java.sql.PreparedStatement** 中实现 **java.sql.PreparedStatement.setDecimal**。解决方法是数据类型为 Decimal 的字段使用 **java.sql.PreparedStatement.setString()**。使用 DBMaster JDBC 驱动实现接口 **java.sql.PreparedStatement**，详情请参考本文档章节 4.2.8 。 DBMaster JDBC 驱动实现的 JDBC 标准接口在第四章中列出。

Q6: 如何在DBMaster bundle版中使用DBMaster JDBC驱动？

A6: 首先，Java AP需要下载dmjdbcxx.jar，该jar文件将会下载dmjdbcxx.dll和dmapixx.dll。为了找到这些.dll路径，请在启动Java VM时添加如下路径到Java库路径中：Java –Djava.library.path=C:\YourBundlePath。

6 附录样例编码

该附录给出之前章节中所使用样例的完整代码。

6.1 样例1 Clob的使用

该样例的运行方法和章节运行样例程序中的运行方法相同。

详情请参考章节运行样例程序。

文件：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("DBMaster.sql.JdbcOdbcDriver").newInstance();
           conn =
DriverManager.getConnection("jdbc:DBMaster: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 样例2 Blob的使用

该样例的运行方法和章节运行样例程序中的运行方法相同。

详情请参考章节运行样例程序。

文件：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("DBMaster.sql.JdbcOdbcDriver").newInstance();
            conn =
DriverManager.getConnection("jdbc:DBMaster: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 样例3 FO的使用

该样例的运行方法和章节运行样例程序中的运行方法相同。

详情请参考章节运行样例程序。

文件：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("DBMaster.sql.JdbcOdbcDriver").newInstance();
            conn =
DriverManager.getConnection("jdbc:DBMaster: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 样例 4 存储过程的使用演示

运行样例程序步骤如下所示：

1. 使用如下代码在dmSQL中创建表模式：

```
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. 使用如下代码在dmSQL中创建存储过程insert_or_update：

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

请注意首先用户需复制文件insert_or_update.ec到目录DBMaster/4.x/bin。

3. 使用如下代码在dmSQL中创建程序query_data：

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

请注意首先用户需复制文件insert_or_update.ec到目录DBMaster/4.x/bin。

4. 使用javac编译程序UsageOfBlob.java，并在包含dmjdbc20.jar的路径下使用java运行该程序。

请注意用户需在运行程序前启动数据库。有关运行样例程序，详情请参考章节运行样例程序。

文件：insert_or_update.ec

文件：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;
}
```

文件：query_data.ec

文件：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;
}
```

文件：UsageOfStoredProcedure.java

文件：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("DBMaster.sql.JdbcOdbcDriver").newInstance();
            conn =
DriverManager.getConnection("jdbc:DBMaster: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){}
        }
    }
}
```