DBMaster

JDBCプログラマー参照編

SYSCOM

CASEMaker Inc./Corporate Headquarters

1680 Civic Center Drive
Santa Clara, CA 95050, U.S.A.

Contact Information:
CASEMaker US Division
E-mail : info@casemaker.com
Europe Division
E-mail : casemaker.europe@casemaker.com
Asia Division
E-mail : casemaker.asia@casemaker.com(Taiwan)
E-mail : info@casemaker.co.jp(Japan)

www.casemaker.com
www.casemaker.com/support

Document No. 645049-236340/DBM54J-M09302015-JDBC
発行日:2015-09-30

本文には記されていない新しい機能についての説明は、CASEMakerのDBMasterをインストールしてからREADME.TXTを読んでください。

登録商標
CASEMaker、CASEMakerのロゴは、CASEMaker社の商標または登録商標です。
DBMasterは、Syscom Computer Engineering社の商標または登録商標です。
Microsoft、MS-DOS、Windows、Windows NTは、Microsoft社の商標または登録商標です。
UNIXは、The Open Groupの商標または登録商標です。
ANSIは、American National Standards Institute, Incの商標または登録商標です。

ここで使用されているその他の製品名は、その所有者の商標または登録商標で、情報として記述しているだけです。SQLは、工業用語であって、いかなる企業、企業集団、組織、組織集団の所有物でもありません。

注意事項
本書で記述されるソフトウェアは、ソフトウェアと共に提供される使用許諾書に基づきます。

保証については、ご利用の販売店にお問い合わせ下さい。販売店は、特定用途への本コンピュータ製品の商品性や適合性について、代表または保証しません。販売店は、突然の衝撃、過度の熱、冷気、湿度等の外的な要因による本コンピュータ製品へ生じたいかなる損害に対しても責任を負いません。不正な電圧や不適合なハードウェアやソフトウェアによってもたらされた損失や損害も同様です。

本書の記載情報は、その内容について十分精査していますが、その誤りについて責任を負うものではありません。本書は、事前の通知無く変更することがあります。

目次

1 はじめに.....1-1

1.1 その他のマニュアル..... 1-1

1.2 字体の規則 1-2

2 JDBC ドライバーを使用 2-1

2.1 JDBC Type II ドライバーを使用する 2-1

DBMaster JDBC ドライバーをインストール.....2-1
DBMaster JDBC ドライバーをレジスター.....2-1
DBMaster JDBC ドライバーによってデータベースに接続 2-2

2.2 JDBC Type III ドライバーを使用する 2-3

JDBC ドライバーをインストール.....2-3
DBMaster JDBC ドライバーをレジスター.....2-3
DBMaster JDBC Driver ドライバーによってデータベースに接続.....2-4

3 サンプルプログラム 3-1

3.1 DBMaster にサンプルプログラムを実行 3-1

TYPE II ドライバー 3-1
TYPE III ドライバー 3-2
3.2 一般 JDBC プログラムの例示 3-3
JDBC を通じて SQL コマンドを実行する方法.....3-3
JDBC を通じてデータベースからデータを取得する方法3-5
JDBC を通じて BLOB データをどう処理するか? 3-7
ストアドプロシージャをどう呼び出すか? 3-12
データベース結果セット/パラメータにメタデータをどう取得するか？ 3-14
4 DBMaster JDBC ドライバーの参考 4-1
4.1 サポートするデータタイプ..... 4-1
4.2 実行される JDBC API 4-2
JDBC TYPE II ドライバー 4-2
JDBC TYPE III ドライバー 4-34
4.3 実行されるシステム関数..... 4-65
5 Q&A..... 5-1
6 付録のサンプルコード..... 6-1
6.1 サンプル 1 Clob の使用方法..... 6-1
6.2 サンプル 2 Blob の使用方法..... 6-3
6.3 サンプル 3 FO の使用方法..... 6-6
6.4 サンプル 4 ストアドプロシージャの使用方法デモ ... 6-8
ファイル： INSERT_OR_UPDATE.EC 6-9
ファイル： QUERY_DATA.EC.....6-11
ファイル：USAGEOFSTOREDPROCEDURE.JAVA 6-11

1 はじめに

JDBC プログラマー参照編にようこそ。この参照編はユーザーがDBMaster JDBCを使用できるため、用法とサンプルを提供します。 As all known, JDBC API標準はソリューションを提供してデータベースをアクセスします、またJavaプログラムにデータベースからのデータを操作します。これはODBC より優勢が強く、DBMS ベンダーとJavaプログラマから使用されます。同時、これはJava でデータベースをアクセスする標準になります。今では、DBMaster JDBC ドライバーはJDBC2.0 と JDBC 3.0 標準のインターフェースをサポートします。

1.1 その他のマニュアル

DBMasterには、本マニュアル以外にも多くのユーザーガイドや参照編があります。特定のテーマについての詳細は、以下のマニュアルをご覧下さい。

- DBMasterの能力と機能性についての概要は、「*DBMaster入門編*」をご覧下さい。
- DBMasterの管理についての詳細は、「*JServer Managerユーザーガイド*」をご覧下さい。
- DBMasterの環境設定についての詳細は、「*JConfiguration Tool参照編*」をご覧下さい。
- DBMasterの機能についての詳細は、「*JDBA Toolユーザーガイド*」をご覧下さい。

- DBMasterで使用しているdmSQLのインターフェースについての詳細は、「*dmSQLユーザーガイド*」をご覧下さい。
- DBMasterで採用しているSQL言語についての詳細は、「*SQL文と関数参照編*」をご覧下さい。
- ESQLプログラムについての詳細は、「*ESQL/Cプログラマー参照編*」をご覧下さい。
- ODBCプログラムについての詳細は、「*ODBCプログラマー参照編*」をご覧下さい。
- エラーと警告メッセージについての詳細は、「*エラー・メッセージ参照編*」をご覧下さい。
- ネイティブDCI APIについての詳細は、「*DCI ユーザーガイド*」をご覧下さい。
- SQLストアドプロシージャ言語の詳細については、「*SQLストアドプロシージャ参照編*」を参照して下さい。
- ロックについての詳細は、「*ロックユーザーガイド*」をご覧下さい。
- DBMasterバンドルバージョンについての詳細は、「*DBMasterバンドルバージョンユーザーガイド*」をご覧下さい。
- データベースの性能についての詳細は、「*性能チューニングユーザーガイド*」をご覧下さい。

1.2 字体の規則

本書は、標準の字体規則を使用しているので、簡単かつ明確に読むことができます。

斜体	斜体は、ユーザー名や表名のような特定の情報を表します。斜体の文字そのものを入力せず、実際に使用する名前をそこに置き換えてください。斜体は、新しく登場した用語や文字を強調する場合にも使用します。
太字	太字は、ファイル名、データベース名、表名、カラム名、関数名やその他同様なケースに使用します。操作の手順においてメニューのコマンドを強調する場合にも、使用します。
キーワード	文中で使用する SQL 言語のキーワードは、すべて英大文字で表現します。
小さい英大文字	小さい英大文字は、キーボードのキーを示します。２つのキー間のプラス記号 (+) は、最初のキーを押したまま次のキーを押すことを示します。キーの間のコンマ(,)は、最初のキーを放してから次のキーを押すことを示します。
注	重要な情報を意味します。
プロシージャ	一連の手順や連続的な事項を表します。ほとんどの作業は、この書式で解説されます。ユーザーが行う論理的な処理の順序です。
例	解説をよりわかりやすくするために与えられる例です。一般的に画面に表示されるテキストと共に表示されます。
コマンドライン	画面に表示されるテキストを意味します。この書式は、一般的に dmSQL コマンドや dmconfig.ini ファイルの内容の入/出力を表示します。

2 JDBC ドライバーを使用

この章では、JDBC ドライバーを快速に使用できる方法を説明します。JDBC ドライバーのインストール、レジスター、接続についての方法を例示されます。ユーザーは幾つのサンプルを通じて容易に理解できます。

2.1 JDBC Type II ドライバーを使用する

DBMaster JDBCドライバーをインストール

DBMaster JDBC ドライバーはDBMaster 製品に置いています。ディレクトリ%DBMASTER_HOME\binのdmjdbc20.jar、 dmjdbc20xa.jar とdmjdbc30.jarはJDBC ドライバーのバイナリファイルです。JDBC API 2.0 インターフェースはdmjdbc20.jar と dmjdbc20xa.jarファイルに実行されます、dmjdbc20xa.jarがdmjdbc20.jarに実行するインターフェースを除いて、XA支持インターフェースを含みます。In dmjdbc30.jar, these interfaces introduced in JDBC API 3.0 インターフェースはdmjdbc30.jar 、dmjdbc20.jar と dmjdbc20xa.jar を含みます。

DBMaster JDBCドライバーをレジスター

DBMaster JDBCドライバは Type II JDBCドライバをサポートします、だから幾つのネイティブバイナリファイルの使用が必要になります。Windows では、ネイティブファイル “dmjdbcXX.dll”はDBMasterをインストール中にディレクトリ%winnt\system32にコピーされます、Unix/Linux/Solaris では、ユーザーは環境変数“LD_LIBRARY_PATH”を通じて、ファイル

libdmjdbcXX.soを含むパス“"/DBMASTER_HOME/lib/so"を指定しなければなりません。ユーザーは自分のシエルイニシャルファイルを以下のように設定できます：

sh (.profile), bash (.bashrc)に対して、イニシャルファイルに以下のコマンドを追加

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

csh/tcsh (.cshrc) に対して、イニシャルファイルに以下のコマンドを追加

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

そのほか、DBMaster JDBCドライバを使用してJavaプログラムをコンパイル、実行する場合、.jarファイルを含むディレクトリにCLASSPATH環境を設定するべきです。

Javaプログラムに、接続はDriverManager から得ると、接続はDriverManager から得る前、以下のコマンドを呼び出してDBMaster JDBCドライバをレジスターします。

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

DBMaster JDBCドライバーによってデータベースに接続

DBMaster JDBCドライバーはDriverManagerとDataSourceによるデータベースと接続できます。 まず、DriverManagerの使用方法を説明します。

簡単のケースは：

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

コードの表示のように、'dbname'は“SYSADM” で、パスワードは“abc123” で、'dbname'は“SYSADM” というデータベースと接続します。

同様に、Propertiesを通じて接続できます：

```
Properties connect property = new Properties();
connect property.setProperty("user","SYSADM");
connect_property..setProperty("password", "abc123");
```

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

ユーザーは
"jdbc:DBMaster://SERVER_ADDRESS:PORT_NUMBER/DATABASE_NAME"
ようなURLを通じてデータベースと接続すると、以下のようにしてください：

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

この方法を使用すると、dmconfig.iniにデータベースセクションを指定する必要がありません。.

一方、DBMaster JDBCはインターフェースDataSource と XADataSource がデータベースと接続することが実現します。詳細についてはUsing DBMaster with J2EE Application Server Manualを参考してください。

2.2 JDBC Type III ドライバーを使用する

JDBCドライバーをインストール

Type III JDBC ドライバーは純粋的な java JDBC ドライバーで、呼び出すプログラムとデータベースの間に中間層です。この中間層（アプリケーションサーバー）は直接または間接にJDBCcalls をベンダー特定のデータベースプロトコルに転換します。Type II JDBCを使用するため、ユーザーは少なくともDBMaster クライアントをインストールします。Type III JDBCを使用するため、ユーザーはType III JDBC のjarファイルdmjdbct3c.jarを通じてデータベースアプリケーションコードと接続できます。 Type III JDBC は DBMaster クライアントをインストールしたくないユーザーに準備されます。

DBMaster JDBCドライバーをレジスター

DBMaster JDBCドライバーはDBMasterサーバーまで接続するため Type III JDBCドライバーをサポートします。Type III JDBC ドライバーを使用して、 Type III JDBC jar ファイルだけ必要です。

Java プログラムに、この接続はDriverManager から得ると、呼び出してください：

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

DriverManagerは接続を得る前、DBMaster JDBCドライバをレジスターします。

DBMaster JDBC Driverドライバーによってデータベースに接続

Type III サーバーは独立的な中間層サーバーですが、任意の特別のデータベースとバインドされていません。ライフサイクルはデータベースと異なります。Type IIIサーバーはDBMasterサーバーと同時にインストールされます、データベースサーバーとType III サーバーは同じサイトにある必要がありません。ユーザーはデータベースサーバーとType III サーバーを起動すべきです。Type IIIサーバーが読み取れるためデータベースの設定はdmconfig.iniに正確に書き入れるはずです。Type III サーバーはセッションタイムを設定するため使用されません、でも、ユーザーはデータベースサーバーを使用してセッションタイムが設定できます。同じのlcodeをもつデータベースだけType IIIサーバーによってアクセスできます。例えば、db1 (lcode=1) と db2 (lcode=2)。Type III サーバーはdb1までもう接続した後、ユーザーはType III サーバーをdb2と接続するとエラーを返します。可能な解決方法はlcodeデータベースをサーブするため異なるポートに2 jetty サーバーを起動すべきです。Type III サーバーはヘッセンによる実現して、jetty 7 サーバーに配置されます。jre 1.6環境はType III クライアントとサーバーに必要です。

WindowsでType IIIサーバーを起動すると、ユーザーはメニューで **DBMaster 5.4**をクリックして**JDBCT3Server Start** クリックします。
Linux/UnixでType III サーバーを起動するため以下のコマンドを使用してください：

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

Type IIIサーバーのディフォルトポート番号は 8083ですが、ユーザーはこれを変更することができます。

Windowsで**shotcut** tab in **JDBCT3Server Start**プロパティページの**ショートカット**タブをクリックして**ターゲット**フィールドが見えます。以下のは**ターゲット**フィールドの値です：

```
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
```

最後、ユーザーは“Djetty.port=8083”をエディットして簡単に8083を変更することができます。

Linux/Unixで以下の**t3svr**スクリプトをエディットします：

```
#!/bin/sh
#
PROG=`basename $0`
case ${PROG} in
  t3svr stop) ARGS="--stop" ;;
  *) ARGS="--daemon" ;;
esac

cd /home/dbmaster/5.4/jetty
exec /home/dbmaster/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 *このdll/soライブラリパス (dmjdbc54.dll/libdmjdbc54.so)はショートカット/スクリプトに設定されました (-Djava.library.path=xxxx)。 dbmaster/5.4/binをライブラリパスに追加する必要がありません。*

手動でType III サーバーを起動します：

```
cd [DBMaster installed directory]/jetty
```

簡単に起動：

```
java -jar start.jar
```

ユーザーはjettyサーバーを終了するため、ストップポートとキーを指定して適当なサインを送ります：

```
java -DSTOP.PORT=8084 -DSTOP.KEY=stopme -jar start.jar
```

ディフォルトのjetty ポートは 8080, これを以下のように指定します：

```
java -Djetty.port=8083 -jar start.jar
```

STOP.PORT とKEYを指定した、ユーザーは以下のようにしてjettyを終了できます。

```
java -DSTOP.PORT=8084 -DSTOP.KEY=stopme -jar start.jar --stop
```

DBMaster は二つのショートカットが作成します。一つはjettyを起動すること、もう一つはjettyを終了することです：

```
java -DSTOP.PORT=8084 -DSTOP. KEY=stoptype3server -Djetty.port=8088 -
jar start.jar

java -DSTOP.PORT=8084 -DSTOP. KEY= stoptype3server -jar start.jar --
stop
```

Type III サーバーは相応的なservletを提供します、servletはdmjdbc (Type II)のカプセル化のクラスで、HTTPプロトコルに従います。このクラスとメソッドはType III JDBC jarファイル **dmjdbct3c.jar** からサポートされてJDBC インタフェースを提供します。サポートされたクラス/メソッドはdmjdbc (type II)と似ています。主な異なる所はユーザーはType III 中間サーバーを接続するとホスト名とポート番号を指定しなければならないことです。

dmjdbc (Type II)を使って接続をします：

- コードする前、classpathにdmjdbc30.jarが必要です。. Two ways to specify java classpathを指定するため二つの方法があります、一つは環境変数CLASSPATHを設定すること、もう一つはjavaコマンド引数–classpathを設定することです。Type 2 jdbc はPATH 或いはLD_LIBRARY_PATHにlibdmjdbc54.so/dmjdbc54.dllが必要です
- コードにクラスDBMaster.sql.JdbcOdbcDriverをロードします
- URLはjdbc:DBMaster:<dbname>のようにします

コードの例示：

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

Type III JDBCを使って接続をします：

- コードをする前、ユーザーはtype 3 jdbc jar ファイル**dmjdbct3c.jar** クラスパスに必要がありますが、PATH或いは LD_LIBRAY_PATHにjdbc so/dllがある必要がありません
- コードにクラスDBMaster.jdbc.ws.client.Driverをロードします

- URLはjdbc:DBMaster:type3://<type3server ip>:<port number>/<dbname>のようです

コードの例示：

```
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/JDBCTE
ST", "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 サンプルプログラム

この章では、JDBCの使用についての詳細なサンプルを提供します。またDBMasterにこれを実行するステップを説明します。.

3.1 DBMaster にサンプルプログラムを実行

Type IIドライバー

以下の説明に、In the following description, we use DBMASTER_HOME を使用してDBMasterのインストールディレクトリを表示します。例えば、DBMASTER_HOME はUnix でのDBMaster 5.4のディレクトリを ‘/home/DBMaster/5.4’ように表示します或いはでのDBMaster 5.4のディレクトリを‘C:\DBMaster\5.4’ように.表示します。

ex_Resultset.java、 ex_ExecuteParam.java とex_Resultset_update.javaという三つのJDBCサンプルプログラムはディレクトリ %DBMASTER_HOME%\samples\JDBCに置いています。それらのファイルはDBMaster JDBC Driverの一般使用を示します。ex_Resultset.javaにデータを挿入と取り戻す用法を示します。ex_ExecuteParams.javaにパラメータがあるデータを更新/挿入することを説明します。最後、結果セットがあるデータを更新/挿入する用法、ex_Resultset_update.javaにJDBC2.0 以上のJDBC 標準をもつAPI の用法を説明します。

To compile and run the sample programs of JDBCのサンプルプログラムをコンパイル/実行するため、まず、DBMaster にインストールされたデータベースDBSAMPLE5を起動する必要があります。同時に、JDK1.2或いはJDK1.2

以上のバージョンをOSにインストールされるべきで、QSにJREを正常に実行することを確保してください。

*2.1章のDBMaster JDBC ドライバーをレジスター*を参考してドライバーをレジスタします。java コマンドとjavacコマンドを通じてユーザーはサンプルプログラムをコンパイル、実行することができます。

ユーザーはdmjdbcXX.jarをCLASSPATH 環境変数にレジスタする場合、ex_ExecuteParam.javaサンプルプログラムは以下のコマンドによってコンパイル\実行されます（UNIX とWindows）：

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

dmjdbcXX.jar はCLASSPATHまでレジスタしないと、javac と javaの引数 -classpathを使用してCLASSPATHを指定することを通じてex_ExecuteParam.javaサンプルプログラムをコンパイル、実行します。

以下のはUNIXでの構文です：

```
#javac -classpath /DBMASTER HOME/lib/java/dmjdbcxx.jar:./
ex ExecuteParam.java
#java -classpath /DBMASTER HOME/lib/java/dmjdbcxx.jar:./
ex_ExecuteParam
```

以下のはWindowsでの構文です：

```
#javac -classpath \DBMASTER HOME\bin\dmjdbcxx.jar;.\
ex ExecuteParam.java
#java -classpath \DBMASTER_HOME\bin\dmjdbcxx.jar;.\ ex_ExecuteParam
```

Type IIIドライバー

ディレクトリ%DBMASTER_HOME%\samples\JDBC にEmployee.javaがあります。このサンプルはデータベースからデータを取り戻すためtype 3 jdbcの使用方法を示します。

まず、JDBCのサンプルプログラムをコンパイルして実行するため、DBMaster にインストールされたデータベースDBSAMPLE5を起動する必要があります。同時に、JDK1.6或いはJDK1.6以上のバージョンをOSにイン

ストールされるべきで、QSにJREを正常に実行することを確保してください。

*2.2章のDBMaster JDBCドライバーをレジスター*を参考してドライバーをレジスタします。java コマンドとjavacコマンドを通じてユーザーはサンプルプログラムをコンパイル、実行することができます。

ユーザーはdmjdbct3c.jarをCLASSPATH 環境変数にレジスタする場合、Employee.javaサンプルプログラムは以下のコマンドによってコンパイル\実行されます（UNIX とWindows）：

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

dmjdbct3c.jar はCLASSPATHまでレジスタしないと、javac と javaの引数 -classpathを使用してCLASSPATHを指定することを通じてEmployee.javaサンプルプログラムをコンパイル、実行します。

以下のはUNIXでの構文です：

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

以下のはWindowsでの構文です：

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

3.2 一般 JDBC プログラムの例示

JDBCを通じてSQLコマンドを実行する方法

ここにはStatement と PreparedStatementを使用してデータベースオブジェクトを操作する一般のステップ（CREATE TABLE、INSERT TUPLESなど）を説明します。

このconnオブジェクトはデータベースと接続します。DBMaster JDBCによるデータベースと接続する方法は第2章を参考してください。

接続に一つのStatementを作成します：

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

このStatementを通じてDLLを実行します：

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

このStatementを通じてDMLを実行します：

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

ホスト変数によって **PreparedStatement** を準備します：

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

方法PreparedStatementを呼び出すことは時間をかかりますので、同じホスト変数を使って同じDMLにPreparedStatement を再利用します。

従業員の情報を挿入する場合：従業員のid は４で、従業員の名前は "Mickey Mouse"で、給料は“30000.00”、そしてhired_date は“1950-01-01”です。それらのデータは、ホスト変数を持つ**PreparedStatement**によって挿入されます：

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

三人の従業員“John”、“Mary”と“Eric”の情報を前に用意された同じ **PreparedStatement**で挿入する場合、コードは次のようになります：

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

そこで、一回の“prepare”と三回の“execution”で三つのタブルを挿入します。

JDBCを通じてデータベースからデータを取得する方法

demo の便利のために、テーブルスキーマは次のように仮定します：

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

まず、Statement オブジェクトはDBMaster データベース・サーバーに接続した接続によって作成されます：

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

それから、Statementのオブジェクトの executeQuery 方法によって、クエリコマンドを実行し、結果セットを取得することができます：

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

そして、ステートメントのクエリを通じて取った結果セットを表示することができます。

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

ResultSetのオブジェクトでカーソル更新を使用するために、まず、**ResultSet** Type ResultSet.CONCUR_UPDATABLE を使ってStatement のオブジェクトを作成する必要があります：

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

ResultSet タイプについての詳細をJDBC 仕様書を参考してください。

現在、前と同じような方法を使ってクエリコマンドを実行して、**ResultSet**を取得します：

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

しかし、このたびにカーソルを使用して**ResultSet**を更新することができます。

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

データを取得している場合、より効率のメモリ割り当てを取得するため、トレースとカラムを再マップする場合重複チェックを避けるため、二つの重要な変更点があります。

- dmjdbc 以前のバージョンで、JdbcOdbcResultSet getXXXでは、毎回javaでバッファを割り当てる、それをc に渡し、c コードはバッファに渡した値を割り当てます。割り当てたバッファが再利用されません。拡張機能は、空きメモリを頻繁に割り当てることを回避することによって割り当てたバッファを再利用しようとします。
- JDBCのデバッグ機構は、DriverManager.setLogStream または DirverManager.setLogWriter を呼び出します。JdbcOdbcXXX クラスの以前のバージョンでは、JDBC 実行された方法の最初ラインがログ流れ、またはログライターをチェックし、デバッグメッセージを出力する必要があるか、確認します。このチェック動作はパフォーマンスに重要な影響を与えます。そのため、この改善はJdbcOdbc.java、全ての JdbcOdbcXXX クラスの最上位でフラグ（変数）needTraceを維持します。だから、全てのJdbcOdbcXXは、オブジェクトが作成される時にその変数を使用することができます。各方法では、フラグneedTrace だけをチェックし、DriverManager.getLogStream またはgetLogWriter を呼び出すことはありません。

dmjdbc パフォーマンスを向上することに、needTraceの問題 を注意してください。オブジェクトが作成される時に、その値が設定され、その後ユーザーはそれを変更することができません。そのため、ユーザーは以下のようなことをする場合：

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

DriverManager.setLogWriter(new
java.io.PrintWriter("c:\temp\jdbc.log"));

ResultSet rs = stmt.executeQuery("xxx");
```

それで、rs からのtrace メッセージだけがログに出力され、stmt からのではありません。その原因は、stmt が作成された後、ログライターが設定されるのためです。

JDBCを通じてblobデータをどう処理するか?

DBMasterは、Blob、Clob およびFOデータタイプを含むラージオブジェクトの操作のために強力な機能を提供します。DBMaster JDBC Driverは、

JavaプログラムでのBlobとClobデータタイプにインタフェースjava.sql.Clobとjava.sql.Blobを実装します。

DBMaster JDBC Driverを通じてJavaプログラムでのBlobとClobデータタイプの使用方法を紹介します。

DBMaster JDBC Driverを通じてAPIsが実装された詳細は、4.2章を参考してください。ここでは、BlobとClobデータ、またはFOにJDBCコードのサンプルを提供します。

CLOBサンプルの使用方法

このサンプルでは、DBMaster JDBC Driverを通じてClobデータの使用方法を示します。

このサンプルでは、テキストファイル'demo.txt'の内容をデータベースに格納し、それから、それを取得してコンソールに出力します。.

まず、Clobデータを格納するためにテーブルjdbc_clob_demoを作成します。そのテーブルはlong varcharタイプを持つカラムで構成します。

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

それから、以下のようにテーブルにテキストファイル'demo.txt'の内容を格納します：

```
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();
```

現在、データベースにテキストファイル'demo.txt'の内容を格納する仕事が終わりました。次に、その内容を取得して、コンソールに出力します。実

際には、ユーザーはそれをより有効に使用できますが、出力しません。取得および出力するためのコードは次のとおりです：

```
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);
}
```

このサンプルの全部のコードには、Appendix for Sample 1.を参考してください。

それと同時に、**ResultSet** によって取り出されたClobは、次のように新しい行 を挿入するために使用されることができます：

```
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 サンプルの使用方法

このサンプルでは、DBMaster JDBC Driverを通じてBlobデータの使用方法を示します。

このサンプルでは、データベースに図‘demo.gif’を格納して、それを取り出し、gif ファイルとして保存します。

まず、図のファイルを格納するためにテーブル**jdbc_blob_demo**を作成します。そのテーブルはlong varbinary タイプを持つカラムで構成します：

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

それから、図のファイル‘demo.gif’を読んで、**jdbc_blob_demo** に戻ります：

```
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();
```

現在、図‘demo.gif’はbinary データとしてデータベースに格納されています。次に、**jdbc_blob_demo** からのbinary データを取得して、‘dup-demo.gif’と呼ばれ、gif ファイルとして保存します。アプリケーションには、おそらく幾つのコントロールに関する図を表示します。このジョブにのコードは次のとおりです：

```
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 ();
}
```

このサンプルの完全なコードについては、 Appendix for Sample 2.を参考してください。

一方、結果セットを通じて取得されたBlob は、以下のように幾つの新しい行を挿入するために使用されることができます：

```
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();
 }
```

ファイル・オブジェクト・サンプルの使用方法

DBMasterの有効な機能として、FILE データタイプをサポートしています。FILEタイプカラムは、binaryデータを、データベースblobファイル(*.bb)外にあるファイルとして格納することができます。キーワードDB_FoTypがdmconfig.iniで“1”に設定される場合では、FILEデータタイプはJDBCプログラムために、LONG VARBINARY またはLONG VARCHARと同じように使用されることができます、キーワードの詳細については、DBA Manual (dba.chm)を参考してください。付録のthe Sample 3 では、JDBCのFILEデータタイプの使用ための完全なdemoを提供しています。

ストアドプロシージャをどう呼び出すか?

EC文でストアドプロシージャを記述し、ECプログラムファイルからdmSQLでストアドプロシージャを作成できますが、詳細はdba.chmマニュアルのストアドプロシージャのセクションを参考してください。

JDBC で以下の構文を通じてストアドプロシージャを呼び出すことができます：

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

以下のサンプルは、DBMaster JDBC Driverでストアドプロシージャの使用方法を証明します。

サンプルで使用されるテーブルスキーマは次のように示します：

```
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;
```

ここでは、使用された二つのストアドプロシージャがあります：レコードがテーブルに存在しない場合、**UPDATE_OR_INSERT**はテーブル**JDBC_SP_DEMO**にレコードを挿入するために使用されます。または、レコードがテーブルに存在している場合、**UPDATE_OR_INSERT**はテーブル**JDBC_SP_DEMO**に同じIDによってレコードを更新するために使用されます。付録Sample 4 in Appendixのupdate_or_insert.ecファイルを参考してください。

JDBCを通じてプロシージャーを呼び出す前に、まずデータベースでこのプロシージャーを作成する必要があります。DBMasterは、ストアド・プロシージャーためのECプログラムを含むファイルからのデータベースに、ストアド・プロシージャを作成するためのコマンド'CREATE PROCEDURE FROM …'を提供します。詳細については、DBA マニュアル dba.chmのストアドプロシージャのセクションを参考してください。

次に、ストアドプロシージャを呼び出す普通の手順を紹介します.. 説明を簡単にするために、呼び出しようとするプログラムが‘demo_sp’と呼ばれ、二つのパラメータを持っているとします。第一パラメータは、Stringタイプを持って、INPUTとして使用されます。第二のは、Integerタイプを持って、OUTPUTパラメータとして使用されます。また、ストアド・プロシージャ‘demo_sp’は、それぞれIntegerとStringを持つ二つのカラムを含んでいるクエリの結果セット を返します。

1. java.sql.Connection.prepareCall()を通じてCallableStatementを準備します。

ストアド・プロシージャは JDBC エスケープ構文を通じて準備される必要があります。次のコードはサンプルです：

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

2. いずれかの場合には**CallableStatement.registerOutputParameter()**を通じてoutput変数を登録します。

DBMaster JDBC Driveはストアドプロシージャにoutputパラメータをサポートします。任意のoutputパラメータがある場合、それが次のように登録される必要があります：

```
// 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);
```

インデックス(1-base started)とカラムの名前の両方が、outputパラメータを登録するためにサポートされることに注意してください。

3. CallableStatement.setXXX()を通じてinputパラメータを設定します。

PreparedStatementのように、インデックス(1-base started) とカラムの名前の両方が、inputパラメータを設定するためにサポートされます。inputパラメータが次のように設定することができます：

```
// Set the input parameter, say, the first parameter named by 'id' with varchar(10)
```

```
// datatype.  Variable cstmt is prepared by calling
Connection.prepareCall().
cstmt.setString(1,'21935265');
```

4. cstmt.executeQuery()を通じてクエリ結果を実行、取得します。

ストアドプロシージャはクエリに何かを行い、または結果セットを返す場合では、結果セットがクラスjava.sqlの例によってアクセスされることができます。cstmt.executeQuery()によって結果セットを返しました。

```
//  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) ;
}
```

インデックスーベースと名前ーベースのカラムの両方は、outputパラメータを取得するためにサポートされることに注意してください。

データベース結果セット/パラメータにメタデータをどう取得するか？

JDBC仕様書のメタデータは、高級なプログラミングに重要な役割を果たしています。

DatabaseMetaDataインタフェースは、データベースサーバーの仕様書に関する方法を含くまれています。それは、トランザクション、ストアドプロ

シージャ、外部結合などがサポートされるかどうか、または、ターゲット・データベースサーバーに関する他の情報のようなものです。

以下のコードによって**DatabaseMetaData**が取得されることができます：

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

connは、DBMasterデータベースサーバーに接続されたConnection オブジェクトです。

例えば、DBMSはトランザクションをサポートするかどうかがわからない場合では、**DatabaseMetaData**インタフェースのTransactions()をサポートする方々によってメタデータを取得することができます。

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

その上、DBMSはストアドプロシージャをサポートするかどうかを知りたい場合では、**DatabaseMetaData**インタフェースのStoredProcedure()をサポートする方法によってメタデータを取得することができます。

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

DatabaseMetaDataインタフェースは、クエリによって返した**ResultSet**の情報に関する方法を含まれています。それは、カラム数、各カラムの名前とタイプなどのようなことです。プログラマはその情報を使用して一般的なプログラミングを実現することができます。

以下のコードによってResultSetMetaDataを取得することができます：

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

rsは、幾つかのクエリ文によって返されたResultSetオブジェクトです。例えば、幾つかのクエリによって取得されたResultSetについて何も知らない場合、次のようにResultSetMetaDataを助けて、ResultSetを繰り返すことができます：

```
/* 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 インタフェースは、JDBCタイプおよびこれらの変数の精度のようなホスト変数を使用してステートメントでのパラメータの情報に関する全ての方法を含んでいます。

以下のコードによって ParameterMetaData を取得することができます：

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

Pstmtは、 幾つの接続オブジェクトを通して準備されたPreparedStatement オブジェクトです。

CallableStatementによって実行されるストアドプロシージャのパラメータに、メタデータを取得することもできます。

ここでは、サンプルとして、CallableStatement内のパラメータのメタデータ情報を取得するためにParameterMetaDataを使用するコードセクションをリストします。

```
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 ドライバーの参考

この章では、DBMaster JDBC ドライバーによってサポートされるデータタイプ、DBMaster JDBC ドライバーによって実行されるAPIs、および実行されるシステム関数を含むDBMaster JDBC ドライバーを説明します。

4.1 サポートするデータタイプ

以下のテーブル5-1は、DBMaster JDBC ドライバーによってサポートされるデータタイプを表示します。

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

テーブル 4-1: サポートデータタイプ

4.2 実行される JDBC API

JDBC Type II ドライバー

目前、DBMaster JDBC Type IIドライバーは、JDBC 2.0 および JDBC 3.0標準に最も有効な方法とインタフェースをサポートします。そのインタフェースと方法が以下に記述されます。

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

テーブル 4-2: java.sql.Blob

JAVA.SQL.CALLABLESTATEMENT

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

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

length) throws java.sql.SQLException;	
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

テーブル 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

テーブル 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

テーブル 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	1.0

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

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

getMaxSchemaNameLength() throws SQLException	1.0
getMaxStatementLength() throws SQLException	1.0
getMaxStatements() throws SQLException	1.0
getMaxTableNameLength() throws SQLException	1.0
getMaxTablesInSelect() throws SQLException	1.0
getMaxTablesInSelect() throws java.sql.SQLException;	1.0
getMaxUserNameLength() throws java.sql.SQLException;	1.0
getPrimaryKeys(java.lang.String catalog,,java.lang.String schema,,java.lang.String table) throws java.sql.SQLException;	1.0
getProcedureColumns(java.lang.String catalog,,java.lang.String schemaPattern,,java.lang.String procedureNamePattern,,java.lang.String columnNamePattern) throws java.sql.SQLException;	1.0
getProcedureTerm() throws java.sql.SQLException;	1.0
getProcedures(java.lang.String catalog,,java.lang.String schemaPattern,java.lang.String procedureNamePattern) throws java.sql.SQLException;	1.0
getSQLKeywords() throws java.sql.SQLException;	1.0
getSQLStateType() throws java.sql.SQLException;	3.0
getSchemaTerm() throws java.sql.SQLException;	1.0
getSchemas() throws java.sql.SQLException;	1.0
getSearchStringEscape() throws java.sql.SQLException;	1.0
getStringFunctions() throws java.sql.SQLException;	1.0

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

java.sql.SQLException;	
supportsCatalogsInIndexDefinitions() throws java.sql.SQLException;	1.0
supportsCatalogsInPrivilegeDefinitions() throws java.sql.SQLException;	1.0
supportsCatalogsInProcedureCalls() throws java.sql.SQLException;	1.0
supportsCatalogsInTableDefinitions() throws java.sql.SQLException;	1.0
supportsColumnAliasing() throws java.sql.SQLException;	1.0
supportsConvert() throws java.sql.SQLException;	1.0
supportsConvert(int fromType,int toType) throws java.sql.SQLException;	1.0
supportsCoreSQLGrammar() throws java.sql.SQLException;	1.0
supportsCorrelatedSubqueries() throws java.sql.SQLException;	1.0
supportsDataDefinitionAndDataManipulationTransactions( ) throws java.sql.SQLException;	1.0
supportsDataManipulationTransactionsOnly() throws java.sql.SQLException;	1.0
supportsDifferentTableCorrelationNames() throws java.sql.SQLException;	1.0
supportsExpressionsInOrderBy() throws java.sql.SQLException;	1.0
supportsExtendedSQLGrammar() throws java.sql.SQLException;	1.0

supportsFullOuterJoins() throws java.sql.SQLException;	1.0
supportsGetGeneratedKeys() throws java.sql.SQLException;	3.0
supportsGroupBy() throws java.sql.SQLException;	1.0
supportsGroupByBeyondSelect() throws java.sql.SQLException;	1.0
supportsGroupByUnrelated() throws java.sql.SQLException;	1.0
supportsIntegrityEnhancementFacility() throws java.sql.SQLException;	1.0
supportsLikeEscapeClause() throws java.sql.SQLException;	1.0
supportsLimitedOuterJoins() throws java.sql.SQLException;	1.0
supportsMinimumSQLGrammar() throws java.sql.SQLException;	1.0
supportsMixedCaseIdentifiers() throws java.sql.SQLException;	1.0
supportsMixedCaseQuotedIdentifiers() throws java.sql.SQLException;	1.0
supportsMultipleOpenResults() throws java.sql.SQLException;	3.0
supportsMultipleTransactions() throws java.sql.SQLException;	1.0
supportsNamedParameters() throws java.sql.SQLException;	3.0
supportsNonNullableColumns() throws java.sql.SQLException;	1.0
supportsOpenCursorsAcrossCommit() throws java.sql.SQLException;	1.0

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

テーブル 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

テーブル 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

テーブル 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	2.0

java.sql.SQLException;	
setDate(int parameterIndex,,java.sql.Date x,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
setDate(int parameterIndex,,java.sql.Date x) throws java.sql.SQLException;	1.0
setNull(int parameterIndex,int jdbcype) throws java.sql.SQLException;	1.0
setObject(int parameterIndex,,java.lang.Object x) throws java.sql.SQLException;	2.0
setObject(int parameterIndex,,java.lang.Object x,int targetJdbcType,int scale) throws java.sql.SQLException;	1.0
setObject(int parameterIndex,,java.lang.Object x,int targetJdbcType) throws java.sql.SQLException;	1.0
setString(int parameterIndex,,java.lang.String x) throws java.sql.SQLException;	1.0
setTime(int parameterIndex,,java.sql.Time x) throws java.sql.SQLException;	1.0
setTime(int parameterIndex,,java.sql.Time x,,java.util.Calendar cal) throws java.sql.SQLException;	2.0
executeQuery() throws java.sql.SQLException;	1.0
executeUpdate() throws java.sql.SQLException;	1.0
getParameterMetaData() throws java.sql.SQLException;	3.0
close() throws java.sql.SQLException;	1.0
execute(java.lang.String sql) throws java.sql.SQLException;	1.0
getConnection() throws java.sql.SQLException;	2.0
clearWarnings() throws java.sql.SQLException;	1.0

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

テーブル 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 java.sql.SQLException;	1.0
getAsciiStream(int columnIndex) throws java.sql.SQLException;	1.0
getBigDecimal(java.lang.String columnName) throws java.sql.SQLException;	1.0
getBigDecimal(int columnIndex) throws java.sql.SQLException;	1.0
getBigDecimal(int columnIndex,int scale) throws java.sql.SQLException;	1.0
getBigDecimal(java.lang.String columnName,int scale) throws java.sql.SQLException;	1.0
getBinaryStream(java.lang.String columnName) throws java.sql.SQLException;	1.0
getBinaryStream(int columnIndex) throws java.sql.SQLException;	1.0
getBlob(int columnIndex) throws java.sql.SQLException;	2.0
getBlob(java.lang.String columnName) throws java.sql.SQLException;	2.0
getCharacterStream(int columnIndex) throws java.sql.SQLException;	2.0
getCharacterStream(java.lang.String columnName) throws java.sql.SQLException;	2.0
getClob(int columnIndex) throws java.sql.SQLException;	2.0
getClob(java.lang.String columnName) throws java.sql.SQLException;	2.0
getConcurrency() throws java.sql.SQLException;	2.0

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

java.sql.SQLException;	
insertRow() throws java.sql.SQLException;	2.0
isAfterLast() throws java.sql.SQLException;	2.0
isBeforeFirst() throws java.sql.SQLException;	2.0
isFirst() throws java.sql.SQLException;	2.0
isLast() throws java.sql.SQLException;	2.0
moveToCurrentRow() throws java.sql.SQLException;	2.0
moveToInsertRow() throws java.sql.SQLException;	2.0
refreshRow() throws java.sql.SQLException;	2.0
relative(int rows) throws java.sql.SQLException;	2.0
rowDeleted() throws java.sql.SQLException;	2.0
rowInserted() throws java.sql.SQLException;	2.0
rowUpdated() throws java.sql.SQLException;	2.0
setFetchDirection(int direction) throws java.sql.SQLException;	2.0
setFetchSize(int rows) throws java.sql.SQLException;	2.0
updateArray(int columnIndex,,java.sql.Array x) throws java.sql.SQLException;	3.0
updateArray(java.lang.String columnName,,java.sql.Array x) throws java.sql.SQLException;	3.0
updateAsciiStream(int columnIndex,,java.io.InputStream x,int length) throws java.sql.SQLException;	2.0
updateAsciiStream(java.lang.String columnName,,java.io.InputStream x,int length) throws java.sql.SQLException;	2.0

updateBigDecimal(int columnIndex,java.math.BigDecimal x) throws java.sql.SQLException;	2.0
updateBigDecimal(java.lang.String columnName,,java.math.BigDecimal x) throws java.sql.SQLException;	2.0
updateBinaryStream(int columnIndex,java.io.InputStream x,int length) throws java.sql.SQLException;	2.0
updateBinaryStream(java.lang.String columnName,,java.io.InputStream x,int length) throws java.sql.SQLException;	2.0
updateBlob(int columnIndex,,java.sql.Blob x) throws java.sql.SQLException;	3.0
updateBlob(java.lang.String columnName,,java.sql.Blob x) throws java.sql.SQLException;	3.0
updateBoolean(int columnIndex,boolean x) throws java.sql.SQLException;	2.0
updateBoolean(java.lang.String columnName,boolean x) throws java.sql.SQLException;	2.0
updateByte(int columnIndex,byte x) throws java.sql.SQLException;	2.0
updateByte(java.lang.String columnName,byte x) throws java.sql.SQLException;	2.0
updateBytes(int columnIndex,byte[] x) throws java.sql.SQLException;	2.0
updateBytes(java.lang.String columnName,byte[] x) throws java.sql.SQLException;	2.0
updateCharacterStream(java.lang.String	2.0

columnName,,java.io.Reader x1,int length) throws java.sql.SQLException;	
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

テーブル 4-10 java.sql.ResultSet

JAVA.SQL.RESULTSETMETADATA

Methods	Version
isReadOnly(int column) throws java.sql.SQLException;	1.0
getCatalogName(int column) throws java.sql.SQLException;	1.0
getColumnClassName(int column) throws java.sql.SQLException;	2.0
getColumnCount() throws java.sql.SQLException;	1.0
getColumnDisplaySize(int column) throws java.sql.SQLException;	1.0
getColumnLabel(int column) throws java.sql.SQLException;	1.0
getColumnName(int column) throws java.sql.SQLException;	1.0
getColumnType(int column) throws java.sql.SQLException;	1.0

getColumnTypeName(int column) throws java.sql.SQLException;	1.0
getPrecision(int column) throws java.sql.SQLException;	1.0
getScale(int column) throws java.sql.SQLException;	1.0
getSchemaName(int column) throws java.sql.SQLException;	1.0
getTableName(int column) throws java.sql.SQLException;	1.0
isAutoIncrement(int column) throws java.sql.SQLException;	1.0
isCaseSensitive(int column) throws java.sql.SQLException;	1.0
isCurrency(int column) throws java.sql.SQLException;	1.0
isDefinitelyWritable(int column) throws 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

テーブル 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 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

テーブル 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

テーブル 4-13 javax.sql.ConnectionPoolDataSource

javax.SQL.Datesource

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

テーブル 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

テーブル 4-15 javax.sql. PooledConnection

javax.sql.XAConnection

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

テーブル 4-16 javax.sql. PooledConnection

javax.transaction.xa.Xid

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

テーブル 4-17 javax.transaction.xa.Xid

JDBC Type III ドライバー

目前、DBMaster JDBC Type IIIドライバーは、JDBC 2.0, JDBC 3.0とJDBC 4.0標準に最も有効な方法とインタフェースをサポートします。そのインタフェースと方法が以下に記述されます

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

テーブル 4-18 java.sql.Blob

java.sql.CallableStatement

Methods	Version
getBigDecimal(int parameterIndex) throws SQLException	2.0
getBlob(int parameterIndex) throws SQLException	2.0
getBoolean(int parameterIndex) throws SQLException	1.0
getByte(int parameterIndex) throws SQLException	1.0
getBytes(int parameterIndex) throws SQLException	1.0
getCharacterStream(int parameterIndex) throws SQLException	4.0
getClob(int parameterIndex) throws SQLException	2.0
getDate(int parameterIndex) throws SQLException	1.0
getDate(int parameterIndex, Calendar cal) throws SQLException	2.0
getDouble(int parameterIndex) throws SQLException	1.0
getFloat(int parameterIndex) throws SQLException	1.0
getInt(int parameterIndex)	1.0
getLong(int parameterIndex) throws SQLException	1.0
getNCharacterStream(int parameterIndex) throws SQLException	4.0
getNClob(int parameterIndex) throws SQLException	4.0
getNString(int parameterIndex) throws SQLException	4.0
getObject(int parameterIndex) throws SQLException	1.0
getShort(int parameterIndex) throws SQLException	1.0
getString(int parameterIndex) throws SQLException	1.0
getTime(int parameterIndex) throws SQLException	1.0
getTime(int parameterIndex, Calendar cal) throws SQLException	2.0

getTimestamp(int parameterIndex) throws SQLException	1.0
getTimestamp(int parameterIndex, Calendar cal) throws SQLException	2.0
registerOutParameter(int parameterIndex, int sqlType) throws SQLException	1.0
registerOutParameter(int parameterIndex, int sqlType, int scale) throws SQLException	1.0
registerOutParameter(int parameterIndex, int sqlType, String typeName) throws SQLException	2.0
getNClob(int parameterIndex) throws SQLException	4.0
getNString(int parameterIndex) throws SQLException	4.0
wasNull() throws SQLException	1.0

テーブル 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

テーブル 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

テーブル 4-21 java.sql.Connection

java.sql.DatabaseMetaData

Methods	Version
allProceduresAreCallable() throws SQLException	1.0
allTablesAreSelectable() throws SQLException	1.0
dataDefinitionCausesTransactionCommit() throws SQLException	1.0
dataDefinitionIgnoredInTransactions() throws SQLException	1.0
deletesAreDetected(int type) throws SQLException	2.0
doesMaxRowSizeIncludeBlobs() throws SQLException	1.0
getAttributes(String catalog, String schemaPattern,String typeNamePattern, String attributeNamePattern) throws SQLException	3.0
getBestRowIdentifier(String catalog, String schema,String table, int scope, boolean nullable) throws SQLException	1.0
getCatalogSeparator() throws SQLException	1.0
getCatalogTerm() throws SQLException	1.0
getCatalogs() throws SQLException	1.0
getColumnPrivileges(String catalog, String schema,String table, String columnNamePattern) throws SQLException	1.0
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern) throws SQLException	1.0

getConnection() throws SQLException	2.0
getCrossReference(String parentCatalog, String parentSchema,String parentTable, String foreignCatalog, String foreignSchema,String foreignTable) throws SQLException	1.0
getDatabaseMajorVersion() throws SQLException	3.0
getDatabaseMinorVersion() throws SQLException	3.0
getDatabaseProductName() throws SQLException	1.0
getDatabaseProductVersion() throws SQLException	1.0
getDefaultTransactionIsolation() throws SQLException	1.0
getDriverMajorVersion()	1.0
getDriverMinorVersion()	1.0
getDriverName() throws SQLException	1.0
getDriverVersion() throws SQLException	1.0
getExportedKeys(String catalog, String schema, String table)throws SQLException	1.0
getExtraNameCharacters() throws SQLException	1.0
getIdentifierQuoteString() throws SQLException	1.0
getImportedKeys(String catalog, String schema, String table) throws SQLException	1.0
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate) throws SQLException	1.0
getJDBCMajorVersion() throws SQLException	3.0
getJDBCMinorVersion() throws SQLException	3.0
getMaxBinaryLiteralLength() throws SQLException	1.0

getMaxCatalogNameLength() throws SQLException	1.0
getMaxCharLiteralLength() throws SQLException	1.0
getMaxColumnNameLength() throws SQLException	1.0
getMaxColumnsInGroupBy() throws SQLException	1.0
getMaxColumnsInIndex() throws SQLException	1.0
getMaxColumnsInOrderBy() throws SQLException	1.0
getMaxColumnsInSelect() throws SQLException	1.0
getMaxColumnsInTable() throws SQLException	1.0
getMaxConnections() throws SQLException	1.0
getMaxCursorNameLength() throws SQLException	1.0
getMaxIndexLength() throws SQLException	1.0
getMaxProcedureNameLength() throws SQLException	1.0
getMaxRowSize() throws SQLException	1.0
getMaxSchemaNameLength() throws SQLException	1.0
getMaxStatementLength() throws SQLException	1.0
getMaxStatements() throws SQLException	1.0
getMaxTableNameLength() throws SQLException	1.0
getMaxTablesInSelect() throws SQLException	1.0
getMaxUserNameLength() throws SQLException	1.0
getNumericFunctions() throws SQLException	1.0
getPrimaryKeys(String catalog, String schema, String table)throws SQLException	1.0

getProcedureColumns(String catalog, String schemaPattern,String procedureNamePattern, String columnNamePattern)throws SQLException	1.0
getProcedureTerm() throws SQLException	1.0
getProcedures(String catalog, String schemaPattern,String procedureNamePattern) throws SQLException	1.0
getResultSetHoldability() throws SQLException	3.0
getSQLKeywords() throws SQLException	1.0
getSQLStateType() throws SQLException	3.0
getSchemaTerm() throws SQLException	1.0
getSchemas() throws SQLException;	1.0
getSearchStringEscape() throws SQLException	1.0
getStringFunctions() throws SQLException	1.0
getSuperTables(String catalog, String schemaPattern, String tableNamePattern) throws SQLException	3.0
getSuperTypes(String catalog, String schemaPattern,String typeNamePattern) throws SQLException	3.0
getSystemFunctions() throws SQLException	1.0
getTablePrivileges(String catalog, String schemaPattern,String tableNamePattern) throws SQLException	1.0
getTableTypes() throws SQLException	1.0
getTables(String catalog, String schemaPattern,String tableNamePattern, String[] types) throws SQLException	1.0
getTimeDateFunctions() throws SQLException	1.0
getTypeInfo() throws SQLException	1.0

getUDTs(String catalog, String schemaPattern,String typeNamePattern, int[] types) throws SQLException	2.0
getURL() throws SQLException	1.0
getUserName() throws SQLException	1.0
getVersionColumns(String catalog, String schema, String table) throws SQLException	1.0
insertsAreDetected(int type) throws SQLException	2.0
isCatalogAtStart() throws SQLException	1.0
isReadOnly() throws SQLException	1.0
locatorsUpdateCopy() throws SQLException	3.0
nullPlusNonNullIsNull() throws SQLException	1.0
nullsAreSortedAtEnd() throws SQLException	1.0
nullsAreSortedAtStart() throws SQLException	1.0
nullsAreSortedHigh() throws SQLException	1.0
nullsAreSortedLow() throws SQLException	1.0
othersDeletesAreVisible(int type) throws SQLException	2.0
othersInsertsAreVisible(int type) throws SQLException	2.0
othersUpdatesAreVisible(int type) throws SQLException	2.0
ownDeletesAreVisible(int type) throws SQLException	2.0
ownInsertsAreVisible(int type) throws SQLException	2.0
ownUpdatesAreVisible(int type) throws SQLException	2.0
storesLowerCaseIdentifiers() throws SQLException	2.0
storesLowerCaseQuotedIdentifiers() throws SQLException	1.0

storesMixedCaseIdentifiers() throws SQLException	1.0
storesMixedCaseQuotedIdentifiers() throws SQLException	1.0
storesUpperCaseIdentifiers() throws SQLException	1.0
storesUpperCaseQuotedIdentifiers() throws SQLException	1.0
supportsANSI92EntryLevelSQL() throws SQLException	1.0
supportsANSI92FullSQL() throws SQLException	1.0
supportsANSI92IntermediateSQL() throws SQLException	1.0
supportsAlterTableWithAddColumn() throws SQLException	1.0
supportsAlterTableWithDropColumn() throws SQLException	1.0
supportsBatchUpdates() throws SQLException	2.0
supportsCatalogsInDataManipulation() throws SQLException	1.0
supportsCatalogsInIndexDefinitions() throws SQLException	1.0
supportsCatalogsInPrivilegeDefinitions() throws SQLException	1.0
supportsCatalogsInProcedureCalls() throws SQLException	1.0
supportsCatalogsInTableDefinitions() throws SQLException	1.0
supportsColumnAliasing() throws SQLException	1.0
supportsConvert() throws SQLException	1.0
supportsConvert(int fromType, int toType) throws SQLException	1.0
supportsCoreSQLGrammar() throws SQLException	1.0
supportsCorrelatedSubqueries() throws SQLException	1.0
supportsDataDefinitionAndDataManipulationTransactions()throws SQLException	1.0
supportsDataManipulationTransactionsOnly() throws	1.0

SQLException	
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

テーブル 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

テーブル 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

テーブル 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)	4.0

throws SQLException	
setBigDecimal(int parameterIndex, BigDecimal x) throws SQLException	1.0
setBinaryStream(int parameterIndex, InputStream x)throws SQLException	4.0
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	4.0

SQLException	
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

テーブル 4-25 java.sql.PreparedStatement

java.sql.ResultSet

Methods	Version
absolute(int row) throws SQLException	2.0
afterLast() throws SQLException	2.0
beforeFirst() throws SQLException	2.0
cancelRowUpdates() throws SQLException	2.0
clearWarnings() throws SQLException	1.0
close() throws SQLException	1.0
deleteRow() throws SQLException	2.0
findColumn(String columnLabel) throws SQLException	1.0
first() throws SQLException	2.0

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

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

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

SQLException	
getTimestamp(String columnLabel, Calendar cal) throws SQLException	2.0
getType() throws SQLException	2.0
getWarnings() throws SQLException	1.0
insertRow() throws SQLException	2.0
isAfterLast() throws SQLException	2.0
isBeforeFirst() throws SQLException	2.0
isClosed() throws SQLException	4.0
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 x) throws SQLException	4.0
updateAsciiStream(String columnLabel, InputStream x)throws SQLException	4.0
updateAsciiStream(int columnIndex, InputStream x, int length) throws SQLException	2.0
updateAsciiStream(String columnLabel, InputStream x, int length)throws SQLException	2.0
updateAsciiStream(int columnIndex, InputStream x, long length)throws SQLException	4.0
updateAsciiStream(String columnLabel, InputStream x, long length)throws SQLException	4.0
updateBigDecimal(int columnIndex, BigDecimal x) throws SQLException	2.0
updateBigDecimal(String columnLabel, BigDecimal x)throws SQLException	2.0
updateBinaryStream(int columnIndex, InputStream x)throws SQLException	4.0
updateBinaryStream(String columnLabel, InputStream x)throws SQLException	4.0
updateBinaryStream(int columnIndex, InputStream x, int length)throws SQLException	2.0
updateBinaryStream(String columnLabel, InputStream x, int length)throws SQLException	2.0
updateBinaryStream(int columnIndex, InputStream x, long length) throws SQLException	4.0
updateBinaryStream(String columnLabel, InputStream	4.0

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 x)throws SQLException	4.0

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

SQLException	
updateDouble(String columnLabel, double x) throws SQLException	2.0
updateFloat(int columnIndex, float x) throws SQLException	2.0
updateFloat(String columnLabel, float x) throws SQLException	2.0
updateInt(int columnIndex, int x) throws SQLException	2.0
updateInt(String columnLabel, int x) throws SQLException	2.0
updateLong(int columnIndex, long x) throws SQLException	2.0
updateLong(String columnLabel, long x) throws SQLException	2.0
updateNCharacterStream(int columnIndex, Reader x) throws SQLException	4.0
updateNCharacterStream(String columnLabel, Reader reader) throws SQLException	4.0
updateNCharacterStream(int columnIndex, Reader x, long length)throws SQLException	4.0
updateNCharacterStream(String columnLabel, Reader reader,long length) throws SQLException	4.0
updateNClob(int columnIndex, java.sql.NClob nClob) throws SQLException	4.0
updateNClob(String columnLabel, java.sql.NClob nClob) throws SQLException	4.0
updateNClob(int columnIndex, Reader reader) throws	4.0

SQLException	
updateNClob(String columnLabel, Reader reader)throws SQLException	4.0
updateNClob(int columnIndex, Reader reader, long length)throws SQLException	4.0
updateNClob(String columnLabel, Reader reader, long length) throws SQLException	4.0
updateNString(int columnIndex, String nString)throws SQLException	4.0
updateNString(String columnLabel, String nString) throws SQLException	4.0
updateNull(int columnIndex) throws SQLException	2.0
updateNull(String columnLabel) throws SQLException	2.0
updateObject(int columnIndex, Object x) throws SQLException	2.0
updateObject(String columnLabel, Object x) throws SQLException	2.0
updateRow() throws SQLException	2.0
updateShort(int columnIndex, short x) throws SQLException	2.0
updateShort(String columnLabel, short x) throws SQLException	2.0
updateString(int columnIndex, String x) throws SQLException	2.0
updateString(String columnLabel, String x) throws SQLException	2.0

updateTime(int columnIndex, Time x) throws SQLException	2.0
updateTime(String columnLabel, Time x) throws SQLException	2.0
updateTimestamp(int columnIndex, Timestamp x) throws SQLException	2.0
updateTimestamp(String columnLabel, Timestamp x)throws SQLException	2.0
wasNull() throws SQLException	1.0

テーブル 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

テーブル 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

テーブル 4-28 java.sql.Statement

javax.sql.ConnectionPoolDataSource

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

テーブル 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

テーブル 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

テーブル 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

テーブル 4-32 javax.sql.XAConnection

javax.sql.XADataSource

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

テーブル 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

テーブル 4-34 javax.transaction.xa. XAResource

javax.transaction.xa.Xid

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

テーブル 4-35 javax.transaction.xa.Xid

4.3 実行されるシステム関数

以下のテーブルは、DBMaster JDBCによって実行されるシステム関数を表示します。

数値関数	ABS,ACOS,ASIN,ATAN,ATAN2,CEILING,COS,COT, DEGREES,EXP,FLOOR,LOG,LOG10,MOD,PI,POWER, RADIANS,RAND,ROUND,SIGN,SIN,SQRT,TAN
文字列関数	ASCII,CHAR,CONCAT,DIFFERENCE,INSERT,LCASE,LEFT,LENGTH,LOCATE,LTRIM,REPEAT,REPLACE,RIGHT,RTRIM,SPACE,SUBSTRING,UCASE
システム	DBNAME, IFNULL, USERNAME
時間データの関数	CURDATE,CURTIME,DAYNAME,DAYOFMONTH,DAYOFWEEK,DAYOFYEAR,HOUR,MINUTE,MO

	NTH,MONTHNAME,NOW,QUARTER,SECOND, TIMESTAMPADD, TIMESTAMPDIFF,WEEK,YEAR
例	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.
コマンドライン	This format is commonly used to show the JDBC code or the other code that the procedure needed

テーブル 4-36実行される関数

5 Q&A

Q1: “ClassNotFoundException “エラーが発生する原因：DBMaster.sql.JdbcOdbcDriver SQLException：DBMaster JDBC ドライバーによってデータベースにアクセスするJavaプログラムを実行する場合、適切なドライバーがありません？

A1: Javaプログラムを実行する時、クラスバスにあるdmjdbc20.jar または dmjdbc30.jarを参考してください。詳細には、サンプルを実行する方法を参考してください。

Q2: “Exception in thread "main" エラーが発生する原因 java.lang.UnsatisfiedLinkError： DBMaster JDBC ドライバーによってデータベースにアクセスするJavaプログラムを実行する場合、dmjdbcxx in java.library.pathがありません？

A2:ユーザーは、DBMaster JDBC ドライバーによってデータベースにアクセスするJavaプログラムを実行するには、DBMaster ネイティブドライバーが必要です。java ライブラリのpass変数に含んでいるjava.library.pathにあるネイティブドライバーdmjdbcxx.dll (so)を参考してください。詳細には、サンプルを実行する方法を参考してください。

Q3: DBMaster JDBC ドライバーによってデータベースにアクセスするJavaプログラムを実行する場合では、エラー “SQLException：接続を確立するために失敗する(8007), [DBMaster]はコンフィギュレーションファイルで指定されたデータベースが見つけられません。”の発生する原因？

A3: dmconfig.iniファイルに、アクセスする予定のデータベースがよく構成されたことを参考してください。dmconfig.iniについての詳細は、DBAマニュアル(dba.chm)を参考してください。

Q4: DBMaster JDBC ドライバーはトランザクションをサポートしますか、トランザクションを使用するための提案がありますか？

A4: DBMasteはJDBCでトランザクションをサポートします。トランザクションを持つ他のDBMSとして、プログラムが慎重にデザインされるべきで、長いトランザクションが小さなものに分割されることが必要です。トランザクションがマルチスレッド環境で使用される場合、実行する予定のトランザクションをデザインする時、デッドロックの問題を注意してください。

Q5: PreparedStatement.setDecimal()方法を呼び出す時、UnsupportedExceptionが発生する原因？

A5:現在、DBMaster は **java.sql.PreparedStatement** インタフェースで **java.sql.PreparedStatement**.setDecimal を実行しません。解決方法はデータタイプ Decimal でカラムに **java.sql.PreparedStatement**.setString()を使用することです。DBMaster JDBC ドライバーによって **java.sql.PreparedStatement** インタフェースを実行する詳細は、このドキュメントの章 4.2.8を参考してください。DBMaster JDBC ドライバーによって実行された JDBC 標準のすべてのインタフェースが、以前の章４にリストされました。

Q6: DBMasterバンドルバージョンでDBMaster JDBC ドライバーをどう使用するか？

A6: 最初に、Java APIはdmjdbcxx.jarをロードする必要があります、このjarにはdmjdbcxx.dllとdmapixx.dllをロードする予定です。Java VM:Java –Djava.library.path=C:\YourBundlePathを起動する場合、これらの.dllパスを見つけるために、Javaライブラリに次のパス設定を追加してください。

6 付録のサンプルコード

付録では、以前の章で使用されたサンプルのコードを完全に提供します。

6.1 サンプル 1 Clob の使用方法

このサンプルは、RUN SAMPLE PROGRAMセクションの場合と同じ方法で実行されます。

詳細は、RUN SAMPLE PROGRAM セクションを参考してください。

ファイル： Clob.javaの使用方法

```
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 サンプル 2 Blob の使用方法

このサンプルは、RUN SAMPLE PROGRAMセクションの場合と同じ方法で実行されます。

詳細は、RUN SAMPLE PROGRAM セクションを参考してください。

ファイル：Blob.javaの使用方法

```
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 サンプル 3 FO の使用方法

また、 このサンプルは、RUN SAMPLE PROGRAMセクションの場合と同じ方法で実行されます。

詳細は、 RUN SAMPLE PROGRAM セクションを参考してください。

ファイル： Fo.javaの使用方法

```
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 サンプル 4 ストアドプロシージャの使用方法デモ

以下は、このサンプルプログラムを実行するためのステップです：

1. 次のように、dmSQLでテーブルスキーマを作成します：

```
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. 次のように、dmSQLでプロシージャinsert_or_updateを作成します：

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

まず、ディレクトリ DBMaster/5.x/binにファイル'insert_or_update.ec'をコピーすることを注意してください。

3. 次のように、dmSQL でプロシージャquery_data を作成します：

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

まず、ディレクトリDBMaster/5.x/bin にファイル 'queryupdate.ec'をコピーすることを注意してください。

4. javacによってプログラムUsageOfBlob.javaをコンパイルし、dmjdbc20.jarを含んでいるクラスパスでのjavaによってそのプログラムを実行します。

そのプログラムを実行する前に、データベースDBNAMEを起動する必要があることを注意してください。サンプルプログラムを実行している詳細は、RUN SAMPLE PROGRAMセクションを参考してください。

ファイル： insert_or_update.ec

ファイル： 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;
}
```

ファイル： query_data.ec

ファイル： 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;
}
```

ファイル：UsageOfStoredProcedure.java

ファイル：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){}
        }
    }
}
```