DBMaker 5.4.5

DBMaker Type 4 JDBC Reference Guide

Version: 01.00

Document No: 54/DBM545-T04272023-01-T4JD

Author: DBMaker Support Team, SYSCOM Computer Engineering CO.

Publication Date: 27 April, 2023

凌羣電腦
THE SYSCOM GROUP

Table of Content

1. Overview 4

2. JDBC Drivers 5

3. Connecting to a Database 6

3.1 Import package for JDBC Driver 6

3.2 Registering the Driver 6

3.3 Creating the Connection 6

3.4 JDBC connection with properties File 10

4. Executing SQL Statements 12

4.1 Creating a Statement Object 12

4.1.1 execute() method 12

4.1.2 executeUpdate() method 13

4.1.3 executeQuery() method 13

4.1.4 Closing the ResultSet and Statement Objects 14

4.2. PreparedStatement 14

4.3. CallableStatement 15

5. Parsing Query Results 18

6. Parsing Metadata 20

6.1 DatabaseMetadata 20

6.2 ResultSetMetadata 20

7. Closing the Resources 22

8. Connecting with DataSource Objects 23

8.1 DataSource 23

8.2 XADataSource 24

9. Conclusion 25

10. The full source code of the examples 26

TableName: Employees 26

Stored Procedure: insertEmployee.sp.. 26

DBMakerType4DriverDemo.java.. 28

TestDataSource.java... 40

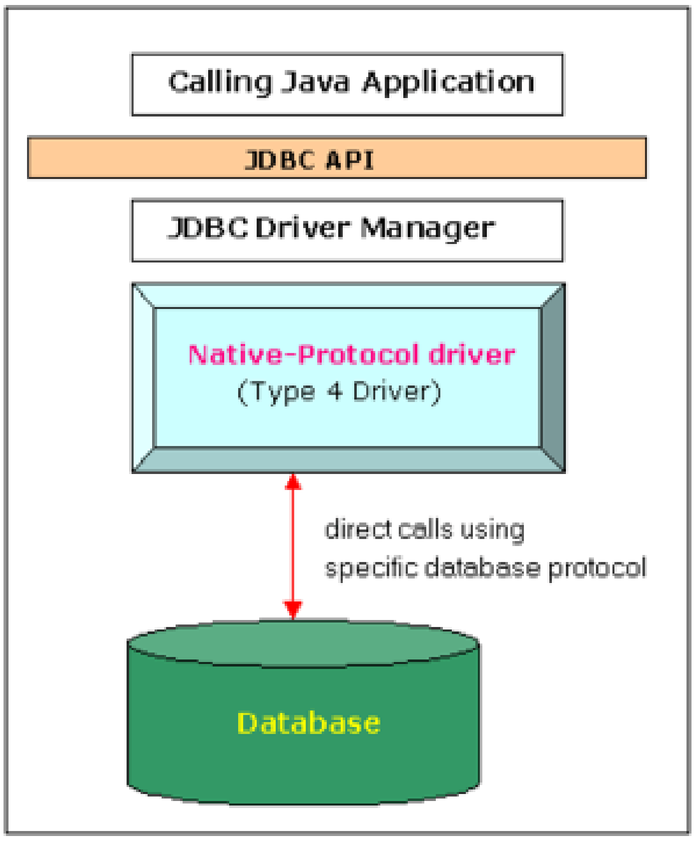
TestXADataSource.java 42

1. Overview

In this article, we're going to take a look at type 4 JDBC (Java Database Connectivity) which is an API for connecting and executing queries on DBMaker database.

2. JDBC Drivers

The JDBC Driver for DBMaker is a Type 4 JDBC driver that provides database connectivity through the standard JDBC application program interfaces (APIs) available in Java Platform, Enterprise Editions.



The Type 4 JDBC driver in DBMaker is certified to work with the 5.4.5 releases. However, they are not certified to work with older database releases, such as 5.4.4.

3. Connecting to a Database

3.1 Import package for JDBC Driver

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

3.2 Registering the Driver

The driver for the type 4 jdbc is dmjdbct4.jar. We need download the driver from installed Directories of DBMaker:

```
Install on Windows
C:\DBMaker\5.4\bin\dmjdbct4.jar


Install on Linux
/home/dbmaker/5.4/lib/java/dmjdbct4.jar
```

Next, let's register the driver using the Class.forName() method, which dynamically loads the driver class:

```
Class.forName("dbmaker.sql.type4.Driver");
```

Note: Applications do not need to explicitly load the dbmaker.sql.type4.Driver class because the JDBC driver jar supports the Java Service Provider mechanism.

3.3 Creating the Connection

To open a connection, we can use the *getConnection()* method of *DriverManager* class. This method requires a connection URL *String* parameter:

```
try (Connection conn = DriverManager
            .getConnection("jdbc:dbmaker:type4://127.0.0.1:2453/DBSAMPLE5", "sysadm", "")) {
      // use con here
}
```

The general form of the connection URL is:

```
jdbc:dbmaker:type4://serverName:portNumber/databseName;property
```

Type 4 Overview of driver properties

Property Name	**Description**	**Default Value**
jdbc:dbmaker:type4://	(Required) is known as the subprotocol	None
serverName	(Required) is the host name or IP address of the database server	None

portNumber	(Required) is the port number of the database server	None
/databseName	(Required) is the database name	None
property		
user	(Optional) User credentials	None
password	(Optional) password credentials	None
atcmt	(Optional) on or off status of the auto-commit mode	default value: 1 valid range: 0, 1
ctimo	(Optional) the connection time-out value, in seconds	default value: 5 (seconds) valid range: 5 ~ 1:00:00 (1 hour)
strsz	(Optional) the length of returned data of STRING type, used only by user-	default value: 255 valid range: 1 ~ 4096

	defined function (UDF)	
strop	(Optional) space padding is removed before applying the string concatenation operator	default value: 0 valid range: 0, 1
dscmt	(Optional) commit a transaction when an application is disconnecting from the database	default value: 0 valid range: 0, 1
ltimo	(Optional) the lock time-out value in seconds	default value: 5 valid range: -1 ~ 65535
netzc	(Optional) the on/off status of the data compression	default value: 0 valid range: 0, 1

Note: atcmt=1 is equal to DB_ATCMT=1

JDBC URL Formats for DBMaker:

```
String url="jdbc:dbmaker:type4://127.0.0.1:2453/DBSAMPLE5;
user=sysadm;password=;atcmt=1;CTIMO=20;strsz=600;strop=0;
dscmt=1;ltimo=30;netzc=1"
```

3.4 JDBC connection with properties File

A resource bundle file or properties file is one which contains the data in the form of (key, value) pair:

connection.prop

```
db.driver.class=dbmaker.sql.type4.Driver
db.conn.url=jdbc:dbmaker:type4://127.0.0.1:2453/DBSAMPLE5
db.username=sysadm
db.password=
```

Now you can get the above connection data in your java code by using the class java.util.Properties as below.

```
FileInputStream fis=new FileInputStream("connection.prop");
// Create Properties object.
Properties props = new Properties();
props.load(fis);

// Get each property value.
String dbDriver = props.getProperty("db.driver.class");
String dbConnUrl = props.getProperty("db.conn.url");

String dbUserName = props.getProperty("db.username");

String dbPassword = props.getProperty("db.password");



// Get database connection object.

Connection dbConn = DriverManager.getConnection(dbConnUrl,

dbUserName, dbPassword);
```

4. Executing SQL Statements

The *Statement* interface contains the essential functions for executing SQL commands.

4.1 Creating a Statement Object

```
try (Statement stmt = conn.createStatement()) {
    // use stmt here
}
```

Anyway, executing SQL instructions can be done through the use of three methods:

- *executeQuery()* for SELECT instructions
- *executeUpdate()* for updating the data or the database structure
- *execute()* can be used for both cases above when the result is unknown

4.1.1 execute() method

Let's use the *execute()* method to add a *employees* table to DBMaker database:

```
String tableSql = "CREATE TABLE employees"
                          + "(empID serial PRIMARY KEY, "
                          + "empName varchar(50), "
                          + "empPosition varchar(50),"
                          + "salary double, "
                          + "joinDate timestamp)";
stmt.execute(tableSql);
```

4.1.2 executeUpdate() method

Next, let's add a record to our table using the *executeUpdate()* method:

```
String insertSql = "INSERT INTO employees(empName,
empPosition, salary,joinDate)"
          + " VALUES('john', 'developer', 8000, now())";
stmt.executeUpdate(insertSql);
```

4.1.3 executeQuery() method

We can retrieve the records from the table using the *executeQuery()* method which returns an object of type *ResultSet*:

```
String selectSql = "SELECT * FROM employees";
try (ResultSet resultSet = stmt.executeQuery(selectSql)) {
    // use resultSet here
}
```

4.1.4 Closing the ResultSet and Statement Objects

You must explicitly close the ResultSet and Statement objects after you finish using them.

```
resultSet.close();
stmt.close();
```

Note: Typically, you should put close statements in a finally clause

4.2. PreparedStatement

PreparedStatement objects contain precompiled SQL sequences. They can have one or more parameters denoted by a question mark.

Let's create a *PreparedStatement* which updates records in the *employees* table based on given parameters:

```
String updateEmployeeSql = "UPDATE employees SET
salary=? WHERE empID=?";
try (PreparedStatement pstmt =
conn.prepareStatement(updateEmployeeSql)) {
    // use pstmt here
}
```

To add parameters to the *PreparedStatement*, we can use simple setters – *setX()*

```
pstmt.setDouble(1,10000);

pstmt.setInt(2, 1);
```

4.3. CallableStatement

The CallableStatement interface allows calling stored procedures.

To create a CallableStatement object, we can use the prepareCall() method of Connection:

```
String preparedSql = "{call insertEmployee(?,?,?,?,?)}";

try (CallableStatement cstmt = conn.prepareCall(preparedSql))

{

    // use cstmt here

}
```

Setting input parameter values for the stored procedure is done like in the PreparedStatement interface, using setX() methods:

```
cstmt.setString(2, "Alex");

cstmt.setString(3, "tester");

cstmt.setDouble(4, 5000);

java.util.Date today = new java.util.Date();

Timestamp todayDate = new Timestamp(today.getTime());

cstmt.setTimestamp(5, todayDate);
```

If the stored procedure has output parameters, we need to add them using the registerOutParameter() method:

```
cstmt.registerOutParameter(1, Types.INTEGER);
```

Then let's execute the statement and retrieve the returned value using a corresponding *getX()* method:

```
cstmt.execute();
int newId = cstmt.getInt(1);
```

For example to work, we need to create the stored procedure in DBMaker database:

```
CREATE PROCEDURE insertEmployee(OUT empId int,
                                IN empName varchar(50) ,
                                IN empPosition varchar(50),
                                IN salary double,
                                IN joinDate timestamp)
LANGUAGE SQL
BEGIN
    INSERT INTO
employees(empName,empPosition,salary,joinDate) VALUES
(empName, empPosition ,salary, joinDate);
```

```
        SET empId = select last_serial from sysconinfo;

END;
```

5. Parsing Query Results

After executing a query, the result is represented by a ResultSet object, which has a structure similar to a table, with lines and columns.

Let's first create an *Employee* class to store our retrieved records:

```
Public class Employee {
        private int id;
        private String name;
        private String position;
        private double salary;
        private timestamp joinDate;


        // standard constructor, getters, setters
}
```

Next, let's traverse the *ResultSet* and create an *Employee* object for each record:

```
String selectSql = "SELECT * FROM employees";
try (ResultSet rs = stmt.executeQuery(selectSql)) {
        List<Employee> employees = new ArrayList<>();
        while (rs.next()) {
            Employee emp = new Employee();
            emp.setId(rs.getInt("empId"));
            emp.setName(rs.getString("empName"));
            emp.setPosition(rs.getString("empPosition"));
            emp.setSalary(rs.getDouble("salary"));
            emp.setJoinDate(rs.getTimeStamp("joinDate"));
            employees.add(emp);
      }
}
```

6. Parsing Metadata

The JDBC API allows looking up information about the database, called metadata.

6.1 DatabaseMetadata

The DatabaseMetadata interface can be used to obtain general information about the database such as the tables, stored procedures, or SQL dialect.

```
DatabaseMetaData dbmd = conn.getMetaData();
ResultSet tablesRs = dbmd.getTables(null, "SYSADM", "%", null);
while (tablesRs.next()) {
	System.out.println(tablesRs.getString("TABLE_NAME"));
}
```

6.2 ResultSetMetadata

This interface can be used to find information about a certain ResultSet, such as the number and name of its columns:

```
ResultSetMetaData rsmd = rs.getMetaData();
int numColumns = rsmd.getColumnCount();

IntStream.range(1, numColumns+1).forEach(i -> {
    try {
        System.out.println(rsmd.getColumnName(i));
    } catch (SQLException e) {
        e.printStackTrace();
    }
});
```

7. Closing the Resources

You can close the connection by using the close method of the Connection object, as follows:

```
conn.close();
```

Note:Typically, you should put close statements in a finally clause.

8. Connecting with DataSource Objects

Data sources are the preferred mechanism by which to create JDBC connections in a Java Platform, Enterprise Edition (Java EE) environment. Data sources provide connections,pooled connections, and distributed connections without hard-coding connection properties into Java code.

8.1 DataSource

Import packages

```
import dbmaker.sql.type4.xa.ConnectionPoolDataSource;
```

Creating the connection

```
ConnectionPoolDataSource ds = null;


ds = new ConnectionPoolDataSource();
ds.setServerName("127.0.0.1");
ds.setPortNumber(2453);
ds.setDatabaseName("dbsample5");
ds.setUser("sysadm");
ds.setPassword("");
```

```
Connection conn = ds.getConnection();
```

8.2 XADataSource

Import packages

```
import dbmaker.sql.type4.xa.XADataSource;

import javax.sql.XAConnection;
```

Creating the connection

```
XADataSource ds = new XADataSource();

ds.setServerName("127.0.0.1");

ds.setPortNumber(2453);

ds.setDatabaseName("dbsample5");

ds.setUser("sysadm");

ds.setPassword("");


XAConnection xaCon = ds.getXAConnection();

Connection conn = xaCon.getConnection();
```

9. Conclusion

In this tutorial, we had a look at the basics of working with the DBMaker Type 4 JDBC API.

10. The full source code of the examples

TableName: Employees

```
CREATE TABLE employees(empID serial primary key,
                        empName varchar(50),
                        empPosition varchar(50),
                        salary double,
                        joinDate timestamp);
```

Stored Procedure: insertEmployee.sp

```
CREATE OR REPLACE PROCEDURE insertEmployee(
        OUT empId int,
        IN empName varchar(50) ,
        IN empPosition varchar(50),
        IN salary double,
        IN joinDate timestamp)
LANGUAGE SQL
BEGIN
    INSERT INTO employees(empName,empPosition,salary,joinDate)
VALUES (empname, empPosition ,salary, joinDate);
```

```
     SET empId = select last_serial from sysconinfo;
END;
```

DBMakerType4DriverDemo.java

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.IntStream;

public class DBMakerType4DriverDemo {

    private final static String dbDriver = "dbmaker.sql.type4.Driver";
    private final static String dburl =
"jdbc:dbmaker:type4://127.0.0.1:2453/DBSAMPLE5";
    private final static String dbUser = "sysadm";
    private final static String dbPassword = "";

    private static Connection conn = null;

    public static void main(String[] args)   {

        try {
            //3.2 Registering the Driver
            Class.forName(dbDriver);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        try   {
            //3.3 Creating the Connection
            conn = DriverManager.getConnection(dburl, dbUser,
dbPassword);

            //4.1 Creating a Statement Object
            stateMentTest();

            //4.2. PreparedStatement
            preparedStatementTest();

            //4.3. CallableStatement
            callableStatementTest();

            //6. Parsing Metadata
            metaDataTest();

            //5. Parsing Query Results
```

```
                parseResult();

        }catch(Exception sqle) {
                sqle.printStackTrace();
        }finally {
                //7. Closing the Resources
                try {
                        if(conn!=null) conn.close();
                } catch(SQLException se) {
                        se.printStackTrace();
                }
        }
}

public static void stateMentTest() {
        Statement stmt = null;
        ResultSet resultSet = null;

        try {
                //4.1 Creating a Statement Object
                stmt = conn.createStatement();

                //4.1.1 execute() method
                String tableSql = "CREATE TABLE employees"
                        + "(empID serial PRIMARY KEY, "
                        + "empName varchar(50), "
                        + "empPosition varchar(50),"
                        + "salary double, "
                        + "joinDate timestamp)";
                stmt.execute(tableSql);

                //4.1.2 executeUpdate() method
                String insertSql = "INSERT INTO employees(empName,
        empPosition, salary,joinDate)"
                        + " VALUES('John', 'developer', 8000, now())";
                stmt.executeUpdate(insertSql);

                //4.1.3 executeQuery() method
                String selectSql = "SELECT empName, empPosition,
        salary,joinDate FROM employees";
                resultSet = stmt.executeQuery(selectSql);

                while(resultSet.next()) { System.out.println("Employee
        Name is " + resultSet.getString(1)); }
```

```

        } catch (SQLException e) {
            e.printStackTrace();
        }finally {
            try {
                if (resultSet != null) resultSet.close();
                if(stmt != null) stmt.close();
            } catch (SQLException sqle) {
                sqle.printStackTrace();
            }
        }
    }

    //4.2. PreparedStatement
    public static void preparedStatementTest() {
        String updateEmployeeSql = "UPDATE employees SET salary=?
WHERE empID=?";
        try {
            PreparedStatement pstmt =
conn.prepareStatement(updateEmployeeSql);
            pstmt.setDouble(1,10000);
            pstmt.setInt(2,1);
```

```
                pstmt.executeUpdate();

                pstmt.close();
            }catch (SQLException sqle) {
                sqle.printStackTrace();
            }
        }

        //4.3. CallableStatement
        public static void callableStatementTest() {
            String preparedSql = "{call insertEmployee(?,?,?,?,?)}";
            try (CallableStatement cstmt = conn.prepareCall(preparedSql)) {
                cstmt.setString(2, "Alex");
                cstmt.setString(3, "tester");
                cstmt.setDouble(4, 15000);
                final java.util.Date today = new java.util.Date();
                final java.sql.Timestamp todayDate = new
java.sql.Timestamp(today.getTime());
                cstmt.setTimestamp(5,todayDate );

                cstmt.registerOutParameter(1, Types.INTEGER);

                cstmt.execute();
                int newId = cstmt.getInt(1);
                System.out.println("New EmpID is " + newId);

            } catch (SQLException sqle) {
                sqle.printStackTrace();
            }
        }

        //5. Parsing Query Results
        public static void parseResult() {

            Statement stmt = null;
            ResultSet rs = null;
            String selectSql = "SELECT * FROM employees";
            try {
                stmt = conn.createStatement();
                rs = stmt.executeQuery(selectSql);
                List<Employee> employees = new ArrayList<>();
                while (rs.next()) {
                    Employee emp = new Employee();
                    emp.setId(rs.getInt("empId"));
                    emp.setName(rs.getString("empName"));
                    emp.setPosition(rs.getString("empPosition"));
                    emp.setSalary(rs.getDouble("salary"));
                    emp.setJoinDate(rs.getTimestamp("joinDate"));
                    employees.add(emp);
                }
            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }finally {
                try {
                    if (rs != null) rs.close();
                    if(stmt != null) stmt.close();
                } catch (SQLException sqle) {
                    sqle.printStackTrace();
                }
            }
        }

        //6. Parsing Metadata
        public static void metaDataTest() {

            Statement stmt = null;
            try {
                stmt = conn.createStatement();
            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }

            //6.1. DatabaseMetadata
            DatabaseMetaData dbmd = null;
            ResultSet tablesResultSet = null;
            try {
                dbmd = conn.getMetaData();

                if(dbmd != null) {
                    // get table details
                    tablesResultSet = dbmd.getTables(null, "SYSADM",
    "%", null);
                }
                System.out.println("Table list:");
                while (tablesResultSet.next()) {
                    System.out.println("\t" +
    tablesResultSet.getString("TABLE_NAME"));
                }

                tablesResultSet.close();
            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }

            //6.2. ResultSetMetadata
            ResultSet rs = null;
            try {
                rs = stmt.executeQuery("SELECT * FROM EMPLOYEES");
                ResultSetMetaData rsmd = rs.getMetaData();
                int numColumns = 0;
                try {
                    numColumns = rsmd.getColumnCount();
                } catch (SQLException throwables) {
                    throwables.printStackTrace();
                }
                System.out.println("Table Employees Column:");
                IntStream.range(1, numColumns+1).forEach(i -> {
                    try {
                        System.out.println("\t" + rsmd.getColumnName(i));
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                });
            } catch (SQLException sqle) {
                sqle.printStackTrace();
            }finally {
                try {
                    if(rs!= null) rs.close();
                    if(stmt!=null) stmt.close();
                } catch (SQLException sqle) {
                    sqle.printStackTrace();
                }
            }
        }
    }

    class Employee {
        private int id;
        private String name;
        private String position;
        private double salary;
        private Timestamp joinDate;

        // standard constructor, getters, setters
        public void setId(int id) {this.id = id;}
        public int getId(){return this.id;}

        public void setPosition(String position) {this.position = position;}
        public String getPosition(){return this.position;}

        public void setName(String name) {this.name = name;}
        public String getName(){return this.name;}

        public void setSalary(double salary) {this.salary = salary;}
        public double getSalary(){return this.salary;}

        public void setJoinDate(Timestamp joinDate) {this.joinDate =
    joinDate;}
        public Timestamp getJoinDate(){return this.joinDate;}
    }
```

TestDataSource.java

```
/**
 *   8.1 DataSource Sample demonstration
 */
import dbmaker.sql.type4.xa.ConnectionPoolDataSource;
import java.sql.*;

public class TestDataSource {
      public static void main(String[] args) throws SQLException {

            ConnectionPoolDataSource ds = null;

            ds = new ConnectionPoolDataSource();
            ds.setServerName("127.0.0.1");
            ds.setPortNumber(2453);
            ds.setDatabaseName("dbsample5");
            ds.setUser("sysadm");
            ds.setPassword("");

            Connection con = ds.getConnection();
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("select table_Owner,
table_Name from systable");


            while(rs.next()){

System.out.println("tableOwner="+rs.getString("table_Owner").trim()
+",tableName="+rs.getString("table_Name").trim());
            }


            rs.close();
            stmt.close();
            con.close();
      }
}
```

TestXADataSource.java

```
/**
  * 8.2 XADataSource Sample demonstration
  */
import dbmaker.sql.type4.xa.XADataSource;


import java.sql.*;
import javax.sql.XAConnection;


public class TestXADataSource {
      public static void main(String[] args) throws SQLException {


            XADataSource ds = new XADataSource();
            ds.setServerName("127.0.0.1");
            ds.setPortNumber(2453);
            ds.setDatabaseName("dbsample5");
            ds.setUser("sysadm");
            ds.setPassword("");


            XAConnection xaCon = ds.getXAConnection();
            Connection conn = xaCon.getConnection();
            Statement stmt = conn.createStatement();

            ResultSet rs = stmt.executeQuery("select table_Owner,
table_Name from systable");

            while(rs.next()){

System.out.println("tableOwner="+rs.getString("table_Owner").trim()
+",tableName="+rs.getString("table_Name").trim());
            }

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