

## Writing JDBC Applications for MySQL with NuSphere Advantage

by Paul DuBois – December, 2001

### TABLE OF CONTENTS

Preliminary Requirements

Connecting to the MySQL Server

Issuing Queries

Issuing Queries that Return NO Result Set

Issuing Queries that Return a Result Set

Error Handling

Links

About NuSphere

NuSphere products provide the tools you need to write MySQL applications in a variety of languages. The languages that most people use with are PHP and Perl, but a sometimes-overlooked component of these products are the MM.MySQL driver, which allows you to develop Java applications that interact with your MySQL server.

MM.MySQL works within the framework of the Java Database Connectivity (JDBC) API, an interface that allows Java programs to use database servers in a portable way. JDBC is based on an approach similar to that used in the design of Perl's DBI module, Python's DB-API module, and (more recently) PHP's PEAR::DB class. This approach uses a two-tier architecture:

- The top level is visible to application programs and presents an abstract interface for connecting to and using database engines. The application interface does not depend on details specific to particular engines.
- The lower level consists of drivers for individual database engines. Each driver handles the details necessary to map the abstract application interface onto operations that a specific engine will understand.

The JDBC interface allows developers to write applications that can be used with different databases with a minimum of porting effort. Once a driver for a given server engine is installed, JDBC applications can communicate with any server of that type. By using MM.MySQL, your Java programs can access MySQL databases.

## Preliminary Requirements

To use Java applications with MySQL, you may need to install some additional software:

- If you want to compile and run Java programs, you'll need a Java compiler (such as `javac` or `jikes`) and a runtime environment. If these are not already installed on your system, you can get them by obtaining a Java Software Development Kit (SDK) from `java.sun.com`.
- If you want only to run precompiled applications, no compiler is necessary, but you'll still need a Java Runtime Environment (JRE). This too may be obtained from `java.sun.com`.

This article assumes that you'll write and compile your own programs, and thus that you have a Java SDK installed. Once you compile a Java program, however, you can deploy it to other machines, even ones that have only a runtime environment. This works even in heterogenous installations, because Java is platform-independent. Applications compiled on one platform can be expected to work on other platforms. For example, you can develop on a Linux box and deploy on Windows.

## Connecting to the MySQL Server

To connect to the MySQL server, register the JDBC driver you plan to use, then invoke its `getConnection()` method. The following short program, `Connect.java`, shows how to connect to and disconnect from a server running on the local host. It accesses a database named `test`, using a MySQL account with a user name and password of `testuser` and `testpass`:

```
import java.sql.*;
public class Connect
{
    public static void main (String[] args)
    {
        Connection conn = null;
        try
        {
```

```
String userName = "testuser";
String password = "testpass";
String url = "jdbc:mysql://localhost/test";
Class.forName ("org.gjt.mm.mysql.Driver").newInstance ();
conn = DriverManager.getConnection (url, userName, password);
System.out.println ("Database connection established");
}
catch (Exception e)
{
    System.err.println ("Cannot connect to database server");
}
finally
{
    try
    {
        if (conn != null)
        {
            conn.close ();
            System.out.println ("Database connection terminated");
        }
    }
    catch (Exception e) { /* ignore close errors */ }
}
}
```

Compile `Connect.java` to produce a class file `Connect.class` that contains executable Java code:

```
% javac Connect.java
```

Then invoke the class file as follows and it should connect to and disconnect from your MySQL server:

```
% java Connect
Database connection established
Database connection terminated
```

If you have trouble compiling `Connect.java`, double check that you have a Java Software Development Kit installed and make sure that the MM.MySQL driver is listed in your `CLASSPATH` environment variable.

The arguments to `getConnection()` are the connection URL and the user name and password of a MySQL account. As illustrated by `Connect.java`, JDBC URLs for MySQL consist of `jdbc:mysql://` followed by the name of the MySQL server host and the database name. An alternate syntax for specifying the user and password is to add them as parameters to the end of the connection URL:

```
jdbc:mysql://localhost/test?user=testuser&password=testpass
```

When you specify a URL using this second format, `getConnection()` requires only one argument. For example, the code for connecting to the MySQL server in `Connect.java` could have been written like this:

```
String userName = "testuser";
String password = "testpass";
String url = "jdbc:mysql://localhost/test?user="
            + userName
            + "&password="
            + password;
Class.forName ("org.gjt.mm.mysql.Driver").newInstance ();
conn = DriverManager.getConnection (url);
```

`getConnection()` returns a `Connection` object that may be used to interact with MySQL by issuing queries and retrieving their results. (The next section describes how to do this.) When you're done with the connection, invoke its `close()` method to disconnect from the MySQL server.

To increase the portability of your applications, you can store the connection parameters (host, database, user name, and password) in a Java properties file and read the properties at runtime. Then they need not be listed in the program itself. This allows you to change the server to which the program connects by editing the properties file, rather than by having to recompile the program.

## Issuing Queries

To process SQL statements in a JDBC-based application, create a `Statement` object from your `Connection` object. `Statement` objects support an `executeUpdate()` method for issuing queries that modify the database and return no result set, and an `executeQuery()` method for queries that do return a result set. The query-processing examples in this article use the following table, `animal`, which contains an integer `id` column and two string columns, `name` and `species`:

```
CREATE TABLE animal
(
  id          INT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (id),
  name       CHAR(40),
  category   CHAR(40)
)
```

`id` is an `AUTO_INCREMENT` column, so MySQL automatically assigns successive values 1, 2, 3, ... as records are added to the table.

## Issuing Queries That Return No Result Set

The following example obtains a `Statement` object from the `Connection` object, then uses it to create and populate the `animal` table. `CREATE TABLE` and `INSERT` both are statements that modify the database, so

`executeUpdate()` is the appropriate method for issuing them:

```
Statement s = conn.createStatement ();
int count;
s.executeUpdate (
    "CREATE TABLE animal ("
    + "id INT UNSIGNED NOT NULL AUTO_INCREMENT,"
    + "PRIMARY KEY (id),"
    + "name CHAR(40), category CHAR(40))");
count = s.executeUpdate (
    "INSERT INTO animal (name, category)"
    + " VALUES"
    + "('snake', 'reptile'),"
    + "('frog', 'amphibian'),"
    + "('tuna', 'fish'),"
    + "('raccoon', 'mammal')");
s.close ();
System.out.println (count + " rows were inserted");
```

The `executeUpdate()` method returns the number of rows affected by a query. As shown above, the `count` is used to report how many rows the `INSERT` statement added to the `animal` table.

A `Statement` object may be used to issue several queries. When you're done with it, invoke its `close()` method to dispose of the object and free any resources associated with it.

## Issuing Queries That Return a Result Set

For statements such as `SELECT` queries that retrieve information from the database, use `executeQuery()`. After calling this method, create a `ResultSet` object and use it to iterate through the rows re

turned by your query. The following example shows one way to retrieve the contents of the `animal` table:

```
Statement s = conn.createStatement ();
s.executeQuery ("SELECT id, name, category FROM animal");
ResultSet rs = s.getResultSet ();
int count = 0;
while (rs.next ())
{
    int idVal = rs.getInt ("id");
    String nameVal = rs.getString ("name");
    String catVal = rs.getString ("category");
    System.out.println (
        "id = " + idVal
        + ", name = " + nameVal
        + ", category = " + catVal);
    ++count;
}
rs.close ();
s.close ();
System.out.println (count + " rows were retrieved");
```

`executeQuery()` does not return a row count, so if you want to know how many rows a result set contains, you should count them yourself as you fetch them.

To obtain the column values from each row, invoke `getXXX()` methods that match the column data types. The `getInt()` and `getString()` methods used in the preceding example return integer and string values. As the example shows, these methods may be called using the name of a result set column. You can also fetch values by position. For the result set retrieved by the `SELECT` query in the example,

`id`, `name`, and `category` are at column positions 1, 2 and 3 and thus could have been obtained like this:

```
int idVal = rs.getInt (1);
String nameVal = rs.getString (2);
String catVal = rs.getString (3);
```

`ResultSet` objects, like `Statement` objects, should be closed when you're done with them.

To check whether or not a column value is `NULL`, invoke the result set object's `wasNull()` method after fetching the value. For example, you could check for a `NULL` value in the `name` column like this:

```
String nameVal = rs.getString ("name");
if (rs.wasNull ())
    nameVal = "(no name available)";
```

## Using Placeholders

Sometimes it's necessary to construct queries from values containing characters that require special treatment. For example, in queries, string values are written enclosed within quotes, but any quote characters in the string itself should be escaped with a backslash to avoid creating malformed SQL. In this case, it's much easier to let JDBC handle the escaping for you, rather than fooling around trying to do so yourself. To use this approach, create a different kind of statement (a `PreparedStatement`), and refer to the data values in the query string by means of placeholder characters. Then tell JDBC to bind the data values to the placeholders and it will handle any special characters automatically.

Suppose you have two variables `nameVal` and `catVal` from which you want to create a new record in the `animal` table. To do so without regard to whether or not the values contain special characters, issue the query like this:

```
PreparedStatement s;
s = conn.prepareStatement (
    "INSERT INTO animal (name, category)"
    + " VALUES (?,?)");
s.setString (1, nameVal);
```

```
s.setString (2, catVal);
int count = s.executeUpdate ();
s.close ();
System.out.println (count + " rows were inserted");
```

The ``?` characters in the query string act as placeholders—special markers indicating where data values should be placed. The `setString()` method takes a placeholder position and a string value and binds the value to the appropriate placeholder, performing any special-character escaping that may be necessary. The method you use to bind a value depends on the data type. For example, `setString()` binds string values and `setInt()` binds integer values.

## Error Handling

If you want to trap errors, execute your JDBC operations within a `try` block and use an exception handler to display information about the cause of any problems that occur. JDBC provides `getMessage()` and `getErrorCode()` methods that may be invoked when an exception occurs to obtain the error message and the numeric error code. The following example deliberately issues a malformed query. When it runs, the `executeQuery()` method fails and raises an exception, which is handled in the `catch` block:

```
try
{
    Statement s = conn.createStatement ();
    s.executeQuery ("XYZ"); // issue invalid query
    s.close ();
}
catch (SQLException e)
{
    System.err.println ("Error message: " + e.getMessage ());
    System.err.println ("Error number: " + e.getErrorCode ());
}
```

## Links

The following sites provide information about the tools discussed in this article:

- Sun's Java site is a clearinghouse for all kinds of Java-related information:

<http://java.sun.com/>

You can obtain the Java Software Development Kit or Java Runtime Environment here. The JDBC API specification is also available on this site, should you wish to read more about it.

- MM.MySQL is included with NuSphere Advantage products, but its author Mark Matthews maintains a site devoted to providing information specifically about the driver:

<http://mmmmysql.sourceforge.net/>

## About NuSphere Corporation

NuSphere delivers the first Internet Application Platform (IAP) based on open source components, providing an integrated foundation that allows companies to deploy reliable, cost-effective, enterprise-class applications across Windows, UNIX and Linux environments. NuSphere® Advantage is an integrated software suite that pairs the reliability and cost-effectiveness of PHP, Apache, Perl and open source databases with new technology for building business-critical web applications and web services. Based in Bedford, Mass., the company's commercial software services include technical support, consulting and training. For more information, visit [www.nusphere.com](http://www.nusphere.com) or call +1-781-280-4600.

NuSphere is a registered trademark in Australia, Norway, Hong Kong, Switzerland, and the European Community; NuSphere and PHPed are trademarks of NuSphere Corporation in the U.S. and other countries. Any other trademarks or service marks contained herein are the property of their respective owners.

MySQL AB distributes the MySQL database pursuant to the applicable GNU General Public License that is available as of the date of this publication at <http://www.fsf.org/licenses/gpl.txt> and all of the terms and disclaimers contained therein. NuSphere Corporation is not affiliated with MySQL AB. The products and services of NuSphere Corporation are not sponsored or endorsed by MySQL AB.