1. MySQL Connector/J

MySQL provides connectivity for client applications developed in the Java programming language via a JDBC driver, which is called MySQL Connector/J.

MySQL Connector/J is a JDBC-3.0 “Type 4” driver, which means that is pure Java, implements version 3.0 of the JDBC specification, and communicates directly with the MySQL server using the MySQL protocol.

This document is arranged for a beginning JDBC developer. If you are already experienced with using JDBC, you might consider starting with the Section 1.2, “Installing Connector/J”.

Although JDBC is useful by itself, we would hope that if you are not familiar with JDBC that after reading the first few sections of this manual, that you would avoid using “naked” JDBC for all but the most trivial problems and consider using one of the popular persistence frameworks such as Hibernate [http://www.hibernate.org/], Spring's JDBC templates [http://www.springframework.org/] or Ibatis SQL Maps [http://ibatis.apache.org/] to do the majority of repetitive work and heavier lifting that is sometimes required with JDBC.

This section is not designed to be a complete JDBC tutorial. If you need more information about using JDBC you might be interested in the following online tutorials that are more in-depth than the information presented here:

- JDBC Basics [http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html] — A tutorial from Sun covering beginner topics in JDBC
- JDBC Short Course [http://java.sun.com/developer/onlineTraining/Database/JDBCShortCourse/index.html] — A more in-depth tutorial from Sun and JGuru

1.1. Basic JDBC concepts

This section provides some general JDBC background.

1.1.1. Connecting to MySQL using the `DriverManager` Interface

When you are using JDBC outside of an application server, the `DriverManager` class manages the establishment of Connections.

The `DriverManager` needs to be told which JDBC drivers it should try to make Connections with. The easiest way to do this is to use `Class.forName()` on the class that implements the `java.sql.Driver` interface. With MySQL Connector/J, the name of this class is `com.mysql.jdbc.Driver`. With this method, you could use an external configuration file to supply the driver class name and driver parameters to use when connecting to a database.

The following section of Java code shows how you might register MySQL Connector/J from the `main()` method of your application:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

// Notice, do not import com.mysql.jdbc.*
// or you will have problems!
```

```
public class LoadDriver {
    public static void main(String[] args) {
        try {
            // The newInstance() call is a work around for some
            // broken Java implementations

            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception ex) {
            // handle the error
        }
}
```

After the driver has been registered with the `DriverManager`, you can obtain a `Connection` instance that is connected to a particular database by calling `DriverManager.getConnection()`:

Example 1. Obtaining a Connection From the `DriverManager`

This example shows how you can obtain a `Connection` instance from the `DriverManager`. There are a few different signatures for the `getConnection()` method. You should see the API documentation that comes with your JDK for more specific information on how to use them.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

    ... try {
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/

            // Do something with the Connection

           ....
        } catch (SQLException ex) {
            // handle any errors
            System.out.println("SQLException: " + ex.getMessage());
            System.out.println("SQLState: " + ex.getSQLState());
            System.out.println("VendorError: " + ex.getErrorCode());
        }
```

Once a `Connection` is established, it can be used to create `Statement` and `PreparedStatement` objects, as well as retrieve metadata about the database. This is explained in the following sections.

1.1.2. Using Statements to Execute SQL

`Statement` objects allow you to execute basic SQL queries and retrieve the results through the `ResultSet` class which is described later.

To create a `Statement` instance, you call the `createStatement()` method on the `Connection` object you have retrieved via one of the `DriverManager.getConnection()` or `DataSource.getConnection()` methods described earlier.

Once you have a `Statement` instance, you can execute a `SELECT` query by calling the `executeQuery(String)` method with the SQL you want to use.

To update data in the database, use the `executeUpdate(String SQL)` method. This method returns the number of rows affected by the update statement.

If you don't know ahead of time whether the SQL statement will be a `SELECT` or an `UPDATE`/`INSERT`, then you can use the `execute(String SQL)` method. This method will return true if the SQL query was a `SELECT`, or false if it was an `UPDATE`, `INSERT`, or `DELETE` statement. If the statement was a `SELECT` query, you can retrieve the results by calling the `getResultSet()` method. If the statement was an `UPDATE`, `INSERT`, or `DELETE` statement, you can retrieve the affected rows count by calling `getUpdateCount()` on the `Statement` instance.

Example 2. Using java.sql.Statement to Execute a SELECT Query

```
// assume that conn is an already created JDBC connection
Statement stmt = null;
ResultSet rs = null;

try {
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT foo FROM bar");

    // or alternatively, if you don't know ahead of time that
    // the query will be a SELECT...

    if (stmt.execute("SELECT foo FROM bar")) {
        rs = stmt.getResultSet();
    }

    // Now do something with the ResultSet ....
} finally {
    // it is a good idea to release
    // resources in a finally{} block
    // in reverse-order of their creation
    // if they are no-longer needed

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException sqlEx) { // ignore }

        rs = null;
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) { // ignore }

        stmt = null;
    }
}
```

1.1.3. Using `CallableStatements` to Execute Stored Procedures

Starting with MySQL server version 5.0 when used with Connector/J 3.1.1 or newer, the `java.sql.CallableStatement` interface is fully implemented with the exception of the `getParameterMetaData()` method.

MySQL's stored procedure syntax is documented in the "Stored Procedures and Functions [http://www.mysql.com/doc/en/stored-procedures.html]" section of the MySQL Reference Manual.

Connector/J exposes stored procedure functionality through JDBC's `CallableStatement` interface.

The following example shows a stored procedure that returns the value of `inOutParam` incremented by 1, and the string passed in via `inputParam` as a `ResultSet`:

Example 3. Stored Procedure Example

```
CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), INOUT inOutParam INT)
BEGIN
    DECLARE z INT;
    SET z = inOutParam + 1;
    SET inOutParam = z;

    SELECT inputParam;

    SELECT CONCAT('zyxw', inputParam);
END
```

To use the `demoSp` procedure with Connector/J, follow these steps:

1. Prepare the callable statement by using `Connection.prepareCall()`.

   Notice that you have to use JDBC escape syntax, and that the parentheses surrounding the parameter placeholders are not optional:

   #### Example 4. Using `Connection.prepareCall()`

   ```
   import java.sql.CallableStatement;

   ...

       //
       // Prepare a call to the stored procedure 'demoSp'
       // with two parameters
       //
       // Notice the use of JDBC-escape syntax ({call ...})
       //

       CallableStatement cStmt = conn.prepareCall("{call demoSp(?, ?)}");


       cStmt.setString(1, "abcdefg");
   ```

   ## Note

   `Connection.prepareCall()` is an expensive method, due to the metadata retrieval that the driver performs to support output parameters. For performance reasons, you should try to minimize unnecessary calls to `Connection.prepareCall()` by reusing `CallableStatement` instances in your code.

2. Register the output parameters (if any exist)

   To retrieve the values of output parameters (parameters specified as `OUT` or `INOUT` when you cre-

ated the stored procedure), JDBC requires that they be specified before statement execution using the various `registerOutputParameter()` methods in the `CallableStatement` interface:

Example 5. Registering Output Parameters

```
import java.sql.Types;
...
//
// Connector/J supports both named and indexed
// output parameters. You can register output
// parameters using either method, as well
// as retrieve output parameters using either
// method, regardless of what method was
// used to register them.
//
// The following examples show how to use
// the various methods of registering
// output parameters (you should of course
// use only one registration per parameter).
//

//
// Registers the second parameter as output, and
// uses the type 'INTEGER' for values returned from
// getObject()
//

cStmt.registerOutParameter(2, Types.INTEGER);

//
// Registers the named parameter 'inOutParam', and
// uses the type 'INTEGER' for values returned from
// getObject()
//

cStmt.registerOutParameter("inOutParam", Types.INTEGER);
...
```

3. Set the input parameters (if any exist)

   Input and in/out parameters are set as for `PreparedStatement` objects. However, `CallableStatement` also supports setting parameters by name:

Example 6. Setting `CallableStatement` Input Parameters

```
...

    //
    // Set a parameter by index
    //

    cStmt.setString(1, "abcdefg");

    //
    // Alternatively, set a parameter using
    // the parameter name
```

```
        //

        cStmt.setString("inputParameter", "abcdefg");

        //
        // Set the 'in/out' parameter using an index
        //

        cStmt.setInt(2, 1);

        //
        // Alternatively, set the 'in/out' parameter
        // by name
        //

        cStmt.setInt("inOutParam", 1);

    ...
```

4. Execute the `CallableStatement`, and retrieve any result sets or output parameters.

   Although `CallableStatement` supports calling any of the `Statement` execute methods (`executeUpdate()`, `executeQuery()` or `execute()`), the most flexible method to call is `execute()`, as you do not need to know ahead of time if the stored procedure returns result sets:

Example 7. Retrieving Results and Output Parameter Values

```
    ...

        boolean hadResults = cStmt.execute();

        //
        // Process all returned result sets
        //

        while (hadResults) {
            ResultSet rs = cStmt.getResultSet();

            // process result set
            ...

            hadResults = cStmt.getMoreResults();
        }

        //
        // Retrieve output parameters
        //
        // Connector/J supports both index-based and
        // name-based retrieval
        //

        int outputValue = cStmt.getInt(2); // index-based

        outputValue = cStmt.getInt("inOutParam"); // name-based

    ...
```

1.1.4. Retrieving AUTO_INCREMENT Column Values

Before version 3.0 of the JDBC API, there was no standard way of retrieving key values from databases that supported “auto increment” or identity columns. With older JDBC drivers for MySQL, you could always use a MySQL-specific method on the `Statement` interface, or issue the query `SELECT LAST_INSERT_ID()` after issuing an `INSERT` to a table that had an `AUTO_INCREMENT` key. Using the MySQL-specific method call isn't portable, and issuing a `SELECT` to get the `AUTO_INCREMENT` key's value requires another round-trip to the database, which isn't as efficient as possible. The following code snippets demonstrate the three different ways to retrieve `AUTO_INCREMENT` values. First, we demonstrate the use of the new JDBC-3.0 method `getGeneratedKeys()` which is now the preferred method to use if you need to retrieve `AUTO_INCREMENT` keys and have access to JDBC-3.0. The second example shows how you can retrieve the same value using a standard `SELECT LAST_INSERT_ID()` query. The final example shows how updatable result sets can retrieve the `AUTO_INCREMENT` value when using the `insertRow()` method.

Example 8. Retrieving `AUTO_INCREMENT` Column Values using `Statement.getGeneratedKeys()`

```
Statement stmt = null;
ResultSet rs = null;

try {

 //
 // Create a Statement instance that we can use for
 // 'normal' result sets assuming you have a
 // Connection 'conn' to a MySQL database already
 // available

 stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                             java.sql.ResultSet.CONCUR_UPDATABLE);

 //
 // Issue the DDL queries for the table for this example
 //

 stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
 stmt.executeUpdate(
         "CREATE TABLE autoIncTutorial ("
         + "priKey INT NOT NULL AUTO_INCREMENT, "
         + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

 //
 // Insert one row that will generate an AUTO INCREMENT
 // key in the 'priKey' field
 //

 stmt.executeUpdate(
         "INSERT INTO autoIncTutorial (dataField) "
         + "values ('Can I Get the Auto Increment Field?')",
         Statement.RETURN_GENERATED_KEYS);

 //
 // Example of using Statement.getGeneratedKeys()
 // to retrieve the value of an auto-increment
 // value
 //
```

```
    int autoIncKeyFromApi = -1;

    rs = stmt.getGeneratedKeys();

    if (rs.next()) {
        autoIncKeyFromApi = rs.getInt(1);
    } else {

        // throw an exception from here
    }

    rs.close();

    rs = null;

    System.out.println("Key returned from getGeneratedKeys():"
        + autoIncKeyFromApi);
} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
```

Example 9. Retrieving `AUTO_INCREMENT` Column Values using `SELECT LAST_INSERT_ID()`

```
   Statement stmt = null;
   ResultSet rs = null;

   try {

    //
    // Create a Statement instance that we can use for
    // 'normal' result sets.

    stmt = conn.createStatement();

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
            "CREATE TABLE autoIncTutorial ("
            + "priKey INT NOT NULL AUTO_INCREMENT, "
```

```
            + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Insert one row that will generate an AUTO INCREMENT
    // key in the 'priKey' field
    //

    stmt.executeUpdate(
            "INSERT INTO autoIncTutorial (dataField) "
            + "values ('Can I Get the Auto Increment Field?')");

    //
    // Use the MySQL LAST_INSERT_ID()
    // function to do the same thing as getGeneratedKeys()
    //

    int autoIncKeyFromFunc = -1;
    rs = stmt.executeQuery("SELECT LAST_INSERT_ID()");

    if (rs.next()) {
        autoIncKeyFromFunc = rs.getInt(1);
    } else {
        // throw an exception from here
    }

    rs.close();

    System.out.println("Key returned from " + "'SELECT LAST_INSERT_ID()': "
        + autoIncKeyFromFunc);

} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
```

Example 10. Retrieving `AUTO_INCREMENT` Column Values in `Updatable ResultSets`

```
   Statement stmt = null;
   ResultSet rs = null;

   try {

    //
```

```
    // Create a Statement instance that we can use for
    // 'normal' result sets as well as an 'updatable'
    // one, assuming you have a Connection 'conn' to
    // a MySQL database already available
    //

    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                                java.sql.ResultSet.CONCUR_UPDATABLE);

    //
    // Issue the DDL queries for the table for this example
    //

    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
            "CREATE TABLE autoIncTutorial ("
            + "priKey INT NOT NULL AUTO_INCREMENT, "
            + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Example of retrieving an AUTO INCREMENT key
    // from an updatable result set
    //

    rs = stmt.executeQuery("SELECT priKey, dataField "
       + "FROM autoIncTutorial");

    rs.moveToInsertRow();

    rs.updateString("dataField", "AUTO INCREMENT here?");
    rs.insertRow();

    //
    // the driver adds rows at the end
    //

    rs.last();

    //
    // We should now be on the row we just inserted
    //

    int autoIncKeyFromRS = rs.getInt("priKey");

    rs.close();

    rs = null;

    System.out.println("Key returned for inserted row: "
        + autoIncKeyFromRS);

} finally {

    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }

    if (stmt != null) {
        try {
            stmt.close();
```

```
        } catch (SQLException ex) {
            // ignore
        }
    }
}
```

When you run the preceding example code, you should get the following output: Key returned from `getGeneratedKeys()`: 1 Key returned from `SELECT LAST_INSERT_ID()`: 1 Key returned for inserted row: 2 You should be aware, that at times, it can be tricky to use the `SELECT LAST_INSERT_ID()` query, as that function's value is scoped to a connection. So, if some other query happens on the same connection, the value will be overwritten. On the other hand, the `getGeneratedKeys()` method is scoped by the `Statement` instance, so it can be used even if other queries happen on the same connection, but not on the same `Statement` instance.

1.2. Installing Connector/J

Use the following instructions to install Connector/J

1.2.1. Required Software Versions

1.2.1.1. Java Versions Supported

MySQL Connector/J supports Java-2 JVMs, including JDK-1.2.x, JDK-1.3.x, JDK-1.4.x and JDK-1.5.x, and requires JDK-1.4.x or newer to compile (but not run). MySQL Connector/J does not support JDK-1.1.x or JDK-1.0.x

Because of the implementation of `java.sql.Savepoint`, Connector/J 3.1.0 and newer will not run on JDKs older than 1.4 unless the class verifier is turned off (`-Xverify:none`), as the class verifier will try to load the class definition for `java.sql.Savepoint` even though it is not accessed by the driver unless you actually use savepoint functionality.

Caching functionality provided by Connector/J 3.1.0 or newer is also not available on JVMs older than 1.4.x, as it relies on `java.util.LinkedHashMap` which was first available in JDK-1.4.0.

1.2.1.2. MySQL Server Version Guidelines

MySQL Connector/J supports all known MySQL server versions. Some features (foreign keys, updatable result sets) require more recent versions of MySQL to operate.

When connecting to MySQL server version 4.1 or newer, it is best to use MySQL Connector/J version 3.1, as it has full support for features in the newer versions of the server, including Unicode characters, views, stored procedures and server-side prepared statements.

Although Connector/J version 3.0 will connect to MySQL server, version 4.1 or newer, and implements Unicode characters and the new authorization mechanism, Connector/J 3.0 will not be updated to support new features in current and future server versions.

1.2.1.3. Installing the Driver and Configuring the `CLASSPATH`

MySQL Connector/J is distributed as a .zip or .tar.gz archive containing the sources, the class files a class-file only “binary” .jar archive named "`mysql-connector-java-[version]-bin.jar`", and starting with Connector/J 3.1.8 a “debug” build of the driver in a file named "`mysql-connector-java-[version]-bin-g.jar`".

Starting with Connector/J 3.1.9, we don't ship the .class files “unbundled,” they are only available in the JAR archives that ship with the driver.

You should not use the “debug” build of the driver unless instructed to do so when reporting a problem or bug to MySQL AB, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the `src/lib/aspectjrt.jar` file that comes with the Connector/J distribution.

You will need to use the appropriate graphical or command-line utility to un-archive the distribution (for example, WinZip for the .zip archive, and `tar` for the .tar.gz archive). Because there are potentially long filenames in the distribution, we use the GNU tar archive format. You will need to use GNU tar (or an application that understands the GNU tar archive format) to unpack the .tar.gz variant of the distribution.

Once you have extracted the distribution archive, you can install the driver by placing mysql-connector-java-[version]-bin.jar in your classpath, either by adding the FULL path to it to your CLASSPATH environment variable, or by directly specifying it with the command line switch -cp when starting your JVM

If you are going to use the driver with the JDBC DriverManager, you would use "com.mysql.jdbc.Driver" as the class that implements java.sql.Driver.

Example 11. Setting the CLASSPATH Under UNIX

The following command works for 'csh' under UNIX:

```
$ setenv CLASSPATH /path/to/mysql-connector-java-[version]-bin.jar:$CLASSPATH
```

The above command can be added to the appropriate startup file for the login shell to make MySQL Connector/J available to all Java applications.

If you want to use MySQL Connector/J with an application server such as Tomcat or JBoss, you will have to read your vendor's documentation for more information on how to configure third-party class libraries, as most application servers ignore the CLASSPATH environment variable. For configuration examples for some J2EE application servers, see Section 1.4, “Using Connector/J with J2EE and Other Java Frameworks”. However, the authoritative source for JDBC connection pool configuration information for your particular application server is the documentation for that application server.

If you are developing servlets or JSPs, and your application server is J2EE-compliant, you can put the driver's .jar file in the WEB-INF/lib subdirectory of your webapp, as this is a standard location for third party class libraries in J2EE web applications.

You can also use the MysqlDataSource or MysqlConnectionPoolDataSource classes in the com.mysql.jdbc.jdbc2.optional package, if your J2EE application server supports or requires them. Starting with Connector/J 5.0.0, the javax.sql.XADataSource interface is implemented via the com.mysql.jdbc.jdbc2.optional.MysqlXADataSource class, which supports XA distributed transactions when used in combination with MySQL server version 5.0.

The various MysqlDataSource classes support the following parameters (through standard "set" mutators):

- user
- password

- serverName (see the previous section about fail-over hosts)
- databaseName
- port

1.2.2. Upgrading from an Older Version

MySQL AB tries to keep the upgrade process as easy as possible, however as is the case with any software, sometimes changes need to be made in new versions to support new features, improve existing functionality, or comply with new standards.

This section has information about what users who are upgrading from one version of Connector/J to another (or to a new version of the MySQL server, with respect to JDBC functionality) should be aware of.

1.2.2.1. Upgrading from MySQL Connector/J 3.0 to 3.1

Connector/J 3.1 is designed to be backward-compatible with Connector/J 3.0 as much as possible. Major changes are isolated to new functionality exposed in MySQL-4.1 and newer, which includes Unicode character sets, server-side prepared statements, SQLState codes returned in error messages by the server and various performance enhancements that can be enabled or disabled via configuration properties.

- Unicode Character Sets — See the next section, as well as ???, for information on this new feature of MySQL. If you have something misconfigured, it will usually show up as an error with a message similar to `Illegal mix of collations`.
- *Server-side Prepared Statements* — Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer).

  Starting with version 3.1.7, the driver scans SQL you are preparing via all variants of `Connection.prepareStatement()` to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this feature by passing 'emulateUnsupportedPstmts=false' in your JDBC URL.

  If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the following connection property:

  ```
  useServerPrepStmts=false
  ```

- Datetimes with all-zero components ('0000-00-00 ...') — These values can not be represented reliably in Java. Connector/J 3.0.x always converted them to NULL when being read from a ResultSet.

  Connector/J 3.1 throws an exception by default when these values are encountered as this is the most correct behavior according to the JDBC and SQL standards. This behavior can be modified using the ' zeroDateTimeBehavior ' configuration property. The allowable values are: 'exception' (the default), which throws an SQLException with an SQLState of 'S1009', 'convertToNull', which returns NULL instead of the date, and 'round', which rounds the date to the nearest closest value which is '0001-01-01'.

  Starting with Connector/J 3.1.7, ResultSet.getString() can be decoupled from this behavior via ' noDatetimeStringSync=true ' (the default value is 'false') so that you can get retrieve the unaltered all-zero value as a String. It should be noted that this also precludes using any time zone conversions, therefore the driver will not allow you to enable noDatetimeStringSync and useTimezone at the same time.

- New SQLState Codes — Connector/J 3.1 uses SQL:1999 SQLState codes returned by the MySQL server (if supported), which are different from the “legacy” X/Open state codes that Connector/J 3.0 uses. If connected to a MySQL server older than MySQL-4.1.0 (the oldest version to return SQL-States as part of the error code), the driver will use a built-in mapping. You can revert to the old mapping by using the following configuration property:

  ```
  useSqlStateCodes=false
  ```

- Calling ResultSet.getString() on a BLOB column will now return the address of the byte[] array that represents it, instead of a String representation of the BLOB. BLOBs have no character set, so they can't be converted to java.lang.Strings without data loss or corruption.

  To store strings in MySQL with LOB behavior, use one of the TEXT types, which the driver will treat as a java.sql.Clob.

- Starting with Connector/J 3.1.8 a “debug” build of the driver in a file named "`mysql-connector-java-[version]-bin-g.jar`" is shipped alongside the normal “binary” jar file that is named "`mysql-connector-java-[version]-bin.jar`".

  Starting with Connector/J 3.1.9, we don't ship the .class files “unbundled,” they are only available in the JAR archives that ship with the driver.

  You should not use the “debug” build of the driver unless instructed to do so when reporting a problem or bug to MySQL AB, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the `src/lib/aspectjrt.jar` file that comes with the Connector/J distribution.

1.2.2.2. JDBC-Specific Issues When Upgrading to MySQL Server 4.1 or Newer

- *Using the UTF-8 Character Encoding* - Prior to MySQL server version 4.1, the UTF-8 character encoding was not supported by the server, however the JDBC driver could use it, allowing storage of multiple character sets in latin1 tables on the server.

  Starting with MySQL-4.1, this functionality is deprecated. If you have applications that rely on this functionality, and can not upgrade them to use the official Unicode character support in MySQL server version 4.1 or newer, you should add the following property to your connection URL:

  ```
  useOldUTF8Behavior=true
  ```

- *Server-side Prepared Statements* - Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer). If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the following connection property:

  ```
  useServerPrepStmts=false
  ```

1.2.3. Installing from the Development Source Tree

Caution

You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL Connector/J up and running on your system, you should use a standard release distribution.

To install MySQL Connector/J from the development source tree, make sure that you have the following prerequisites:

- Subversion, to check out the sources from our repository (available from http://subversion.tigris.org/).
- Apache Ant version 1.6 or newer (available from http://ant.apache.org/).
- JDK-1.4.2 or later. Although MySQL Connector/J can be installed on older JDKs, to compile it from source you must have at least JDK-1.4.2.

The Subversion source code repository for MySQL Connector/J is located at http://svn.mysql.com/svnpublic/connector-j. In general, you should not check out the entire repository because it contains every branch and tag for MySQL Connector/J and is quite large.

To check out and compile a specific branch of MySQL Connector/J, follow these steps:

1. At the time of this writing, there are three active branches of Connector/J: `branch_3_0`, `branch_3_1` and `branch_5_0`. Check out the latest code from the branch that you want with the following command (replacing *`[major]`* and *`[minor]`* with appropriate version numbers):

   ```
   shell> svn co http://svn.mysql.com/svnpublic/connector-j/branches/branch_[major
   ```

   This creates a `connector-j` subdirectory in the current directory that contains the latest sources for the requested branch.

2. Change location to the `connector-j` directory to make it your current working directory:

   ```
   shell> cd connector-j
   ```

3. Issue the following command to compile the driver and create a `.jar` file suitable for installation:

   ```
   shell> ant dist
   ```

   This creates a `build` directory in the current directory, where all build output will go. A directory is created in the `build` directory that includes the version number of the sources you are building from. This directory contains the sources, compiled `.class` files, and a `.jar` file suitable for deployment. For other possible targets, including ones that will create a fully packaged distribution, issue the following command:

   ```
   shell> ant --projecthelp
   ```

4. A newly created `.jar` file containing the JDBC driver will be placed in the directory `build/mysql-connector-java-`*`[version]`*.

   Install the newly created JDBC driver as you would a binary `.jar` file that you download from MySQL by following the instructions in Section 1.2.1.3, “Installing the Driver and Configuring the `CLASSPATH`”.

1.3. JDBC Reference

1.3.1. Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J

The name of the class that implements java.sql.Driver in MySQL Connector/J is 'com.mysql.jdbc.Driver'. The 'org.gjt.mm.mysql.Driver' class name is also usable to remain backward-compatible with MM.MySQL. You should use this class name when registering the driver, or when otherwise configuring software to use MySQL Connector/J.

The JDBC URL format for MySQL Connector/J is as follows, with items in square brackets ([,]) being optional:

```
jdbc:mysql://[host][,failoverhost...][:port]/[database][?propertyName1][=propertyV
```

If the hostname is not specified, it defaults to '127.0.0.1'. If the port is not specified, it defaults to '3306', the default port number for MySQL servers.

```
jdbc:mysql://[host:port],[host:port].../[database][?propertyName1][=propertyValue1
```

If the database is not specified, the connection will be made with no default database. In this case, you will need to either call the `setCatalog()` method on the Connection instance or fully-specify table names using the database name (i.e. 'SELECT dbname.tablename.colname FROM dbname.tablename...') in your SQL. Not specifying the database to use upon connection is generally only useful when building tools that work with multiple databases, such as GUI database managers.

MySQL Connector/J has fail-over support. This allows the driver to fail-over to any number of “slave” hosts and still perform read-only queries. Fail-over only happens when the connection is in an autoCommit(true) state, because fail-over can not happen reliably when a transaction is in progress. Most application servers and connection pools set autoCommit to 'true' at the end of every transaction/connection use.

The fail-over functionality has the following behavior:

If the URL property "autoReconnect" is false: Failover only happens at connection initialization, and failback occurs when the driver determines that the first host has become available again.

If the URL property "autoReconnect" is true: Failover happens when the driver determines that the connection has failed (before *every* query), and falls back to the first host when it determines that the host has become available again (after queriesBeforeRetryMaster queries have been issued).

In either case, whenever you are connected to a "failed-over" server, the connection will be set to read-only state, so queries that would modify data will have exceptions thrown (the query will *never* be processed by the MySQL server).

Configuration properties define how Connector/J will make a connection to a MySQL server. Unless otherwise noted, properties can be set for a DataSource object or for a Connection object.

Configuration Properties can be set in one of the following ways:

- Using the set*() methods on MySQL implementations of java.sql.DataSource (which is the preferred method when using implementations of java.sql.DataSource):
  - com.mysql.jdbc.jdbc2.optional.MysqlDataSource
  - com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource

- As a key/value pair in the java.util.Properties instance passed to DriverManager.getConnection() or Driver.connect()
- As a JDBC URL parameter in the URL given to java.sql.DriverManager.getConnection(), java.sql.Driver.connect() or the MySQL implementations of javax.sql.DataSource's setURL() method.

Note

If the mechanism you use to configure a JDBC URL is XML-based, you will need to use the XML character literal & to separate configuration parameters, as the ampersand is a reserved character for XML.

The properties are listed in the following table:

Table 1. Connection Properties

Property Name	Definition	Required?	Default Value
Connection/Authentication			
user	The user to connect as	No	
password	The password to use when connecting	No	
socketFactory	The name of the class that the driver should use for creating socket connections to the server. This class must implement the interface 'com.mysql.jdbc.SocketFactory' and have public no-args constructor.	No	com.mysql.jdbc.StandardSocketFactory
connectTimeout	Timeout for socket connect (in milliseconds), with 0 being no timeout. Only works on JDK-1.4 or newer. Defaults to '0'.	No	0
socketTimeout	Timeout on network socket operations (0, the default means no timeout).	No	0
useConfigs	Load the comma-delimited list of configuration properties before parsing the URL or applying user-specified properties. These configurations are explained in the 'Configurations' of the documentation.	No	
interactiveClient	Set the CLIENT_INTERACTIVE flag, which tells MySQL to timeout connections	No	false

Property Name	Definition	Required?	Default Value
	based on INTERACTIVE_TIMEOUT instead of WAIT_TIMEOUT		
propertiesTransform	An implementation of com.mysql.jdbc.ConnectionPropertiesTransform that the driver will use to modify URL properties passed to the driver before attempting a connection	No	
useCompression	Use zlib compression when communicating with the server (true/false)? Defaults to 'false'.	No	false
High Availability and Clustering			
autoReconnect	Should the driver try to re-establish stale and/or dead connections? If enabled the driver will throw an exception for a queries issued on a stale or dead connection, which belong to the current transaction, but will attempt reconnect before the next query issued on the connection in a new transaction. The use of this feature is not recommended, because it has side effects related to session state and data consistency when applications don'thandle SQLExceptions properly, and is only designed to be used when you are unable to configure your application to handle SQLExceptions resulting from dead andstale connections properly. Alternatively, investigate setting the MySQL server variable "wait_timeout"to some high value rather than the default of 8 hours.	No	false
autoReconnectForPools	Use a reconnection strategy appropriate for connection pools (defaults to 'false')	No	false
failOverReadOnly	When failing over in	No	true

Property Name	Definition	Required?	Default Value
	autoReconnect mode, should the connection be set to 'read-only'?		
reconnectAtTxEnd	If autoReconnect is set to true, should the driver attempt reconnectionsat the end of every transaction?	No	false
roundRobinLoadBalance	When autoReconnect is enabled, and failoverReadonly is false, should we pick hosts to connect to on a round-robin basis?	No	false
queriesBeforeRetryMaster	Number of queries to issue before falling back to master when failed over (when using multi-host failover). Whichever condition is met first, 'queriesBeforeRetryMaster' or 'secondsBeforeRetryMaster' will cause an attempt to be made to reconnect to the master. Defaults to 50.	No	50
secondsBeforeRetryMaster	How long should the driver wait, when failed over, before attempting to reconnect to the master server? Whichever condition is met first, 'queriesBeforeRetryMaster' or 'secondsBeforeRetryMaster' will cause an attempt to be made to reconnect to the master. Time in seconds, defaults to 30	No	30
enableDeprecatedAutoreconnect	Auto-reconnect functionality is deprecated starting with version 3.2, and will be removed in version 3.3. Set this property to 'true' to disable the check for the feature being configured.	No	false
resourceId	A globally unique name that identifies the resource that this datasource or connection is connected to, used for XARe-	No	

Property Name	Definition	Required?	Default Value
	source.isSameRM() when the driver can't determine this value based on hostnames used in the URL		
Security			
allowMultiQueries	Allow the use of ';' to delimit multiple queries during one statement (true/false, defaults to 'false'	No	false
useSSL	Use SSL when communicating with the server (true/false), defaults to 'false'	No	false
requireSSL	Require SSL connection if useSSL=true? (defaults to 'false').	No	false
allowUrlInLocalInfile	Should the driver allow URLs in 'LOAD DATA LOCAL INFILE' statements?	No	false
paranoid	Take measures to prevent exposure sensitive information in error messages and clear data structures holding sensitive data when possible? (defaults to 'false')	No	false
Performance Extensions			
metadataCacheSize	The number of queries to cacheResultSetMetadata for if cacheResultSetMetaData is set to 'true' (default 50)	No	50
prepStmtCacheSize	If prepared statement caching is enabled, how many prepared statements should be cached?	No	25
prepStmtCacheSqlLimit	If prepared statement caching is enabled, what's the largest SQL the driver will cache the parsing for?	No	256
useCursorFetch	If connected to MySQL > 5.0.2, and setFetchSize() > 0 on a statement, should that statement use cursor-based fetching to retrieve rows?	No	false
blobSendChunkSize	Chunk to use when	No	1048576

Property Name	Definition	Required?	Default Value
	sending BLOB/CLOBs via ServerPreparedStatements		
cacheCallableStmts	Should the driver cache the parsing stage of CallableStatements	No	false
cachePrepStmts	Should the driver cache the parsing stage of PreparedStatements of client-side prepared statements, the "check" for suitability of server-side prepared and server-side prepared statements themselves?	No	false
cacheResultSetMetadata	Should the driver cache ResultSetMetaData for Statements and PreparedStatements? (Req. JDK-1.4+, true/false, default 'false')	No	false
cacheServerConfiguration	Should the driver cache the results of 'SHOW VARIABLES' and 'SHOW COLLATION' on a per-URL basis?	No	false
defaultFetchSize	The driver will call setFetchSize(n) with this value on all newly-created Statements	No	0
dontTrackOpenResources	The JDBC specification requires the driver to automatically track and close resources, however if your application doesn't do a good job of explicitly calling close() on statements or result sets, this can cause memory leakage. Setting this property to true relaxes this constraint, and can be more memory efficient for some applications.	No	false
dynamicCalendars	Should the driver retrieve the default calendar when required, or cache it per connection/session?	No	false
elideSetAutoCommits	If using MySQL-4.1 or newer, should the driver only issue 'set autocommit=n' queries when the	No	false

Property Name	Definition	Required?	Default Value
	server's state doesn't match the requested state by Connection.setAutoCommit(boolean)?		
holdResultsOpenOverStatementClose	Should the driver close result sets on Statement.close() as required by the JDBC specification?	No	false
locatorFetchBufferSize	If 'emulateLocators' is configured to 'true', what size buffer should be used when fetching BLOB data for getBinaryInputStream?	No	1048576
rewriteBatchedStatements	Should the driver use multiqueries (irregardless of the setting of "allowMultiQueries") as well as rewriting of prepared statements for INSERT into multi-value inserts when executeBatch() is called? Notice that this has the potential for SQL injection if using plain java.sql.Statements and your code doesn't sanitize input correctly. Notice that for prepared statements, server-side prepared statements can not currently take advantage of this rewrite option, and that if you don't specify stream lengths when using PreparedStatement.set*Stream(),the driver won't be able to determine the optimium number of parameters per batch and you might receive anan error from the driver that the resultant packet is too large. Statement.getGeneratedKeys() for these rewritten statements only works when the entire batch includes INSERT statements.	No	false

Property Name	Definition	Required?	Default Value
useFastIntParsing	Use internal String->Integer conversion routines to avoid excessive object creation?	No	true
useLocalSessionState	Should the driver refer to the internal values of autocommit and transaction isolation that are set by Connection.setAutoCommit() and Connection.setTransactionIsolation(), rather than querying the database?	No	false
useReadAheadInput	Use newer, optimized non-blocking, buffered input stream when reading from the server?	No	true
Debuging/Profiling			
logger	The name of a class that implements 'com.mysql.jdbc.log.Log' that will be used to log messages to.(default is 'com.mysql.jdbc.log.StandardLogger', which logs to STDERR)	No	com.mysql.jdbc.log.StandardLogger
profileSQL	Trace queries and their execution/fetch times to the configured logger (true/false) defaults to 'false'	No	false
reportMetricsIntervalMillis	If 'gatherPerfMetrics' is enabled, how often should they be logged (in ms)?	No	30000
maxQuerySizeToLog	Controls the maximum length/size of a query that will get logged when profiling or tracing	No	2048
packetDebugBufferSize	The maximum number of packets to retain when 'enablePacketDebug' is true	No	20
slowQueryThresholdMillis	If 'logSlowQueries' is enabled, how long should a query (in ms) before it is logged as 'slow'?	No	2000
useUsageAdvisor	Should the driver issue 'usage' warnings advising proper and efficient usage of JDBC and	No	false

Property Name	Definition	Required?	Default Value
	MySQL Connector/J to the log (true/false, defaults to 'false')?		
autoGenerateTestcaseScript	Should the driver dump the SQL it is executing, including server-side prepared statements to STDERR?	No	false
dumpMetadataOnColumnNotFound	Should the driver dump the field-level metadata of a result set into the exception message when ResultSet.findColumn() fails?	No	false
dumpQueriesOnException	Should the driver dump the contents of the query sent to the server in the message for SQLExceptions?	No	false
enablePacketDebug	When enabled, a ringbuffer of 'packetDebugBufferSize' packets will be kept, and dumped when exceptions are thrown in key areas in the driver's code	No	false
explainSlowQueries	If 'logSlowQueries' is enabled, should the driver automatically issue an 'EXPLAIN' on the server and send the results to the configured log at a WARN level?	No	false
logSlowQueries	Should queries that take longer than 'slowQueryThresholdMillis' be logged?	No	false
traceProtocol	Should trace-level network protocol be logged?	No	false
Miscellaneous			
useUnicode	Should the driver use Unicode character encodings when handling strings? Should only be used when the driver can't determine the character set mapping, or you are trying to 'force' the driver to use a character set that MySQL either doesn't natively support (such as UTF-8),	No	true

Property Name	Definition	Required?	Default Value
	true/false, defaults to 'true'		
characterEncoding	If 'useUnicode' is set to true, what character encoding should the driver use when dealing with strings? (defaults is to 'autodetect')	No	
characterSetResults	Character set to tell the server to return results as.	No	
connectionCollation	If set, tells the server to use this collation via 'set collation_connection'	No	
sessionVariables	A comma-separated list of name/value pairs to be sent as SET SESSION ... to the server when the driver connects.	No	
allowNanAndInf	Should the driver allow NaN or +/- INF values in PreparedStatement.setDouble()?	No	false
autoClosePStmtStreams	Should the driver automatically call .close() on streams/readers passed as arguments via set*() methods?	No	false
autoDeserialize	Should the driver automatically detect and deserialize objects stored in BLOB fields?	No	false
capitalizeTypeNames	Capitalize type names in DatabaseMetaData? (usually only useful when using WebObjects, true/false, defaults to 'false')	No	false
clobCharacterEncoding	The character encoding to use for sending and retrieving TEXT, MEDIUMTEXT and LONGTEXT values instead of the configured connection characterEncoding	No	
clobberStreamingResults	This will cause a 'streaming' ResultSet to be automatically closed, and any outstanding data still streaming from the server to be discarded if an-	No	false

Property Name	Definition	Required?	Default Value
	other query is executed before all the data has been read from the server.		
continueBatchOnError	Should the driver continue processing batch commands if one statement fails. The JDBC spec allows either way (defaults to 'true').	No	true
createDatabaseIfNotExist	Creates the database given in the URL if it doesn't yet exist. Assumes the configured user has permissions to create databases.	No	false
emptyStringsConvertToZero	Should the driver allow conversions from empty string fields to numeric values of '0'?	No	true
emulateLocators	N/A	No	false
emulateUnsupportedPstmts	Should the driver detect prepared statements that are not supported by the server, and replace them with client-side emulated versions?	No	true
ignoreNonTxTables	Ignore non-transactional table warning for rollback? (defaults to 'false').	No	false
jdbcCompliantTruncation	Should the driver throw java.sql.DataTruncation exceptions when data is truncated as is required by the JDBC specification when connected to a server that supports warnings(MySQL 4.1.0 and newer)?	No	true
maxRows	The maximum number of rows to return (0, the default means return all rows).	No	-1
noDatetimeStringSync	Don't ensure that ResultSet.getDatetimeType().toString().equals(ResultSet.getString())	No	false
noTimezoneConversionForTimeType	Don't convert TIME values using the server timezone if 'useTimezone'='true'	No	false

Property Name	Definition	Required?	Default Value
nullCatalogMeansCurrent	When DatabaseMetadataMethods ask for a 'catalog' parameter, does the value null mean use the current catalog? (this is not JDBC-compliant, but follows legacy behavior from earlier versions of the driver)	No	true
nullNamePatternMatchesAll	Should DatabaseMetaData methods that accept *pattern parameters treat null the same as '%' (this is not JDBC-compliant, however older versions of the driver accepted this departure from the specification)	No	true
overrideSupportsIntegrityEnhancementFacility	Should the driver return "true" for DatabaseMetaData.supportsIntegrityEnhancementFacility() even if the database doesn't support it to workaround applications that require this method to return "true" to signal support of foreign keys, even though the SQL specification states that this facility contains much more than just foreign key support (one such application being OpenOffice)?	No	false
pedantic	Follow the JDBC spec to the letter.	No	false
processEscapeCodesForPrepStmts	Should the driver process escape codes in queries that are prepared?	No	true
relaxAutoCommit	If the version of MySQL the driver connects to does not support transactions, still allow calls to commit(), rollback() and setAutoCommit() (true/false, defaults to 'false')?	No	false
retainStatementAfterResultSetClose	Should the driver retain the Statement reference in a ResultSet after ResultSet.close() has been called. This is not JD-	No	false

Property Name	Definition	Required?	Default Value
	BC-compliant after JDBC-4.0.		
rollbackOnPooledClose	Should the driver issue a rollback() when the logical connection in a pool is closed?	No	true
runningCTS13	Enables workarounds for bugs in Sun's JDBC compliance testsuite version 1.3	No	false
serverTimezone	Override detection/mapping of timezone. Used when timezone from server doesn't map to Java timezone	No	
strictFloatingPoint	Used only in older versions of compliance test	No	false
strictUpdates	Should the driver do strict checking (all primary keys selected) of updatable result sets (true, false, defaults to 'true')?	No	true
tinyInt1isBit	Should the driver treat the datatype TINYINT(1) as the BIT type (because the server silently converts BIT -> TINYINT(1) when creating tables)?	No	true
transformedBitIsBoolean	If the driver converts TINYINT(1) to a different type, should it use BOOLEAN instead of BIT for future compatibility with MySQL-5.0, as MySQL-5.0 has a BIT type?	No	false
ultraDevHack	Create PreparedStatements for prepareCall() when required, because UltraDev is broken and issues a prepareCall() for _all_ statements? (true/false, defaults to 'false')	No	false
useGmtMillisForDatetimes	Convert between session timezone and GMT before creating Date and Timestamp instances (value of "false" is legacy behavior, "true" leads to more JDBC-	No	false

Property Name	Definition	Required?	Default Value
	compliant behavior.		
useHostsInPrivileges	Add '@hostname' to users in DatabaseMetaData.getColumn/TablePrivileges() (true/false), defaults to 'true'.	No	true
useInformationSchema	When connected to MySQL-5.0.7 or newer, should the driver use the INFORMATION_SCHEMA to derive information used by DatabaseMetaData?	No	false
useJDBCCompliantTimezoneShift	Should the driver use JDBC-compliant rules when converting TIME/TIMESTAMP/DATETIME values' timezone information for those JDBC arguments which take a java.util.Calendar argument? (Notice that this option is exclusive of the "useTimezone=true" configuration option.)	No	false
useOldUTF8Behavior	Use the UTF-8 behavior the driver did when communicating with 4.0 and older servers	No	false
useOnlyServerErrorMessages	Don't prepend 'standard' SQLState error messages to error messages returned by the server.	No	true
useServerPrepStmts	Use server-side prepared statements if the server supports them? (defaults to 'true').	No	true
useSqlStateCodes	Use SQL Standard state codes instead of 'legacy' X/Open/SQL state codes (true/false), default is 'true'	No	true
useStreamLengthsInPrepStmts	Honor stream length parameter in PreparedStatement/ResultSet.setXXXStream() method calls (true/false, defaults to 'true')?	No	true
useTimezone	Convert time/date types between client and server timezones (true/false,	No	false

Property Name	Definition	Required?	Default Value
	defaults to 'false')?		
useUnbufferedInput	Don't use BufferedInput-Stream for reading data from the server	No	true
yearIsDateType	Should the JDBC driver treat the MySQL type "YEAR" as a java.sql.Date, or as a SHORT?	No	true
zeroDateTimeBehavior	What should happen when the driver encounters DATETIME values that are composed entirely of zeroes (used by MySQL to represent invalid dates)? Valid values are 'exception', 'round' and 'convertToNull'.	No	exception

Connector/J also supports access to MySQL via named pipes on Windows NT/2000/XP using the 'NamedPipeSocketFactory' as a plugin-socket factory via the 'socketFactory' property. If you don't use a 'namedPipePath' property, the default of '\\.\pipe\MySQL' will be used. If you use the NamedPipeSocketFactory, the hostname and port number values in the JDBC url will be ignored.

Adding the following property to your URL will enable the NamedPipeSocketFactory:

socketFactory=com.mysql.jdbc.NamedPipeSocketFactory

Named pipes only work when connecting to a MySQL server on the same physical machine as the one the JDBC driver is being used on. In simple performance tests, it appears that named pipe access is between 30%-50% faster than the standard TCP/IP access.

You can create your own socket factories by following the example code in `com.mysql.jdbc.NamedPipeSocketFactory`, or `com.mysql.jdbc.StandardSocketFactory`.

1.3.2. JDBC API Implementation Notes

MySQL Connector/J passes all of the tests in the publicly-available version of Sun's JDBC compliance test suite. However, in many places the JDBC specification is vague about how certain functionality should be implemented, or the specification allows leeway in implementation.

This section gives details on a interface-by-interface level about how certain implementation decisions may affect how you use MySQL Connector/J.

- Blob

  The Blob implementation does not allow in-place modification (they are 'copies', as reported by the DatabaseMetaData.locatorsUpdateCopies() method). Because of this, you should use the corresponding PreparedStatement.setBlob() or ResultSet.updateBlob() (in the case of updatable result sets) methods to save changes back to the database.

  Starting with Connector/J version 3.1.0, you can emulate Blobs with locators by adding the property

'emulateLocators=true' to your JDBC URL. You must then use a column alias with the value of the column set to the actual name of the Blob column in the SELECT that you write to retrieve the Blob. The SELECT must also reference only one table, the table must have a primary key, and the SELECT must cover all columns that make up the primary key. The driver will then delay loading the actual Blob data until you retrieve the Blob and call retrieval methods (getInputStream(), getBytes(), and so forth) on it.

- CallableStatement

  Starting with Connector/J 3.1.1, stored procedures are supported when connecting to MySQL version 5.0 or newer via the `CallableStatement` interface. Currently, the `getParameterMetaData()` method of `CallableStatement` is not supported.

- Clob

  The Clob implementation does not allow in-place modification (they are 'copies', as reported by the DatabaseMetaData.locatorsUpdateCopies() method). Because of this, you should use the PreparedStatement.setClob() method to save changes back to the database. The JDBC API does not have a ResultSet.updateClob() method.

- Connection

  Unlike older versions of MM.MySQL the `isClosed()` method does not “ping” the server to determine if it is alive. In accordance with the JDBC specification, it only returns true if 'closed()' has been called on the connection. If you need to determine if the connection is still valid, you should issue a simple query, such as "SELECT 1". The driver will throw an exception if the connection is no longer valid.

- DatabaseMetaData

  Foreign Key information (getImported/ExportedKeys() and getCrossReference()) is only available from 'InnoDB'-type tables. However, the driver uses 'SHOW CREATE TABLE' to retrieve this information, so when other storage engines support foreign keys, the driver will transparently support them as well.

- Driver

- PreparedStatement

  PreparedStatements are implemented by the driver, as MySQL does not have a prepared statement feature. Because of this, the driver does not implement getParameterMetaData() or getMetaData() as it would require the driver to have a complete SQL parser in the client.

  Starting with version 3.1.0 MySQL Connector/J, server-side prepared statements and 'binary-encoded' result sets are used when the server supports them.

  Take care when using a server-side prepared statement with “large” parameters that are set via setBinaryStream(), setAsciiStream(), setUnicodeStream(), setBlob(), or setClob(). If you want to re-execute the statement with any “large” parameter changed to a non-“large” parameter, it is necessary to call clearParameters() and set all parameters again. The reason for this is as follows:

  - The driver streams the 'large' data 'out-of-band' to the prepared statement on the server side when the parameter is set (before execution of the prepared statement).

  - Once that has been done, the stream used to read the data on the client side is closed (as per the JDBC spec), and can't be read from again.

  - If a parameter changes from “large” to non-“large,” the driver must reset the server-side state of

the prepared statement to allow the parameter that is being changed to take the place of the prior “large” value. This removes all of the 'large' data that has already been sent to the server, thus requiring the data to be re-sent, via the setBinaryStream(), setAsciiStream(), setUnicodeStream(), setBlob() or setClob() methods.

Consequently, if you want to change the “type” of a parameter to a non-“large” one, you must call clearParameters() and set all parameters of the prepared statement again before it can be re-executed.

- ResultSet

  By default, ResultSets are completely retrieved and stored in memory. In most cases this is the most efficient way to operate, and due to the design of the MySQL network protocol is easier to implement. If you are working with ResultSets that have a large number of rows or large values, and can not allocate heap space in your JVM for the memory required, you can tell the driver to 'stream' the results back one row at a time.

  To enable this functionality, you need to create a Statement instance in the following manner:

  ```
  stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                 java.sql.ResultSet.CONCUR_READ_ONLY);
  stmt.setFetchSize(Integer.MIN_VALUE);
  ```

  The combination of a forward-only, read-only result set, with a fetch size of Integer.MIN_VALUE serves as a signal to the driver to “stream” result sets row-by-row. After this any result sets created with the statement will be retrieved row-by-row.

  There are some caveats with this approach. You will have to read all of the rows in the result set (or close it) before you can issue any other queries on the connection, or an exception will be thrown.

  The earliest the locks these statements hold can be released (whether they be `MyISAM` table-level locks or row-level locks in some other storage engine such as `InnoDB`) is when the statement completes.

  If the statement is within scope of a transaction, then locks are released when the transaction completes (which implies that the statement needs to complete first). As with most other databases, statements are not complete until all the results pending on the statement are read or the active result set for the statement is closed.

  Therefore, if using “streaming” results, you should process them as quickly as possible if you want to maintain concurrent access to the tables referenced by the statement producing the result set.

- ResultSetMetaData

  The "isAutoIncrement()" method only works when using MySQL servers 4.0 and newer.

- Statement

  When using versions of the JDBC driver earlier than 3.2.1, and connected to server versions earlier than 5.0.3, the "setFetchSize()" method has no effect, other than to toggle result set streaming as described above.

  MySQL does not support SQL cursors, and the JDBC driver doesn't emulate them, so "setCursorName()" has no effect.

1.3.3. Java, JDBC and MySQL Types

MySQL Connector/J is flexible in the way it handles conversions between MySQL data types and Java data types.

In general, any MySQL data type can be converted to a java.lang.String, and any numerical type can be converted to any of the Java numerical types, although round-off, overflow, or loss of precision may occur.

Starting with Connector/J 3.1.0, the JDBC driver will issue warnings or throw DataTruncation exceptions as is required by the JDBC specification unless the connection was configured not to do so by using the property "jdbcCompliantTruncation" and setting it to "false".

The conversions that are always guaranteed to work are listed in the following table:

Table 2. Conversion Table

These MySQL Data Types	Can always be converted to these Java types
CHAR, VARCHAR, BLOB, TEXT, ENUM, and SET	`java.lang.String`, `java.io.InputStream`, `java.io.Reader`, `java.sql.Blob`, `java.sql.Clob`
FLOAT, REAL, DOUBLE PRECISION, NUMERIC, DECIMAL, TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT	`java.lang.String`, `java.lang.Short`, `java.lang.Integer`, `java.lang.Long`, `java.lang.Double`, `java.math.BigDecimal` **Note** round-off, overflow or loss of precision may occur if you choose a Java numeric data type that has less precision or capacity than the MySQL data type you are converting to/from.
DATE, TIME, DATETIME, TIMESTAMP	`java.lang.String`, `java.sql.Date`, `java.sql.Timestamp`

The `ResultSet.getObject()` method uses the following type conversions between MySQL and Java types, following the JDBC specification where appropriate:

Table 3. MySQL Types to Java Types for ResultSet.getObject()

MySQL Type Name	Returned as Java Class
BIT(1) (new in MySQL-5.0)	`java.lang.Boolean`
BIT(> 1) (new in MySQL-5.0)	`byte[]`
TINYINT	`java.lang.Boolean` if the configuration property "tinyInt1isBit" is set to "true" (the default) and the storage size is "1", or `java.lang.Integer` if not.
BOOL , BOOLEAN	See TINYINT, above as these are aliases for TINYINT(1), currently.
SMALLINT[(M)] [UNSIGNED]	`java.lang.Integer` (regardless if UNSIGNED or not)

MySQL Type Name	Returned as Java Class
MEDIUMINT[(M)] [UNSIGNED]	`java.lang.Integer`, if UNSIGNED `java.lang.Long`
INT,INTEGER[(M)] [UNSIGNED]	`java.lang.Integer`, if UNSIGNED `java.lang.Long`
BIGINT[(M)] [UNSIGNED]	`java.lang.Long`, if UNSIGNED `java.math.BigInteger`
FLOAT[(M,D)]	`java.lang.Float`
DOUBLE[(M,B)]	`java.lang.Double`
DECIMAL[(M[,D])]	`java.math.BigDecimal`
DATE	`java.sql.Date`
DATETIME	`java.sql.Timestamp`
TIMESTAMP[(M)]	`java.sql.Timestamp`
TIME	`java.sql.Time`
YEAR[(2|4)]	`java.sql.Date` (with the date set two January 1st, at midnight)
CHAR(M)	`java.lang.String` (unless the character set for the column is BINARY, then `byte[]` is returned.
VARCHAR(M) [BINARY]	`java.lang.String` (unless the character set for the column is BINARY, then `byte[]` is returned.
BINARY(M)	`byte[]`
VARBINARY(M)	`byte[]`
TINYBLOB	`byte[]`
TINYTEXT	`java.lang.String`
BLOB	`byte[]`
TEXT	`java.lang.String`
MEDIUMBLOB	`byte[]`
MEDIUMTEXT	`java.lang.String`
LONGBLOB	`byte[]`
LONGTEXT	`java.lang.String`
ENUM('value1','value2',...)	`java.lang.String`
SET('value1','value2',...)	`java.lang.String`

1.3.4. Using Character Sets and Unicode

All strings sent from the JDBC driver to the server are converted automatically from native Java Unicode form to the client character encoding, including all queries sent via `Statement.execute()`, `Statement.executeUpdate()`, `Statement.executeQuery()` as well as all `PreparedStatement` and `CallableStatement` parameters with the exclusion of parameters set using `setBytes()`, `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()` and `setBlob()`.

Prior to MySQL Server 4.1, Connector/J supported a single character encoding per connection, which could either be automatically detected from the server configuration, or could be configured by the user through the *`"useUnicode"`* and "*`characterEncoding`*" properties.

Starting with MySQL Server 4.1, Connector/J supports a single character encoding between client and server, and any number of character encodings for data returned by the server to the client in `ResultSets`.

The character encoding between client and server is automatically detected upon connection. The encoding used by the driver is specified on the server via the `character_set` system variable for server versions older than 4.1.0 and `character_set_server` for server versions 4.1.0 and newer. For more information, see ???.

To override the automatically-detected encoding on the client side, use the *`characterEncoding`* property in the URL used to connect to the server.

When specifying character encodings on the client side, Java-style names should be used. The following table lists Java-style names for MySQL character sets:

Table 4. MySQL to Java Encoding Name Translations

MySQL Character Set Name	Java-Style Character Encoding Name
usa7	US-ASCII
big5	Big5
gbk	GBK
sjis	SJIS (or Cp932 or MS932 for MySQL Server < 4.1.11)
cp932	Cp932 or MS932 (MySQL Server > 4.1.11)
gb2312	EUC_CN
ujis	EUC_JP
euc_kr	EUC_KR
latin1	ISO8859_1
latin1_de	ISO8859_1
german1	ISO8859_1
danish	ISO8859_1
latin2	ISO8859_2
czech	ISO8859_2
hungarian	ISO8859_2
croat	ISO8859_2
greek	ISO8859_7
hebrew	ISO8859_8
latin5	ISO8859_9
latvian	ISO8859_13
latvian1	ISO8859_13
estonia	ISO8859_13
dos	Cp437
pclatin2	Cp852
cp866	Cp866
koi8_ru	KOI8_R
tis620	TIS620

MySQL Character Set Name	Java-Style Character Encoding Name
win1250	Cp1250
win1250ch	Cp1250
win1251	Cp1251
cp1251	Cp1251
win1251ukr	Cp1251
cp1257	Cp1257
macroman	MacRoman
macce	MacCentralEurope
utf8	UTF-8
ucs2	UnicodeBig

Warning

Do not issue the query 'set names' with Connector/J, as the driver will not detect that the character set has changed, and will continue to use the character set detected during the initial connection setup.

To allow multiple character sets to be sent from the client, the "UTF-8" encoding should be used, either by configuring "utf8" as the default server character set, or by configuring the JDBC driver to use "UTF-8" through the `characterEncoding` property.

1.3.5. Connecting Securely Using SSL

SSL in MySQL Connector/J encrypts all data (other than the initial handshake) between the JDBC driver and the server. The performance penalty for enabling SSL is an increase in query processing time between 35% and 50%, depending on the size of the query, and the amount of data it returns.

For SSL Support to work, you must have the following:

- A JDK that includes JSSE (Java Secure Sockets Extension), like JDK-1.4.1 or newer. SSL does not currently work with a JDK that you can add JSSE to, like JDK-1.2.x or JDK-1.3.x due to the following JSSE bug: http://developer.java.sun.com/developer/bugParade/bugs/4273544.html
- A MySQL server that supports SSL and has been compiled and configured to do so, which is MySQL-4.0.4 or later, see: http://www.mysql.com/doc/en/secure-connections.html
- A client certificate (covered later in this section)

You will first need to import the MySQL server CA Certificate into a Java truststore. A sample MySQL server CA Certificate is located in the 'SSL' subdirectory of the MySQL source distribution. This is what SSL will use to determine if you are communicating with a secure MySQL server.

To use Java's 'keytool' to create a truststore in the current directory , and import the server's CA certificate ('cacert.pem'), you can do the following (assuming that'keytool' is in your path. It's located in the 'bin' subdirectory of your JDK or JRE):

```
shell> keytool -import -alias mysqlServerCACert -file cacert.pem -keystore truststo
```

Keytool will respond with the following information:

```
Enter keystore password:  *********
Owner: EMAILADDRESS=walrus@example.com, CN=Walrus, O=MySQL AB, L=Orenburg, ST=Some
-State, C=RU
Issuer: EMAILADDRESS=walrus@example.com, CN=Walrus, O=MySQL AB, L=Orenburg, ST=Som
e-State, C=RU
Serial number: 0
Valid from: Fri Aug 02 16:55:53 CDT 2002 until: Sat Aug 02 16:55:53 CDT 2003
Certificate fingerprints:
         MD5:  61:91:A0:F2:03:07:61:7A:81:38:66:DA:19:C4:8D:AB
         SHA1: 25:77:41:05:D5:AD:99:8C:14:8C:CA:68:9C:2F:B8:89:C3:34:4D:6C
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

You will then need to generate a client certificate, so that the MySQL server knows that it is talking to a secure client:

```
 shell> keytool -genkey -keyalg rsa -alias mysqlClientCertificate -keystore keysto
```

Keytool will prompt you for the following information, and create a keystore named 'keystore' in the current directory.

You should respond with information that is appropriate for your situation:

```
Enter keystore password:  *********
What is your first and last name?
  [Unknown]:  Matthews
What is the name of your organizational unit?
  [Unknown]:  Software Development
What is the name of your organization?
  [Unknown]:  MySQL AB
What is the name of your City or Locality?
  [Unknown]:  Flossmoor
What is the name of your State or Province?
  [Unknown]:  IL
What is the two-letter country code for this unit?
  [Unknown]:  US
Is <CN=Matthews, OU=Software Development, O=MySQL AB,
 L=Flossmoor, ST=IL, C=US> correct?
  [no]:  y

Enter key password for <mysqlClientCertificate>
        (RETURN if same as keystore password):
```

Finally, to get JSSE to use the keystore and truststore that you have generated, you need to set the following system properties when you start your JVM, replacing 'path_to_keystore_file' with the full path to the keystore file you created, 'path_to_truststore_file' with the path to the truststore file you created, and using the appropriate password values for each property.

```
-Djavax.net.ssl.keyStore=path_to_keystore_file
-Djavax.net.ssl.keyStorePassword=*********
-Djavax.net.ssl.trustStore=path_to_truststore_file
-Djavax.net.ssl.trustStorePassword=*********
```

You will also need to set 'useSSL' to 'true' in your connection parameters for MySQL Connector/J, either by adding 'useSSL=true' to your URL, or by setting the property 'useSSL' to 'true' in the java.util.Properties instance you pass to DriverManager.getConnection().

You can test that SSL is working by turning on JSSE debugging (as detailed below), and look for the

following key events:

```
...
*** ClientHello, v3.1
RandomCookie:  GMT: 1018531834 bytes = { 199, 148, 180, 215, 74, 12, 54, 244, 0,
Session ID:  {}
Cipher Suites:  { 0, 5, 0, 4, 0, 9, 0, 10, 0, 18, 0, 19, 0, 3, 0, 17 }
Compression Methods:  { 0 }
***
[write] MD5 and SHA1 hashes:  len = 59
0000: 01 00 00 37 03 01 3D B6   90 FA C7 94 B4 D7 4A 0C  ...7..=.......J.
0010: 36 F4 00 A8 37 67 D7 40   10 8A E1 BE 84 99 02 D9  6...7g.@........
0020: DB EF CA 13 79 4E 00 00   10 00 05 00 04 00 09 00  ....yN..........
0030: 0A 00 12 00 13 00 03 00   11 01 00                 ...........
main, WRITE:  SSL v3.1 Handshake, length = 59
main, READ:  SSL v3.1 Handshake, length = 74
*** ServerHello, v3.1
RandomCookie:  GMT: 1018577560 bytes = { 116, 50, 4, 103, 25, 100, 58, 202, 79, 1
Session ID:  {163, 227, 84, 53, 81, 127, 252, 254, 178, 179, 68, 63, 182, 158, 30
Cipher Suite:  { 0, 5 }
Compression Method: 0
***
%% Created:  [Session-1, SSL_RSA_WITH_RC4_128_SHA]
** SSL_RSA_WITH_RC4_128_SHA
[read] MD5 and SHA1 hashes:  len = 74
0000: 02 00 00 46 03 01 3D B6   43 98 74 32 04 67 19 64  ...F..=.C.t2.g.d
0010: 3A CA 4F B9 B2 64 D7 42   FE 15 53 BB BE 2A AA 03  :.O..d.B..S..*..
0020: 84 6E 52 94 A0 5C 20 A3   E3 54 35 51 7F FC FE B2  .nR..\ ..T5Q....
0030: B3 44 3F B6 9E 1E 0B 96   4F AA 4C FF 5C 0F E2 18  .D?.....O.L.\...
0040: 11 B1 DB 9E B1 BB 8F 00   05 00                    ..........
main, READ:  SSL v3.1 Handshake, length = 1712
...
```

JSSE provides debugging (to STDOUT) when you set the following system property: -Djavax.net.debug=all This will tell you what keystores and truststores are being used, as well as what is going on during the SSL handshake and certificate exchange. It will be helpful when trying to determine what is not working when trying to get an SSL connection to happen.

1.3.6. Using Master/Slave Replication with ReplicationConnection

Starting with Connector/J 3.1.7, we've made available a variant of the driver that will automatically send queries to a read/write master, or a failover or round-robin loadbalanced set of slaves based on the state of `Connection.getReadOnly()` .

An application signals that it wants a transaction to be read-only by calling `Connection.setReadOnly(true)`, this “replication-aware” connection will use one of the slave connections, which are load-balanced per-vm using a round-robin scheme (a given connection is “sticky” to a slave unless that slave is removed from service). If you have a write transaction, or if you have a read that is “time-sensitive” (remember, replication in MySQL is asynchronous), set the connection to be not read-only, by calling `Connection.setReadOnly(false)` and the driver will ensure that further calls are sent to the “master” MySQL server. The driver takes care of propagating the current state of autocommit, isolation level, and catalog between all of the connections that it uses to accomplish this load balancing functionality.

To enable this functionality, use the " `com.mysql.jdbc.ReplicationDriver` " class when configuring your application server's connection pool or when creating an instance of a JDBC driver for your standalone application. Because it accepts the same URL format as the standard MySQL JDBC driver, `ReplicationDriver` does not currently work with `java.sql.DriverManager` -based connection creation unless it is the only MySQL JDBC driver registered with the `DriverManager` .

Here is a short, simple example of how ReplicationDriver might be used in a standalone application.

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.util.Properties;

import com.mysql.jdbc.ReplicationDriver;

public class ReplicationDriverDemo {

    public static void main(String[] args) throws Exception {
        ReplicationDriver driver = new ReplicationDriver();

        Properties props = new Properties();

        // We want this for failover on the slaves
        props.put("autoReconnect", "true");

        // We want to load balance between the slaves
        props.put("roundRobinLoadBalance", "true");

        props.put("user", "foo");
        props.put("password", "bar");

        //
        // Looks like a normal MySQL JDBC url, with a comma-separated list
        // of hosts, the first being the 'master', the rest being any number
        // of slaves that the driver will load balance against
        //

        Connection conn =
            driver.connect("jdbc:mysql://master,slave1,slave2,slave3/test",
                props);

        //
        // Perform read/write work on the master
        // by setting the read-only flag to "false"
        //

        conn.setReadOnly(false);
        conn.setAutoCommit(false);
        conn.createStatement().executeUpdate("UPDATE some_table ....");
        conn.commit();

        //
        // Now, do a query from a slave, the driver automatically picks one
        // from the list
        //

        conn.setReadOnly(true);

        ResultSet rs = conn.createStatement().executeQuery("SELECT a,b,c FROM some

         .......
    }
}
```

1.4. Using Connector/J with J2EE and Other Java Frameworks

This section describes how to use Connector/J in several contexts.

1.4.1. General J2EE Concepts

This section provides general background on J2EE concepts that pertain to use of Connector/J.

1.4.1.1. Understanding Connection Pooling

Connection pooling is a technique of creating and managing a pool of connections that are ready for use by any thread that needs them.

This technique of “pooling” connections is based on the fact that most applications only need a thread to have access to a JDBC connection when they are actively processing a transaction, which usually take only milliseconds to complete. When not processing a transaction, the connection would otherwise sit idle. Instead, connection pooling allows the idle connection to be used by some other thread to do useful work.

In practice, when a thread needs to do work against a MySQL or other database with JDBC, it requests a connection from the pool. When the thread is finished using the connection, it returns it to the pool, so that it may be used by any other threads that want to use it.

When the connection is “loaned out” from the pool, it is used exclusively by the thread that requested it. From a programming point of view, it is the same as if your thread called DriverManager.getConnection() every time it needed a JDBC connection, however with connection pooling, your thread may end up using either a new, or already-existing connection.

Connection pooling can greatly increase the performance of your Java application, while reducing overall resource usage. The main benefits to connection pooling are:

- Reduced connection creation time

  Although this is not usually an issue with the quick connection setup that MySQL offers compared to other databases, creating new JDBC connections still incurs networking and JDBC driver overhead that will be avoided if connections are “recycled.”

- Simplified programming model

  When using connection pooling, each individual thread can act as though it has created its own JDBC connection, allowing you to use straight-forward JDBC programming techniques.

- Controlled resource usage

  If you don't use connection pooling, and instead create a new connection every time a thread needs one, your application's resource usage can be quite wasteful and lead to unpredictable behavior under load.

Remember that each connection to MySQL has overhead (memory, CPU, context switches, and so forth) on both the client and server side. Every connection limits how many resources there are available to your application as well as the MySQL server. Many of these resources will be used whether or not the connection is actually doing any useful work!

Connection pools can be tuned to maximize performance, while keeping resource utilization below the point where your application will start to fail rather than just run slower.

Luckily, Sun has standardized the concept of connection pooling in JDBC through the JDBC-2.0 "Optional" interfaces, and all major application servers have implementations of these APIs that work fine with MySQL Connector/J.

Generally, you configure a connection pool in your application server configuration files, and access it via the Java Naming and Directory Interface (JNDI). The following code shows how you might use a connection pool from an application deployed in a J2EE application server:

Example 12. Using a Connection Pool with a J2EE Application Server

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.InitialContext;
import javax.sql.DataSource;


public class MyServletJspOrEjb {

    public void doSomething() throws Exception {
        /*
         * Create a JNDI Initial context to be able to
         *  lookup  the DataSource
         *
         * In production-level code, this should be cached as
         * an instance or static variable, as it can
         * be quite expensive to create a JNDI context.
         *
         * Note: This code only works when you are using servlets
         * or EJBs in a J2EE application server. If you are
         * using connection pooling in standalone Java code, you
         * will have to create/configure datasources using whatever
         * mechanisms your particular connection pooling library
         * provides.
         */

        InitialContext ctx = new InitialContext();

         /*
          * Lookup the DataSource, which will be backed by a pool
          * that the application server provides. DataSource instances
          * are also a good candidate for caching as an instance
          * variable, as JNDI lookups can be expensive as well.
          */

        DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/MySQLDB");

        /*
         * The following code is what would actually be in your
         * Servlet, JSP or EJB 'service' method...where you need
         * to work with a JDBC connection.
         */

        Connection conn = null;
        Statement stmt = null;

        try {
            conn = ds.getConnection();

            /*
             * Now, use normal JDBC programming to work with
             * MySQL, making sure to close each resource when you're
             * finished with it, which allows the connection pool
             * resources to be recovered as quickly as possible
```

```
             */

            stmt = conn.createStatement();
            stmt.execute("SOME SQL QUERY");

            stmt.close();
            stmt = null;

            conn.close();
            conn = null;
        } finally {
            /*
             * close any jdbc instances here that weren't
             * explicitly closed during normal code path, so
             * that we don't 'leak' resources...
             */

            if (stmt != null) {
                try {
                    stmt.close();
                } catch (sqlexception sqlex) {
                    // ignore -- as we can't do anything about it here
                }

                stmt = null;
            }

            if (conn != null) {
                try {
                    conn.close();
                } catch (sqlexception sqlex) {
                    // ignore -- as we can't do anything about it here
                }

                conn = null;
            }
        }
    }
}
```

As shown in the example above, after obtaining the JNDI InitialContext, and looking up the DataSource, the rest of the code should look familiar to anyone who has done JDBC programming in the past.

The most important thing to remember when using connection pooling is to make sure that no matter what happens in your code (exceptions, flow-of-control, and so forth), connections, and anything created by them (such as statements or result sets) are closed, so that they may be re-used, otherwise they will be "stranded," which in the best case means that the MySQL server resources they represent (such as buffers, locks, or sockets) may be tied up for some time, or worst case, may be tied up forever.

What's the Best Size for my Connection Pool?

As with all other configuration rules-of-thumb, the answer is "It depends." Although the optimal size depends on anticipated load and average database transaction time, the optimum connection pool size is smaller than you might expect. If you take Sun's Java Petstore blueprint application for example, a connection pool of 15-20 connections can serve a relatively moderate load (600 concurrent users) using MySQL and Tomcat with response times that are acceptable.

To correctly size a connection pool for your application, you should create load test scripts with tools such as Apache JMeter or The Grinder, and load test your application.

An easy way to determine a starting point is to configure your connection pool's maximum number of

connections to be “unbounded,” run a load test, and measure the largest amount of concurrently used connections. You can then work backward from there to determine what values of minimum and maximum pooled connections give the best performance for your particular application.

1.4.2. Using Connector/J with Tomcat

The following instructions are based on the instructions for Tomcat-5.x, available at http://jakarta.apache.org/tomcat/tomcat-5.0-doc/jndi-datasource-examples-howto.html which is current at the time this document was written.

First, install the .jar file that comes with Connector/J in `$CATALINA_HOME/common/lib` so that it is available to all applications installed in the container.

Next, Configure the JNDI DataSource by adding a declaration resource to `$CATALINA_HOME/conf/server.xml` in the context that defines your web application:

```
<Context ....>

  ...

  <Resource name="jdbc/MySQLDB"
              auth="Container"
              type="javax.sql.DataSource"/>

  <!-- The name you used above, must match _exactly_ here!

       The connection pool will be bound into JNDI with the name
       "java:/comp/env/jdbc/MySQLDB"
  -->

  <ResourceParams name="jdbc/MySQLDB">
    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>

    <!-- Don't set this any higher than max_connections on your
         MySQL server, usually this should be a 10 or a few 10's
         of connections, not hundreds or thousands -->

    <parameter>
      <name>maxActive</name>
      <value>10</value>
    </parameter>

    <!-- You don't want to many idle connections hanging around
         if you can avoid it, only enough to soak up a spike in
         the load -->

    <parameter>
      <name>maxIdle</name>
      <value>5</value>
    </parameter>

    <!-- Don't use autoReconnect=true, it's going away eventually
         and it's a crutch for older connection pools that couldn't
         test connections. You need to decide whether your application is
         supposed to deal with SQLExceptions (hint, it should), and
         how much of a performance penalty you're willing to pay
         to ensure 'freshness' of the connection -->

    <parameter>
```

```
    <name>validationQuery</name>
    <value>SELECT 1</value>
  </parameter>

 <!-- The most conservative approach is to test connections
      before they're given to your application. For most applications
      this is okay, the query used above is very small and takes
      no real server resources to process, other than the time used
      to traverse the network.

      If you have a high-load application you'll need to rely on
      something else. -->

  <parameter>
    <name>testOnBorrow</name>
    <value>true</value>
  </parameter>

 <!-- Otherwise, or in addition to testOnBorrow, you can test
      while connections are sitting idle -->

  <parameter>
    <name>testWhileIdle</name>
    <value>true</value>
  </parameter>

  <!-- You have to set this value, otherwise even though
       you've asked connections to be tested while idle,
       the idle evicter thread will never run -->

  <parameter>
    <name>timeBetweenEvictionRunsMillis</name>
    <value>10000</value>
  </parameter>

  <!-- Don't allow connections to hang out idle too long,
       never longer than what wait_timeout is set to on the
       server...A few minutes or even fraction of a minute
       is sometimes okay here, it depends on your application
       and how much spikey load it will see -->

  <parameter>
    <name>minEvictableIdleTimeMillis</name>
    <value>60000</value>
  </parameter>

  <!-- Username and password used when connecting to MySQL -->

  <parameter>
   <name>username</name>
   <value>someuser</value>
  </parameter>

  <parameter>
   <name>password</name>
   <value>somepass</value>
  </parameter>

  <!-- Class name for the Connector/J driver -->

  <parameter>
     <name>driverClassName</name>
     <value>com.mysql.jdbc.Driver</value>
  </parameter>
```

```
    <!-- The JDBC connection url for connecting to MySQL, notice
         that if you want to pass any other MySQL-specific parameters
         you should pass them here in the URL, setting them using the
         parameter tags above will have no effect, you will also
         need to use &amp;amp; to separate parameter values as the
         ampersand is a reserved character in XML -->

    <parameter>
      <name>url</name>
      <value>jdbc:mysql://localhost:3306/test</value>
    </parameter>

  </ResourceParams>
</Context>
```

In general, you should follow the installation instructions that come with your version of Tomcat, as the way you configure datasources in Tomcat changes from time-to-time, and unfortunately if you use the wrong syntax in your XML file, you will most likely end up with an exception similar to the following:

```
Error: java.sql.SQLException: Cannot load JDBC driver class 'null ' SQL
state: null
```

1.4.3. Using Connector/J with JBoss

These instructions cover JBoss-4.x. To make the JDBC driver classes available to the application server, copy the .jar file that comes with Connector/J to the `lib` directory for your server configuration (which is usually called "`default`"). Then, in the same configuration directory, in the subdirectory named “deploy,” create a datasource configuration file that ends with "-ds.xml", which tells JBoss to deploy this file as a JDBC Datasource. The file should have the following contents:

```
<datasources>
    <local-tx-datasource>
        <!-- This connection pool will be bound into JNDI with the name
             "java:/MySQLDB" -->

        <jndi-name>MySQLDB</jndi-name>
        <connection-url>jdbc:mysql://localhost:3306/dbname</connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>user</user-name>
        <password>pass</password>

        <min-pool-size>5</min-pool-size>

        <!-- Don't set this any higher than max_connections on your
         MySQL server, usually this should be a 10 or a few 10's
         of connections, not hundreds or thousands -->

        <max-pool-size>20</max-pool-size>

        <!-- Don't allow connections to hang out idle too long,
         never longer than what wait_timeout is set to on the
         server...A few minutes is usually okay here,
         it depends on your application
         and how much spikey load it will see -->

        <idle-timeout-minutes>5</idle-timeout-minutes>

        <!-- If you're using Connector/J 3.1.8 or newer, you can use
             our implementation of these to increase the robustness
```

```
            of the connection pool. -->

        <exception-sorter-class-name>com.mysql.jdbc.integration.jboss.ExtendedMysq
        <valid-connection-checker-class-name>com.mysql.jdbc.integration.jboss.Mysq

    </local-tx-datasource>
</datasources>
```

1.5. Diagnosing Connector/J Problems

This section describes how to solve problems that you may encounter when using Connector/J.

1.5.1. Common Problems and Solutions

There are a few issues that seem to be commonly encountered often by users of MySQL Connector/J. This section deals with their symptoms, and their resolutions.

1.5.1.1:

Question:

When I try to connect to the database with MySQL Connector/J, I get the following exception:

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

What's going on? I can connect just fine with the MySQL command-line client.

Answer:

MySQL Connector/J must use TCP/IP sockets to connect to MySQL, as Java does not support Unix Domain Sockets. Therefore, when MySQL Connector/J connects to MySQL, the security manager in MySQL server will use its grant tables to determine whether the connection should be allowed.

You must add grants to allow this to happen. The following is an example of how to do this (but not the most secure).

From the mysql command-line client, logged in as a user that can grant privileges, issue the following command:

```
GRANT ALL PRIVILEGES ON [dbname].* to
                '[user]'@'[hostname]' identified by
                '[password]'
```

replacing [dbname] with the name of your database, [user] with the user name, [hostname] with the host that MySQL Connector/J will be connecting from, and [password] with the password you want to use. Be aware that RedHat Linux is broken with respect to the hostname portion for the case when you are connecting from localhost. You need to use "localhost.localdomain" for the [hostname] value in this case. Follow this by issuing the "FLUSH PRIVILEGES" command.

Note

Testing your connectivity with the `mysql` command-line client will not work unless you add the `--host` flag, and use something other than `localhost` for the host. The `mysql` command-line client will use Unix domain sockets if you use the special hostname `localhost`. If

you are testing connectivity to `localhost`, use `127.0.0.1` as the hostname instead.

Warning

If you don't understand what the 'GRANT' command does, or how it works, you should read and understand the 'General Security Issues and the MySQL Access Privilege System' [http://www.mysql.com/doc/en/Privilege_system.html] section of the MySQL manual before attempting to change privileges.

Changing privileges and permissions improperly in MySQL can potentially cause your server installation to not have optimal security properties.

1.5.1.2:

Question:

My application throws an SQLException 'No Suitable Driver'. Why is this happening?

Answer:

One of two things are happening. Either the driver is not in your CLASSPATH or your URL format is incorrect (see the Section 1.2, “Installing Connector/J” section.).

1.5.1.3:

Question:

I'm trying to use MySQL Connector/J in an applet or application and I get an exception similar to:

```
SQLException: Cannot connect to MySQL server on host:3306.
Is there a MySQL server running on the machine/port you
are trying to connect to?

(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

Answer:

Either you're running an Applet, your MySQL server has been installed with the "--skip-networking" option set, or your MySQL server has a firewall sitting in front of it.

Applets can only make network connections back to the machine that runs the web server that served the .class files for the applet. This means that MySQL must run on the same machine (or you must have some sort of port re-direction) for this to work. This also means that you will not be able to test applets from your local file system, you must always deploy them to a web server.

MySQL Connector/J can only communicate with MySQL using TCP/IP, as Java does not support Unix domain sockets. TCP/IP communication with MySQL might be affected if MySQL was started with the "--skip-networking" flag, or if it is firewalled.

If MySQL has been started with the "--skip-networking" option set (the Debian Linux package of MySQL server does this for example), you need to comment it out in the file /etc/mysql/my.cnf or /etc/my.cnf. Of course your my.cnf file might also exist in the `data` directory of your MySQL server, or anywhere else (depending on how MySQL was compiled for your system). Binaries created by MySQL AB always look in /etc/my.cnf and [datadir]/my.cnf. If your MySQL server has been firewalled, you will need to have the firewall configured to allow TCP/IP connections from the host where your Java code is running to the MySQL server on the port that MySQL is listening to (by default, 3306).

1.5.1.4:

Question:

I have a servlet/application that works fine for a day, and then stops working overnight

Answer:

MySQL closes connections after 8 hours of inactivity. You either need to use a connection pool that handles stale connections or use the "autoReconnect" parameter (see Section 1.3.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”).

Also, you should be catching SQLExceptions in your application and dealing with them, rather than propagating them all the way until your application exits, this is just good programming practice. MySQL Connector/J will set the SQLState (see java.sql.SQLException.getSQLState() in your APIDOCS) to "08S01" when it encounters network-connectivity issues during the processing of a query. Your application code should then attempt to re-connect to MySQL at this point.

The following (simplistic) example shows what code that can handle these exceptions might look like:

Example 13. Example of transaction with retry logic

```
public void doBusinessOp() throws SQLException {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        //
        // How many times do you want to retry the transaction
        // (or at least _getting_ a connection)?
        //
        int retryCount = 5;

        boolean transactionCompleted = false;

        do {
            try {
                conn = getConnection(); // assume getting this from a
                                        // javax.sql.DataSource, or the
                                        // java.sql.DriverManager

                conn.setAutoCommit(false);

                //
                // Okay, at this point, the 'retry-ability' of the
                // transaction really depends on your application logic,
                // whether or not you're using autocommit (in this case
                // not), and whether you're using transacational storage
                // engines
                //
                // For this example, we'll assume that it's _not_ safe
                // to retry the entire transaction, so we set retry count
                // to 0 at this point
                //
                // If you were using exclusively transaction-safe tables,
                // or your application could recover from a connection going
                // bad in the middle of an operation, then you would not
                // touch 'retryCount' here, and just let the loop repeat
                // until retryCount == 0.
```

```
            //
            retryCount = 0;

            stmt = conn.createStatement();

            String query = "SELECT foo FROM bar ORDER BY baz";

            rs = stmt.executeQuery(query);

            while (rs.next()) {
            }

            rs.close();
            rs = null;

            stmt.close();
            stmt = null;

            conn.commit();
            conn.close();
            conn = null;

            transactionCompleted = true;
        } catch (SQLException sqlEx) {

            //
            // The two SQL states that are 'retry-able' are 08S01
            // for a communications error, and 40001 for deadlock.
            //
            // Only retry if the error was due to a stale connection,
            // communications problem or deadlock
            //

            String sqlState = sqlEx.getSQLState();

            if ("08S01".equals(sqlState) || "40001".equals(sqlState)) {
                retryCount--;
            } else {
                retryCount = 0;
            }
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException sqlEx) {
                    // You'd probably want to log this . . .
                }
            }

            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException sqlEx) {
                    // You'd probably want to log this as well . . .
                }
            }

            if (conn != null) {
                try {
                    //
                    // If we got here, and conn is not null, the
                    // transaction should be rolled back, as not
                    // all work has been done
```

```
                    try {
                        conn.rollback();
                    } finally {
                        conn.close();
                    }
                } catch (SQLException sqlEx) {
                    //
                    // If we got an exception here, something
                    // pretty serious is going on, so we better
                    // pass it up the stack, rather than just
                    // logging it. . .

                    throw sqlEx;
                }
            }
        }
    } while (!transactionCompleted && (retryCount > 0));
}
```

1.5.1.5:

Question:

I'm trying to use JDBC-2.0 updatable result sets, and I get an exception saying my result set is not updatable.

Answer:

Because MySQL does not have row identifiers, MySQL Connector/J can only update result sets that have come from queries on tables that have at least one primary key, the query must select every primary key and the query can only span one table (that is, no joins). This is outlined in the JDBC specification.

1.5.2. How to Report Connector/J Bugs or Problems

The normal place to report bugs is http://bugs.mysql.com/, which is the address for our bugs database. This database is public, and can be browsed and searched by anyone. If you log in to the system, you will also be able to enter new reports.

If you have found a sensitive security bug in MySQL, you can send email to security@mysql.com [mailto:security@mysql.com].

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release.

This section will help you write your report correctly so that you don't waste your time doing things that may not help us much or at all.

If you have a repeatable bug report, please report it to the bugs database at http://bugs.mysql.com/.

Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

To report other problems, you can use one of the MySQL mailing lists.

Remember that it is possible for us to respond to a message containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details don't matter.

A good principle is this: If you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of Connector/J or MySQL used, and (b) not fully describing the platform on which Connector/J is installed (including the JVM version, and the platform type and version number that MySQL itself is installed on).

This is highly relevant information, and in 99 cases out of 100, the bug report is useless without it. Very often we get questions like, ``Why doesn't this work for me?'' Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has already been fixed in newer MySQL versions.

Sometimes the error is platform-dependent; in such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If at all possible, you should create a repeatable, stanalone testcase that doesn't involve any third-party classes.

To streamline this process, we ship a base class for testcases with Connector/J, named '`com.mysql.jdbc.util.BaseBugReport`'. To create a testcase for Connector/J using this class, create your own class that inherits from `com.mysql.jdbc.util.BaseBugReport` and override the methods `setUp()`, `tearDown()` and `runTest` ().

In the `setUp()` method, create code that creates your tables, and populates them with any data needed to demonstrate the bug.

In the `runTest` () method, create code that demonstrates the bug using the tables and data you created in the `setUp` method.

In the `tearDown()` method, drop any tables you created in the `setUp()` method.

In any of the above three methods, you should use one of the variants of the `getConnection` () method to create a JDBC connection to MySQL:

- getConnection() - Provides a connection to the JDBC URL specified in getUrl(). If a connection already exists, that connection is returned, otherwise a new connection is created.
- getNewConnection() - Use this if you need to get a new connection for your bug report (i.e. there's more than one connection involved).
- getConnection(String url) - Returns a connection using the given URL.
- getConnection(String url, Properties props) - Returns a connection using the given URL and properties.

If you need to use a JDBC URL that is different from 'jdbc:mysql:///test', override the method `getUrl()` as well.

Use the `assertTrue(boolean expression)` and `assertTrue(String failureMessage, boolean expression)` methods to create conditions that must be met in your testcase demonstrating the behavior you are expecting (vs. the behavior you are observing, which is why you are most likely filing a bug report).

Finally, create a `main` () method that creates a new instance of your testcase, and calls the `run` method:

```
public static void main(String[] args) throws Exception {
```

```
        new MyBugReport().run();
}
```

Once you have finished your testcase, and have verified that it demonstrates the bug you are reporting, upload it with your bug report to http://bugs.mysql.com/.

1.6. MySQL Connector/J Change History

1.6.1. Changes in MySQL Connector/J 5.0.0-beta (22 December 2005)

- `XADataSource` implemented (ported from 3.2 branch which won't be released as a product). Use `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` as your datasource class name in your application server to utilize XA transactions in MySQL-5.0.10 and newer.
- `PreparedStatement.setString()` didn't work correctly when `sql_mode` on server contained `NO_BACKSLASH_ESCAPES` and no characters that needed escaping were present in the string.
- Attempt detection of the MySQL type `BINARY` (it's an alias, so this isn't always reliable), and use the `java.sql.Types.BINARY` type mapping for it.
- Moved `-bin-g.jar` file into separate `debug` subdirectory to avoid confusion.
- Don't allow `.setAutoCommit(true)`, or `.commit()` or `.rollback()` on an XA-managed connection as per the JDBC specification.
- If the connection `useTimezone` is set to `true`, then also respect time zone conversions in escape-processed string literals (for example, `"{ts ...}"` and `"{t ...}"`).
- Return original column name for `RSMD.getColumnName()` if the column was aliased, alias name for `.getColumnLabel()` (if aliased), and original table name for `.getTableName()`. Note this only works for MySQL-4.1 and newer, as older servers don't make this information available to clients.
- Setting `useJDBCCompliantTimezoneShift=true` (it's not the default) causes the driver to use GMT for *all* `TIMESTAMP`/`DATETIME` time zones, and the current VM time zone for any other type that refers to time zones. This feature can not be used when `useTimezone=true` to convert between server and client time zones.
- Add one level of indirection of internal representation of `CallableStatement` parameter metadata to avoid class not found issues on JDK-1.3 for `ParameterMetadata` interface (which doesn't exist prior to JDBC-3.0).
- Added unit tests for `XADatasource`, as well as friendlier exceptions for XA failures compared to the "stock" `XAException` (which has no messages).
- Idle timeouts cause `XAConnections` to whine about rolling themselves back. (Bug#14729 [http://bugs.mysql.com/14729])
- Added support for Connector/MXJ integration via url subprotocol `jdbc:mysql:mxj://...`.
- Moved all `SQLException` constructor usage to a factory in `SQLError` (ground-work for JDBC-4.0 `SQLState`-based exception classes).
- Removed Java5-specific calls to `BigDecimal` constructor (when result set value is `''`, `(int)0`

was being used as an argument indirectly via method return value. This signature doesn't exist prior to Java5.)

- Added service-provider entry to `META-INF/services/java.sql.Driver` for JDBC-4.0 support.
- Return "[VAR]BINARY" for `RSMD.getColumnTypeName()` when that is actually the type, and it can be distinguished (MySQL-4.1 and newer).
- When fix for Bug#14562 [http://bugs.mysql.com/14562] was merged from 3.1.12, added functionality for `CallableStatement`'s parameter metadata to return correct information for `.getParameterClassName()`.
- Fuller synchronization of `Connection` to avoid deadlocks when using multithreaded frameworks that multithread a single connection (usually not recommended, but the JDBC spec allows it anyways), part of fix to Bug#14972 [http://bugs.mysql.com/14972]).
- Implementation of `Statement.cancel()` and `Statement.setQueryTimeout()`. Both require MySQL-5.0.0 or newer server, require a separate connection to issue the `KILL QUERY` statement, and in the case of `setQueryTimeout()` creates an additional thread to handle the timeout functionality.

  Note: Failures to cancel the statement for `setQueryTimeout()` may manifest themselves as `RuntimeExceptions` rather than failing silently, as there is currently no way to unblock the thread that is executing the query being cancelled due to timeout expiration and have it throw the exception instead.

1.6.2. Changes in MySQL Connector/J 3.1.13 (xx xxx 2005)

- `INOUT` parameter does not store `IN` value. (Bug#15464 [http://bugs.mysql.com/15464])
- Exception thrown for new decimal type when using updatable result sets. (Bug#14609 [http://bugs.mysql.com/14609])
- No "dos" character set in MySQL > 4.1.0. (Bug#15544 [http://bugs.mysql.com/15544])
- `PreparedStatement.setObject()` serializes `BigInteger` as object, rather than sending as numeric value (and is thus not complementary to `.getObject()` on an `UNSIGNED LONG` type). (Bug#15383 [http://bugs.mysql.com/15383])
- `ResultSet.getShort()` for `UNSIGNED TINYINT` returned wrong values. (Bug#11874 [http://bugs.mysql.com/11874])
- `lib-nodist` directory missing from package breaks out-of-box build. (Bug#15676 [http://bugs.mysql.com/15676])
- `DBMD.getColumns()` returns wrong type for `BIT`. (Bug#15854 [http://bugs.mysql.com/15854])

1.6.3. Changes in MySQL Connector/J 3.1.12 (30 November 2005)

- Fixed client-side prepared statement bug with embedded `?` characters inside quoted identifiers (it was recognized as a placeholder, when it was not).
- Don't allow `executeBatch()` for `CallableStatements` with registered `OUT`/`INOUT` para-

meters (JDBC compliance).

- Fall back to platform-encoding for `URLDecoder.decode()` when parsing driver URL properties if the platform doesn't have a two-argument version of this method.
- Java type conversion may be incorrect for `MEDIUMINT`. (Bug#14562 [http://bugs.mysql.com/14562])
- Added configuration property `useGmtMillisForDatetimes` which when set to `true` causes `ResultSet.getDate()`, `.getTimestamp()` to return correct millis-since GMT when `.getTime()` is called on the return value (currently default is `false` for legacy behavior).
- Fixed `DatabaseMetaData.stores*Identifiers()`:
  - If `lower_case_table_names=0` (on server):
    - `storesLowerCaseIdentifiers()` returns `false`
    - `storesLowerCaseQuotedIdentifiers()` returns `false`
    - `storesMixedCaseIdentifiers()` returns `true`
    - `storesMixedCaseQuotedIdentifiers()` returns `true`
    - `storesUpperCaseIdentifiers()` returns `false`
    - `storesUpperCaseQuotedIdentifiers()` returns `true`
  - If `lower_case_table_names=1` (on server):
    - `storesLowerCaseIdentifiers()` returns `true`
    - `storesLowerCaseQuotedIdentifiers()` returns `true`
    - `storesMixedCaseIdentifiers()` returns `false`
    - `storesMixedCaseQuotedIdentifiers()` `returns false`
    - `storesUpperCaseIdentifiers()` returns `false`
    - `storesUpperCaseQuotedIdentifiers()` returns `true`
- `DatabaseMetaData.getColumns()` doesn't return `TABLE_NAME` correctly. (Bug#14815 [http://bugs.mysql.com/14815])
- Escape processor replaces quote character in quoted string with string delimiter. (Bug#14909 [http://bugs.mysql.com/14909])
- OpenOffice expects `DBMD.supportsIntegrityEnhancementFacility()` to return `true` if foreign keys are supported by the datasource, even though this method also covers support for check constraints, which MySQL *doesn't* have. Setting the configuration property `overrideSupportsIntegrityEnhancementFacility` to `true` causes the driver to return `true` for this method. (Bug#12975 [http://bugs.mysql.com/12975])
- Added `com.mysql.jdbc.testsuite.url.default` system property to set default JDBC url for testsuite (to speed up bug resolution when I'm working in Eclipse).
- Unable to initialize character set mapping tables (due to J2EE classloader differences). (Bug#14938 [http://bugs.mysql.com/14938])

- Deadlock while closing server-side prepared statements from multiple threads sharing one connection. (Bug#14972 [http://bugs.mysql.com/14972])
- `logSlowQueries` should give better info. (Bug#12230 [http://bugs.mysql.com/12230])
- Extraneous sleep on `autoReconnect`. (Bug#13775 [http://bugs.mysql.com/13775])
- Driver incorrectly closes streams passed as arguments to `PreparedStatements`. Reverts to legacy behavior by setting the JDBC configuration property `autoClosePStmtStreams` to `true` (also included in the 3-0-Compat configuration "bundle"). (Bug#15024 [http://bugs.mysql.com/15024])
- `maxQuerySizeToLog` is not respected. Added logging of bound values for `execute()` phase of server-side prepared statements when `profileSQL=true` as well. (Bug#13048 [http://bugs.mysql.com/13048])
- Usage advisor complains about unreferenced columns, even though they've been referenced. (Bug#15065 [http://bugs.mysql.com/15065])
- Don't increase timeout for failover/reconnect. (Bug#6577 [http://bugs.mysql.com/6577])
- Process escape tokens in `Connection.prepareStatement(...)`. (Bug#15141 [http://bugs.mysql.com/15141]) You can disable this behavior by setting the JDBC URL configuration property `processEscapeCodesForPrepStmts` to `false`.
- Reconnect during middle of `executeBatch()` should not occur if `autoReconnect` is enabled. (Bug#13255 [http://bugs.mysql.com/13255])

1.6.4. Changes in MySQL Connector/J 3.1.11-stable (07 October 2005)

- Spurious ! on console when character encoding is `utf8`. (Bug#11629 [http://bugs.mysql.com/11629])
- Fixed statements generated for testcases missing `;` for "plain" statements.
- Incorrect generation of testcase scripts for server-side prepared statements. (Bug#11663 [http://bugs.mysql.com/11663])
- Fixed regression caused by fix for Bug#11552 [http://bugs.mysql.com/11552] that caused driver to return incorrect values for unsigned integers when those integers where within the range of the positive signed type.
- Moved source code to Subversion repository.
- Escape tokenizer doesn't respect stacked single quotes for escapes. (Bug#11797 [http://bugs.mysql.com/11797])
- `GEOMETRY` type not recognized when using server-side prepared statements.
- `ReplicationConnection` won't switch to slave, throws "Catalog can't be null" exception. (Bug#11879 [http://bugs.mysql.com/11879])
- Properties shared between master and slave with replication connection. (Bug#12218 [http://bugs.mysql.com/12218])

- `Statement.getWarnings()` fails with NPE if statement has been closed. (Bug#10630 [http://bugs.mysql.com/10630])
- Only get `char[]` from SQL in `PreparedStatement.ParseInfo()` when needed.
- Geometry types not handled with server-side prepared statements. (Bug#12104 [http://bugs.mysql.com/12104])
- `StringUtils.getBytes()` doesn't work when using multi-byte character encodings and a length in *characters* is specified. (Bug#11614 [http://bugs.mysql.com/11614])
- `Pstmt.setObject(...., Types.BOOLEAN)` throws exception. (Bug#11798 [http://bugs.mysql.com/11798])
- `maxPerformance.properties` mis-spells “elideSetAutoCommits”. (Bug#11976 [http://bugs.mysql.com/11976])
- `DBMD.storesLower/Mixed/UpperIdentifiers()` reports incorrect values for servers deployed on Windows. (Bug#11575 [http://bugs.mysql.com/11575])
- `ResultSet.moveToCurrentRow()` fails to work when preceded by a call to `ResultSet.moveToInsertRow()`. (Bug#11190 [http://bugs.mysql.com/11190])
- `VARBINARY` data corrupted when using server-side prepared statements and `.setBytes()`. (Bug#11115 [http://bugs.mysql.com/11115])
- `explainSlowQueries` hangs with server-side prepared statements. (Bug#12229 [http://bugs.mysql.com/12229])
- Escape processor didn't honor strings demarcated with double quotes. (Bug#11498 [http://bugs.mysql.com/11498])
- Lifted restriction of changing streaming parameters with server-side prepared statements. As long as `all` streaming parameters were set before execution, `.clearParameters()` does not have to be called. (due to limitation of client/server protocol, prepared statements can not reset *individual* stream data on the server side).
- Reworked `Field` class, `*Buffer`, and `MysqlIO` to be aware of field lengths > `Integer.MAX_VALUE`.
- Updated `DBMD.supportsCorrelatedQueries()` to return `true` for versions > 4.1, `supportsGroupByUnrelated()` to return `true` and `getResultSetHoldability()` to return `HOLD_CURSORS_OVER_COMMIT`.
- Handling of catalog argument in `DatabaseMetaData.getIndexInfo()`, which also means changes to the following methods in `DatabaseMetaData`: (Bug#12541 [http://bugs.mysql.com/12541])
  - `getBestRowIdentifier()`
  - `getColumns()`
  - `getCrossReference()`
  - `getExportedKeys()`
  - `getImportedKeys()`
  - `getIndexInfo()`

  - `getPrimaryKeys()`
  - `getProcedures()` (and thus indirectly `getProcedureColumns()`)
  - `getTables()`

  The `catalog` argument in all of these methods now behaves in the following way:

  - Specifying `NULL` means that catalog will not be used to filter the results (thus all databases will be searched), unless you've set `nullCatalogMeansCurrent=true` in your JDBC URL properties.
  - Specifying `""` means “current” catalog, even though this isn't quite JDBC spec compliant, it's there for legacy users.
  - Specifying a catalog works as stated in the API docs.
  - Made `Connection.clientPrepare()` available from “wrapped” connections in the `jdbc2.optional` package (connections built by `ConnectionPoolDataSource` instances).

- Added `Connection.isMasterConnection()` for clients to be able to determine if a multi-host master/slave connection is connected to the first host in the list.
- Tokenizer for `=` in URL properties was causing `sessionVariables=....` to be parameterized incorrectly. (Bug#12753 [http://bugs.mysql.com/12753])
- Foreign key information that is quoted is parsed incorrectly when `DatabaseMetaData` methods use that information. (Bug#11781 [http://bugs.mysql.com/11781])
- The `sendBlobChunkSize` property is now clamped to `max_allowed_packet` with consideration of stream buffer size and packet headers to avoid `PacketTooBigExceptions` when `max_allowed_packet` is similar in size to the default `sendBlobChunkSize` which is 1M.
- `CallableStatement.clearParameters()` now clears resources associated with `INOUT`/`OUTPUT` parameters as well as `INPUT` parameters.
- `Connection.prepareCall()` is database name case-sensitive (on Windows systems). (Bug#12417 [http://bugs.mysql.com/12417])
- `cp1251` incorrectly mapped to `win1251` for servers newer than 4.0.x. (Bug#12752 [http://bugs.mysql.com/12752])
- `java.sql.Types.OTHER` returned for `BINARY` and `VARBINARY` columns when using `DatabaseMetaData.getColumns()`. (Bug#12970 [http://bugs.mysql.com/12970])
- `ServerPreparedStatement.getBinding()` now checks if the statement is closed before attempting to reference the list of parameter bindings, to avoid throwing a `NullPointerException`.
- `ResultSetMetaData` from `Statement.getGeneratedKeys()` caused a `NullPointerException` to be thrown whenever a method that required a connection reference was called. (Bug#13277 [http://bugs.mysql.com/13277])
- Backport of `Field` class, `ResultSetMetaData.getColumnClassName()`, and `ResultSet.getObject(int)` changes from 5.0 branch to fix behavior surrounding `VARCHAR BINARY`/`VARBINARY` and related types.
- Fixed `NullPointerException` when converting `catalog` parameter in many `Database-`

`MetaDataMethods` to `byte[]`s (for the result set) when the parameter is `null`. (`null` isn't technically allowed by the JDBC specification, but we've historically allowed it).

- Backport of `VAR[BINARY|CHAR] [BINARY]` types detection from 5.0 branch.
- Read response in `MysqlIO.sendFileToServer()`, even if the local file can't be opened, otherwise next query issued will fail, because it's reading the response to the empty `LOAD DATA INFILE` packet sent to the server.
- Workaround for Bug#13374 [http://bugs.mysql.com/13374]: `ResultSet.getStatement()` on closed result set returns `NULL` (as per JDBC 4.0 spec, but not backward-compatible). Set the connection property `retainStatementAfterResultSetClose` to `true` to be able to retrieve a `ResultSet`'s statement after the `ResultSet` has been closed via `.getStatement()` (the default is `false`, to be JDBC-compliant and to reduce the chance that code using JDBC leaks `Statement` instances).
- URL configuration parameters don't allow '`&`' or '`=`' in their values. The JDBC driver now parses configuration parameters as if they are encoded using the application/x-www-form-urlencoded format as specified by `java.net.URLDecoder` (http://java.sun.com/j2se/1.5.0/docs/api/java/net/URLDecoder.html). (Bug#13453 [http://bugs.mysql.com/13453])

  If the '`%`' character is present in a configuration property, it must now be represented as `%25`, which is the encoded form of '`%`' when using application/x-www-form-urlencoded encoding.
- The configuration property `sessionVariables` now allows you to specify variables that start with the '`@`' sign.
- When `gatherPerfMetrics` is enabled for servers older than 4.1.0, a `NullPointerException` is thrown from the constructor of `ResultSet` if the query doesn't use any tables. (Bug#13043 [http://bugs.mysql.com/13043])

1.6.5. Changes in MySQL Connector/J 3.1.10-stable (23 June 2005)

- Fixed connecting without a database specified raised an exception in `MysqlIO.changeDatabaseTo()`.
- Initial implemention of `ParameterMetadata` for `PreparedStatement.getParameterMetadata()`. Only works fully for `CallableStatements`, as current server-side prepared statements return every parameter as a `VARCHAR` type.

1.6.6. Changes in MySQL Connector/J 3.1.9-stable (22 June 2005)

- Overhaul of character set configuration, everything now lives in a properties file.
- Driver now correctly uses CP932 if available on the server for Windows-31J, CP932 and MS932 java encoding names, otherwise it resorts to SJIS, which is only a close approximation. Currently only MySQL-5.0.3 and newer (and MySQL-4.1.12 or .13, depending on when the character set gets backported) can reliably support any variant of CP932.
- `com.mysql.jdbc.PreparedStatement.ParseInfo` does unnecessary call to `toCharArray()`. (Bug#9064 [http://bugs.mysql.com/9064])
- Memory leak in `ServerPreparedStatement` if `serverPrepare()` fails. (Bug#10144

[http://bugs.mysql.com/10144])

- Actually write manifest file to correct place so it ends up in the binary jar file.

- Added `createDatabaseIfNotExist` property (default is `false`), which will cause the driver to ask the server to create the database specified in the URL if it doesn't exist. You must have the appropriate privileges for database creation for this to work.

- Unsigned `SMALLINT` treated as signed for `ResultSet.getInt()`, fixed all cases for `UNSIGNED` integer values and server-side prepared statements, as well as `ResultSet.getObject()` for `UNSIGNED TINYINT`. (Bug#10156 [http://bugs.mysql.com/10156])

- Double quotes not recognized when parsing client-side prepared statements. (Bug#10155 [http://bugs.mysql.com/10155])

- Made `enableStreamingResults()` visible on `com.mysql.jdbc.jdbc2.optional.StatementWrapper`.

- Made `ServerPreparedStatement.asSql()` work correctly so auto-explain functionality would work with server-side prepared statements.

- Made JDBC2-compliant wrappers public in order to allow access to vendor extensions.

- Cleaned up logging of profiler events, moved code to dump a profiler event as a string to `com.mysql.jdbc.log.LogUtils` so that third parties can use it.

- `DatabaseMetaData.supportsMultipleOpenResults()` now returns `true`. The driver has supported this for some time, DBMD just missed that fact.

- Driver doesn't support `{?=CALL(...)}` for calling stored functions. This involved adding support for function retrieval to `DatabaseMetaData.getProcedures()` and `getProcedureColumns()` as well. (Bug#10310 [http://bugs.mysql.com/10310])

- `SQLException` thrown when retrieving `YEAR(2)` with `ResultSet.getString()`. The driver will now always treat `YEAR` types as `java.sql.Dates` and return the correct values for `getString()`. Alternatively, the `yearIsDateType` connection property can be set to `false` and the values will be treated as `SHORT`s. (Bug#10485 [http://bugs.mysql.com/10485])

- The datatype returned for `TINYINT(1)` columns when `tinyInt1isBit=true` (the default) can be switched between `Types.BOOLEAN` and `Types.BIT` using the new configuration property `transformedBitIsBoolean`, which defaults to `false`. If set to `false` (the default), `DatabaseMetaData.getColumns()` and `ResultSetMetaData.getColumnType()` will return `Types.BOOLEAN` for `TINYINT(1)` columns. If `true`, `Types.BOOLEAN` will be returned instead. Regardless of this configuration property, if `tinyInt1isBit` is enabled, columns with the type `TINYINT(1)` will be returned as `java.lang.Boolean` instances from `ResultSet.getObject(...)`, and `ResultSetMetaData.getColumnClassName()` will return `java.lang.Boolean`.

- `SQLException` is thrown when using property `characterSetResults` with `cp932` or `eucjpms`. (Bug#10496 [http://bugs.mysql.com/10496])

- Reorganized directory layout. Sources now are in `src` folder. Don't pollute parent directory when building, now output goes to `./build`, distribution goes to `./dist`.

- Added support/bug hunting feature that generates `.sql` test scripts to `STDERR` when `autoGenerateTestcaseScript` is set to `true`.

- 0-length streams not sent to server when using server-side prepared statements. (Bug#10850 [http://bugs.mysql.com/10850])

- Setting `cachePrepStmts=true` now causes the `Connection` to also cache the check the driver performs to determine if a prepared statement can be server-side or not, as well as caches server-side prepared statements for the lifetime of a connection. As before, the `prepStmtCacheSize` parameter controls the size of these caches.

- Try to handle `OutOfMemoryErrors` more gracefully. Although not much can be done, they will in most cases close the connection they happened on so that further operations don't run into a connection in some unknown state. When an OOM has happened, any further operations on the connection will fail with a “Connection closed” exception that will also list the OOM exception as the reason for the implicit connection close event.

- Don't send `COM_RESET_STMT` for each execution of a server-side prepared statement if it isn't required.

- Driver detects if you're running MySQL-5.0.7 or later, and does not scan for `LIMIT ?[,?]` in statements being prepared, as the server supports those types of queries now.

- `VARBINARY` data corrupted when using server-side prepared statements and `ResultSet.getBytes()`. (Bug#11115 [http://bugs.mysql.com/11115])

- `Connection.setCatalog()` is now aware of the `useLocalSessionState` configuration property, which when set to `true` will prevent the driver from sending `USE ...` to the server if the requested catalog is the same as the current catalog.

- Added the following configuration bundles, use one or many via the `useConfigs` configuration property:

  - `maxPerformance` — maximum performance without being reckless

  - `solarisMaxPerformance` — maximum performance for Solaris, avoids syscalls where it can

  - `3-0-Compat` — Compatibility with Connector/J 3.0.x functionality

- Added `maintainTimeStats` configuration property (defaults to `true`), which tells the driver whether or not to keep track of the last query time and the last successful packet sent to the server's time. If set to `false`, removes two syscalls per query.

- `autoReconnect` ping causes exception on connection startup. (Bug#11259 [http://bugs.mysql.com/11259])

- Connector/J dumping query into `SQLException` twice. (Bug#11360 [http://bugs.mysql.com/11360])

- Fixed `PreparedStatement.setClob()` not accepting `null` as a parameter.

- Production package doesn't include JBoss integration classes. (Bug#11411 [http://bugs.mysql.com/11411])

- Removed nonsensical “costly type conversion” warnings when using usage advisor.

1.6.7. Changes in MySQL Connector/J 3.1.8-stable (14 April 2005)

- Fixed `DatabaseMetaData.getTables()` returning views when they were not asked for as one of the requested table types.

- Added support for new precision-math `DECIMAL` type in MySQL 5.0.3 and up.

- Fixed `ResultSet.getTime()` on a `NULL` value for server-side prepared statements throws NPE.
- Made `Connection.ping()` a public method.
- `DATE_FORMAT()` queries returned as `BLOB`s from `getObject()`. (Bug#8868 [http://bugs.mysql.com/8868])
- `ServerPreparedStatements` now correctly "stream" `BLOB`/`CLOB` data to the server. You can configure the threshold chunk size using the JDBC URL property `blobSendChunkSize` (the default is 1MB).
- `BlobFromLocator` now uses correct identifier quoting when generating prepared statements.
- Server-side session variables can be preset at connection time by passing them as a comma-delimited list for the connection property `sessionVariables`.
- Fixed regression in `ping()` for users using `autoReconnect=true`.
- `PreparedStatement.addBatch()` doesn't work with server-side prepared statements and streaming `BINARY` data. (Bug#9040 [http://bugs.mysql.com/9040])
- `DBMD.supportsMixedCase*Identifiers()` returns wrong value on servers running on case-sensitive filesystems. (Bug#8800 [http://bugs.mysql.com/8800])
- Cannot use `UTF-8` for characterSetResults configuration property. (Bug#9206 [http://bugs.mysql.com/9206])
- A continuation of Bug#8868 [http://bugs.mysql.com/8868], where functions used in queries that should return non-string types when resolved by temporary tables suddenly become opaque binary strings (work-around for server limitation). Also fixed fields with type of `CHAR(n) CHARACTER SET BINARY` to return correct/matching classes for `RSMD.getColumnClassName()` and `ResultSet.getObject()`. (Bug#9236 [http://bugs.mysql.com/9236])
- `DBMD.supportsResultSetConcurrency()` not returning `true` for forward-only/read-only result sets (we obviously support this). (Bug#8792 [http://bugs.mysql.com/8792])
- `DATA_TYPE` column from `DBMD.getBestRowIdentifier()` causes `ArrayIndexOutOfBoundsException` when accessed (and in fact, didn't return any value). (Bug#8803 [http://bugs.mysql.com/8803])
- Check for empty strings (`''`) when converting `CHAR`/`VARCHAR` column data to numbers, throw exception if `emptyStringsConvertToZero` configuration property is set to `false` (for backward-compatibility with 3.0, it is now set to `true` by default, but will most likely default to `false` in 3.2).
- `PreparedStatement.getMetaData()` inserts blank row in database under certain conditions when not using server-side prepared statements. (Bug#9320 [http://bugs.mysql.com/9320])
- `Connection.canHandleAsPreparedStatement()` now makes "best effort" to distinguish `LIMIT` clauses with placeholders in them from ones without in order to have fewer false positives when generating work-arounds for statements the server cannot currently handle as server-side prepared statements.
- Fixed `build.xml` to not compile `log4j` logging if `log4j` not available.
- Added support for the c3p0 connection pool's (http://c3p0.sf.net/) validation/connection checker interface which uses the lightweight `COM_PING` call to the server if available. To use it, configure your c3p0 connection pool's `connectionTesterClassName` property to use

`com.mysql.jdbc.integration.c3p0.MysqlConnectionTester`.

- Better detection of `LIMIT` inside/outside of quoted strings so that the driver can more correctly determine whether a prepared statement can be prepared on the server or not.
- Stored procedures with same name in different databases confuse the driver when it tries to determine parameter counts/types. (Bug#9319 [http://bugs.mysql.com/9319])
- Added finalizers to `ResultSet` and `Statement` implementations to be JDBC spec-compliant, which requires that if not explicitly closed, these resources should be closed upon garbage collection.
- Stored procedures with `DECIMAL` parameters with storage specifications that contained '`,`' in them would fail. (Bug#9682 [http://bugs.mysql.com/9682])
- `PreparedStatement.setObject(int, Object, int type, int scale)` now uses scale value for `BigDecimal` instances.
- `Statement.getMoreResults()` could throw NPE when existing result set was `.close()`d. (Bug#9704 [http://bugs.mysql.com/9704])
- The performance metrics feature now gathers information about number of tables referenced in a SELECT.
- The logging system is now automatically configured. If the value has been set by the user, via the URL property `logger` or the system property `com.mysql.jdbc.logger`, then use that, otherwise, autodetect it using the following steps:
  1. Log4j, if it's available,
  2. Then JDK1.4 logging,
  3. Then fallback to our `STDERR` logging.
- `DBMD.getTables()` shouldn't return tables if views are asked for, even if the database version doesn't support views. (Bug#9778 [http://bugs.mysql.com/9778])
- Fixed driver not returning `true` for `-1` when `ResultSet.getBoolean()` was called on result sets returned from server-side prepared statements.
- Added a `Manifest.MF` file with implementation information to the `.jar` file.
- More tests in `Field.isOpaqueBinary()` to distinguish opaque binary (that is, fields with type `CHAR(n)` and `CHARACTER SET BINARY`) from output of various scalar and aggregate functions that return strings.
- Should accept `null` for catalog (meaning use current) in DBMD methods, even though it's not JDBC-compliant for legacy's sake. Disable by setting connection property `nullCatalogMeansCurrent` to `false` (which will be the default value in C/J 3.2.x). (Bug#9917 [http://bugs.mysql.com/9917])
- Should accept `null` for name patterns in DBMD (meaning '`%`'), even though it isn't JDBC compliant, for legacy's sake. Disable by setting connection property `nullNamePatternMatchesAll` to `false` (which will be the default value in C/J 3.2.x). (Bug#9769 [http://bugs.mysql.com/9769])

1.6.8. Changes in MySQL Connector/J 3.1.7-stable (18 February 2005)

- Timestamp key column data needed `_binary` stripped for `UpdatableResultSet.refreshRow()`. (Bug#7686 [http://bugs.mysql.com/7686])
- Timestamps converted incorrectly to strings with server-side prepared statements and updatable result sets. (Bug#7715 [http://bugs.mysql.com/7715])
- Detect new `sql_mode` variable in string form (it used to be integer) and adjust quoting method for strings appropriately.
- Added `holdResultsOpenOverStatementClose` property (default is `false`), that keeps result sets open over statement.close() or new execution on same statement (suggested by Kevin Burton).
- Infinite recursion when “falling back” to master in failover configuration. (Bug#7952 [http://bugs.mysql.com/7952])
- Disable multi-statements (if enabled) for MySQL-4.1 versions prior to version 4.1.10 if the query cache is enabled, as the server returns wrong results in this configuration.
- Fixed duplicated code in `configureClientCharset()` that prevented `useOldUTF8Behavior=true` from working properly.
- Removed `dontUnpackBinaryResults` functionality, the driver now always stores results from server-side prepared statements as is from the server and unpacks them on demand.
- Emulated locators corrupt binary data when using server-side prepared statements. (Bug#8096 [http://bugs.mysql.com/8096])
- Fixed synchronization issue with `ServerPreparedStatement.serverPrepare()` that could cause deadlocks/crashes if connection was shared between threads.
- By default, the driver now scans SQL you are preparing via all variants of `Connection.prepareStatement()` to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this by passing `emulateUnsupportedPstmts=false` in your JDBC URL. (Bug#4718 [http://bugs.mysql.com/4718])
- Remove `_binary` introducer from parameters used as in/out parameters in `CallableStatement`.
- Always return `byte[]`s for output parameters registered as `*BINARY`.
- Send correct value for “boolean” `true` to server for `PreparedStatement.setObject(n, "true", Types.BIT)`.
- Fixed bug with Connection not caching statements from `prepareStatement()` when the statement wasn't a server-side prepared statement.
- Choose correct “direction” to apply time adjustments when both client and server are in GMT time zone when using `ResultSet.get(..., cal)` and `PreparedStatement.set(...., cal)`.
- Added `dontTrackOpenResources` option (default is `false`, to be JDBC compliant), which helps with memory use for non-well-behaved apps (that is, applications that don't close `Statement` objects when they should).
- `ResultSet.getString()` doesn't maintain format stored on server, bug fix only enabled when `noDatetimeStringSync` property is set to `true` (the default is `false`). (Bug#8428 [http://bugs.mysql.com/8428])

- Fixed NPE in `ResultSet.realClose()` when using usage advisor and result set was already closed.
- `PreparedStatements` not creating streaming result sets. (Bug#8478 [http://bugs.mysql.com/8478])
- Don't pass `NULL` to `String.valueOf()` in `ResultSet.getNativeConvertToString()`, as it stringifies it (that is, returns `null`), which is not correct for the method in question.
- `ResultSet.getBigDecimal()` throws exception when rounding would need to occur to set scale. The driver now chooses a rounding mode of "half up" if non-rounding `BigDecimal.setScale()` fails. (Bug#8484 [http://bugs.mysql.com/8484])
- Added `useLocalSessionState` configuration property, when set to `true` the JDBC driver trusts that the application is well-behaved and only sets autocommit and transaction isolation levels using the methods provided on `java.sql.Connection`, and therefore can manipulate these values in many cases without incurring round-trips to the database server.
- Added `enableStreamingResults()` to `Statement` for connection pool implementations that check `Statement.setFetchSize()` for specification-compliant values. Call `Statement.setFetchSize(>=0)` to disable the streaming results for that statement.
- Added support for `BIT` type in MySQL-5.0.3. The driver will treat `BIT(1-8)` as the JDBC standard `BIT` type (which maps to `java.lang.Boolean`), as the server does not currently send enough information to determine the size of a bitfield when < 9 bits are declared. `BIT(>9)` will be treated as `VARBINARY`, and will return `byte[]` when `getObject()` is called.

1.6.9. Changes in MySQL Connector/J 3.1.6-stable (23 December 2004)

- Fixed hang on `SocketInputStream.read()` with `Statement.setMaxRows()` and multiple result sets when driver has to truncate result set directly, rather than tacking a `LIMIT` *n* on the end of it.
- `DBMD.getProcedures()` doesn't respect catalog parameter. (Bug#7026 [http://bugs.mysql.com/7026])

1.6.10. Changes in MySQL Connector/J 3.1.5-gamma (02 December 2004)

- Fix comparisons made between string constants and dynamic strings that are converted with either `toUpperCase()` or `toLowerCase()` to use `Locale.ENGLISH`, as some locales "override" case rules for English. Also use `StringUtils.indexOfIgnoreCase()` instead of `.toUpperCase().indexOf()`, avoids creating a very short-lived transient `String` instance.
- Server-side prepared statements did not honor `zeroDateTimeBehavior` property, and would cause class-cast exceptions when using `ResultSet.getObject()`, as the all-zero string was always returned. (Bug#5235 [http://bugs.mysql.com/5235])
- Fixed batched updates with server prepared statements weren't looking if the types had changed for a given batched set of parameters compared to the previous set, causing the server to return the error "Wrong arguments to mysql_stmt_execute()".

- Handle case when string representation of timestamp contains trailing '.' with no numbers following it.
- Inefficient detection of pre-existing string instances in `ResultSet.getNativeString()`. (Bug#5706 [http://bugs.mysql.com/5706])
- Don't throw exceptions for `Connection.releaseSavepoint()`.
- Use a per-session `Calendar` instance by default when decoding dates from `ServerPreparedStatements` (set to old, less performant behavior by setting property `dynamicCalendars=true`).
- Added experimental configuration property `dontUnpackBinaryResults`, which delays unpacking binary result set values until they're asked for, and only creates object instances for non-numerical values (it is set to `false` by default). For some usecase/jvm combinations, this is friendlier on the garbage collector.
- `UNSIGNED BIGINT` unpacked incorrectly from server-side prepared statement result sets. (Bug#5729 [http://bugs.mysql.com/5729])
- `ServerSidePreparedStatement` allocating short-lived objects unnecessarily. (Bug#6225 [http://bugs.mysql.com/6225])
- Removed unwanted new `Throwable()` in `ResultSet` constructor due to bad merge (caused a new object instance that was never used for every result set created). Found while profiling for Bug#6359 [http://bugs.mysql.com/6359].
- Fixed too-early creation of `StringBuffer` in `EscapeProcessor.escapeSQL()`, also return `String` when escaping not needed (to avoid unnecessary object allocations). Found while profiling for Bug#6359 [http://bugs.mysql.com/6359].
- Use null-safe-equals for key comparisons in updatable result sets.
- `SUM()` on `DECIMAL` with server-side prepared statement ignores scale if zero-padding is needed (this ends up being due to conversion to `DOUBLE` by server, which when converted to a string to parse into `BigDecimal`, loses all "padding" zeros). (Bug#6537 [http://bugs.mysql.com/6537])
- Use `DatabaseMetaData.getIdentifierQuoteString()` when building DBMD queries.
- Use 1MB packet for sending file for `LOAD DATA LOCAL INFILE` if that is < `max_allowed_packet` on server.
- `ResultSetMetaData.getColumnDisplaySize()` returns incorrect values for multi-byte charsets. (Bug#6399 [http://bugs.mysql.com/6399])
- Make auto-deserialization of `java.lang.Objects` stored in `BLOB` columns configurable via `autoDeserialize` property (defaults to `false`).
- Re-work `Field.isOpaqueBinary()` to detect `CHAR(`*n*`) CHARACTER SET BINARY` to support fixed-length binary fields for `ResultSet.getObject()`.
- Use our own implementation of buffered input streams to get around blocking behavior of `java.io.BufferedInputStream`. Disable this with `useReadAheadInput=false`.
- Failing to connect to the server when one of the addresses for the given host name is IPV6 (which the server does not yet bind on). The driver now loops through *all* IP addresses for a given host, and stops on the first one that `accepts()` a `socket.connect()`. (Bug#6348 [http://bugs.mysql.com/6348])

1.6.11. Changes in MySQL Connector/J 3.1.4-beta (04 September 2004)

- Connector/J 3.1.3 beta does not handle integers correctly (caused by changes to support unsigned reads in `Buffer.readInt()` -> `Buffer.readShort()`). (Bug#4510 [http://bugs.mysql.com/4510])
- Added support in `DatabaseMetaData.getTables()` and `getTableTypes()` for views, which are now available in MySQL server 5.0.x.
- `ServerPreparedStatement.execute*()` sometimes threw `ArrayIndexOutOfBoundsException` when unpacking field metadata. (Bug#4642 [http://bugs.mysql.com/4642])
- Optimized integer number parsing, enable “old” slower integer parsing using JDK classes via `useFastIntParsing=false` property.
- Added `useOnlyServerErrorMessages` property, which causes message text in exceptions generated by the server to only contain the text sent by the server (as opposed to the SQLState's “standard” description, followed by the server's error message). This property is set to `true` by default.
- `ResultSet.wasNull()` does not work for primatives if a previous `null` was returned. (Bug#4689 [http://bugs.mysql.com/4689])
- Track packet sequence numbers if `enablePacketDebug=true`, and throw an exception if packets received out-of-order.
- `ResultSet.getObject()` returns wrong type for strings when using prepared statements. (Bug#4482 [http://bugs.mysql.com/4482])
- Calling `MysqlPooledConnection.close()` twice (even though an application error), caused NPE. Fixed.
- `ServerPreparedStatements` dealing with return of `DECIMAL` type don't work. (Bug#5012 [http://bugs.mysql.com/5012])
- `ResultSet.getObject()` doesn't return type `Boolean` for pseudo-bit types from prepared statements on 4.1.x (shortcut for avoiding extra type conversion when using binary-encoded result sets obscured test in `getObject()` for “pseudo” bit type). (Bug#5032 [http://bugs.mysql.com/5032])
- You can now use URLs in `LOAD DATA LOCAL INFILE` statements, and the driver will use Java's built-in handlers for retreiving the data and sending it to the server. This feature is not enabled by default, you must set the `allowUrlInLocalInfile` connection property to `true`.
- The driver is more strict about truncation of numerics on `ResultSet.get*()`, and will throw an `SQLException` when truncation is detected. You can disable this by setting `jdbcCompliantTruncation` to `false` (it is enabled by default, as this functionality is required for JDBC compliance).
- Added three ways to deal with all-zero datetimes when reading them from a `ResultSet`: `exception` (the default), which throws an `SQLException` with an SQLState of `S1009`; `convertToNull`, which returns `NULL` instead of the date; and `round`, which rounds the date to the nearest closest value which is `'0001-01-01'`.
- Fixed `ServerPreparedStatement` to read prepared statement metadata off the wire, even though it's currently a placeholder instead of using `MysqlIO.clearInputStream()` which

didn't work at various times because data wasn't available to read from the server yet. This fixes sporadic errors users were having with `ServerPreparedStatements` throwing `ArrayIndexOutOfBoundExceptions`.

- Use `com.mysql.jdbc.Message`'s classloader when loading resource bundle, should fix sporadic issues when the caller's classloader can't locate the resource bundle.

1.6.12. Changes in MySQL Connector/J 3.1.3-beta (07 July 2004)

- Mangle output parameter names for `CallableStatements` so they will not clash with user variable names.
- Added support for `INOUT` parameters in `CallableStatements`.
- Null bitmask sent for server-side prepared statements was incorrect. (Bug#4119 [http://bugs.mysql.com/4119])
- Use SQL Standard SQL states by default, unless `useSqlStateCodes` property is set to `false`.
- Added packet debuging code (see the `enablePacketDebug` property documentation).
- Added constants for MySQL error numbers (publicly accessible, see `com.mysql.jdbc.MysqlErrorNumbers`), and the ability to generate the mappings of vendor error codes to SQLStates that the driver uses (for documentation purposes).
- Externalized more messages (on-going effort).
- Error in retrieval of `mediumint` column with prepared statements and binary protocol. (Bug#4311 [http://bugs.mysql.com/4311])
- Support new time zone variables in MySQL-4.1.3 when `useTimezone=true`.
- Support for unsigned numerics as return types from prepared statements. This also causes a change in `ResultSet.getObject()` for the `bigint unsigned` type, which used to return `BigDecimal` instances, it now returns instances of `java.lang.BigInteger`.

1.6.13. Changes in MySQL Connector/J 3.1.2-alpha (09 June 2004)

- Fixed stored procedure parameter parsing info when size was specified for a parameter (for example, `char()`, `varchar()`).
- Enabled callable statement caching via `cacheCallableStmts` property.
- Fixed case when no output parameters specified for a stored procedure caused a bogus query to be issued to retrieve out parameters, leading to a syntax error from the server.
- Fixed case when no parameters could cause a `NullPointerException` in `CallableStatement.setOutputParameters()`.
- Removed wrapping of exceptions in `MysqlIO.changeUser()`.
- Fixed sending of split packets for large queries, enabled nio ability to send large packets as well.
- Added `.toString()` functionality to `ServerPreparedStatement`, which should help if you're trying to debug a query that is a prepared statement (it shows SQL as the server would pro-

cess).

- Added `gatherPerformanceMetrics` property, along with properties to control when/where this info gets logged (see docs for more info).
- `ServerPreparedStatements` weren't actually de-allocating server-side resources when `.close()` was called.
- Added `logSlowQueries` property, along with `slowQueriesThresholdMillis` property to control when a query should be considered "slow."
- Correctly map output parameters to position given in `prepareCall()` versus. order implied during `registerOutParameter()`. (Bug#3146 [http://bugs.mysql.com/3146])
- Correctly detect initial character set for servers >= 4.1.0.
- Cleaned up detection of server properties.
- Support placeholder for parameter metadata for server >= 4.1.2.
- `getProcedures()` does not return any procedures in result set. (Bug#3539 [http://bugs.mysql.com/3539])
- `getProcedureColumns()` doesn't work with wildcards for procedure name. (Bug#3540 [http://bugs.mysql.com/3540])
- `DBMD.getSQLStateType()` returns incorrect value. (Bug#3520 [http://bugs.mysql.com/3520])
- Added `connectionCollation` property to cause driver to issue `set collation_connection=...` query on connection init if default collation for given charset is not appropriate.
- Fixed `DatabaseMetaData.getProcedures()` when run on MySQL-5.0.0 (output of `SHOW PROCEDURE STATUS` changed between 5.0.0 and 5.0.1.
- `getWarnings()` returns `SQLWarning` instead of `DataTruncation`. (Bug#3804 [http://bugs.mysql.com/3804])
- Don't enable server-side prepared statements for server version 5.0.0 or 5.0.1, as they aren't compatible with the '4.1.2+' style that the driver uses (the driver expects information to come back that isn't there, so it hangs).

1.6.14. Changes in MySQL Connector/J 3.1.1-alpha (14 February 2004)

- Fixed bug with `UpdatableResultSets` not using client-side prepared statements.
- Fixed character encoding issues when converting bytes to ASCII when MySQL doesn't provide the character set, and the JVM is set to a multi-byte encoding (usually affecting retrieval of numeric values).
- Unpack "unknown" data types from server prepared statements as `Strings`.
- Implemented long data (Blobs, Clobs, InputStreams, Readers) for server prepared statements.
- Implemented `Statement.getWarnings()` for MySQL-4.1 and newer (using `SHOW WARN-`

INGS).

- Default result set type changed to TYPE_FORWARD_ONLY (JDBC compliance).
- Centralized setting of result set type and concurrency.
- Refactored how connection properties are set and exposed as DriverPropertyInfo as well as Connection and DataSource properties.
- Support for NIO. Use useNIO=true on platforms that support NIO.
- Support for transaction savepoints (MySQL >= 4.0.14 or 4.1.1).
- Support for mysql_change_user(). See the changeUser() method in com.mysql.jdbc.Connection.
- Reduced number of methods called in average query to be more efficient.
- Prepared Statements will be re-prepared on auto-reconnect. Any errors encountered are postponed until first attempt to re-execute the re-prepared statement.
- Ensure that warnings are cleared before executing queries on prepared statements, as-per JDBC spec (now that we support warnings).
- Support "old" profileSql capitalization in ConnectionProperties. This property is deprecated, you should use profileSQL if possible.
- Optimized Buffer.readLenByteArray() to return shared empty byte array when length is 0.
- Allow contents of PreparedStatement.setBlob() to be retained between calls to .execute*().
- Deal with 0-length tokens in EscapeProcessor (caused by callable statement escape syntax).
- Check for closed connection on delete/update/insert row operations in UpdatableResultSet.
- Fix support for table aliases when checking for all primary keys in UpdatableResultSet.
- Removed useFastDates connection property.
- Correctly initialize datasource properties from JNDI Refs, including explicitly specified URLs.
- DatabaseMetaData now reports supportsStoredProcedures() for MySQL versions >= 5.0.0
- Fixed stack overflow in Connection.prepareCall() (bad merge).
- Fixed IllegalAccessError to Calendar.getTimeInMillis() in DateTimeValue (for JDK < 1.4).
- DatabaseMetaData.getColumns() is not returning correct column ordinal info for non-'%' column name patterns. (Bug#1673 [http://bugs.mysql.com/1673])
- Merged fix of datatype mapping from MySQL type FLOAT to java.sql.Types.REAL from 3.0 branch.
- Detect collation of column for RSMD.isCaseSensitive().
- Fixed sending of queries larger than 16M.

- Added named and indexed input/output parameter support to `CallableStatement`. MySQL-5.0.x or newer.
- Fixed `NullPointerException` in `ServerPreparedStatement.setTimestamp()`, as well as year and month descrepencies in `ServerPreparedStatement.setTimestamp()`, `setDate()`.
- Added ability to have multiple database/JVM targets for compliance and regression/unit tests in `build.xml`.
- Fixed NPE and year/month bad conversions when accessing some datetime functionality in `ServerPreparedStatements` and their resultant result sets.
- Display where/why a connection was implicitly closed (to aid debugging).
- `CommunicationsException` implemented, that tries to determine why communications was lost with a server, and displays possible reasons when `.getMessage()` is called.
- `NULL` values for numeric types in binary encoded result sets causing `NullPointerExceptions`. (Bug#2359 [http://bugs.mysql.com/2359])
- Implemented `Connection.prepareCall()`, and `DatabaseMetaData.getProcedures()` and `getProcedureColumns()`.
- Reset `long binary` parameters in `ServerPreparedStatement` when `clearParameters()` is called, by sending `COM_RESET_STMT` to the server.
- Merged prepared statement caching, and `.getMetaData()` support from 3.0 branch.
- Fixed off-by-1900 error in some cases for years in `TimeUtil.fastDate`/`TimeCreate()` when unpacking results from server-side prepared statements.
- Fixed charset conversion issue in `getTables()`. (Bug#2502 [http://bugs.mysql.com/2502])
- Implemented multiple result sets returned from a statement or stored procedure.
- Server-side prepared statements were not returning datatype `YEAR` correctly. (Bug#2606 [http://bugs.mysql.com/2606])
- Enabled streaming of result sets from server-side prepared statements.
- Class-cast exception when using scrolling result sets and server-side prepared statements. (Bug#2623 [http://bugs.mysql.com/2623])
- Merged unbuffered input code from 3.0.
- Fixed `ConnectionProperties` that weren't properly exposed via accessors, cleaned up `ConnectionProperties` code.
- `NULL` fields were not being encoded correctly in all cases in server-side prepared statements. (Bug#2671 [http://bugs.mysql.com/2671])
- Fixed rare buffer underflow when writing numbers into buffers for sending prepared statement execution requests.
- Use DocBook version of docs for shipped versions of drivers.

1.6.15. Changes in MySQL Connector/J 3.1.0-alpha (18 February

2003)

- Added `requireSSL` property.
- Added `useServerPrepStmts` property (default `false`). The driver will use server-side prepared statements when the server version supports them (4.1 and newer) when this property is set to `true`. It is currently set to `false` by default until all bind/fetch functionality has been implemented. Currently only DML prepared statements are implemented for 4.1 server-side prepared statements.
- Track open `Statements`, close all when `Connection.close()` is called (JDBC compliance).

1.6.16. Changes in MySQL Connector/J 3.0.17-ga (23 June 2005)

- `Timestamp`/`Time` conversion goes in the wrong "direction" when `useTimeZone=true` and server time zone differs from client time zone. (Bug#5874 [http://bugs.mysql.com/5874])
- `DatabaseMetaData.getIndexInfo()` ignored `unique` parameter. (Bug#7081 [http://bugs.mysql.com/7081])
- Support new protocol type `MYSQL_TYPE_VARCHAR`.
- Added `useOldUTF8Behavior`' configuration property, which causes JDBC driver to act like it did with MySQL-4.0.x and earlier when the character encoding is `utf-8` when connected to MySQL-4.1 or newer.
- Statements created from a pooled connection were returning physical connection instead of logical connection when `getConnection()` was called. (Bug#7316 [http://bugs.mysql.com/7316])
- `PreparedStatements` don't encode Big5 (and other multi-byte) character sets correctly in static SQL strings. (Bug#7033 [http://bugs.mysql.com/7033])
- Connections starting up failed-over (due to down master) never retry master. (Bug#6966 [http://bugs.mysql.com/6966])
- `PreparedStatement.fixDecimalExponent()` adding extra `+`, making number unparseable by MySQL server. (Bug#7061 [http://bugs.mysql.com/7061])
- Timestamp key column data needed `_binary` stripped for `UpdatableResultSet.refreshRow()`. (Bug#7686 [http://bugs.mysql.com/7686])
- Backported SQLState codes mapping from Connector/J 3.1, enable with `useSqlStateCodes=true` as a connection property, it defaults to `false` in this release, so that we don't break legacy applications (it defaults to `true` starting with Connector/J 3.1).
- `PreparedStatement.fixDecimalExponent()` adding extra `+`, making number unparseable by MySQL server. (Bug#7601 [http://bugs.mysql.com/7601])
- Escape sequence {fn convert(..., type)} now supports ODBC-style types that are prepended by `SQL_`.
- Fixed duplicated code in `configureClientCharset()` that prevented `useOldUTF8Behavior=true` from working properly.
- Handle streaming result sets with more than 2 billion rows properly by fixing wraparound of row

number counter.

- `MS932`, `SHIFT_JIS`, and `Windows_31J` not recognized as aliases for `sjis`. (Bug#7607 [http://bugs.mysql.com/7607])
- Adding `CP943` to aliases for `sjis`. (Bug#6549 [http://bugs.mysql.com/6549], fixed while fixing Bug#7607 [http://bugs.mysql.com/7607])
- Which requires hex escaping of binary data when using multi-byte charsets with prepared statements. (Bug#8064 [http://bugs.mysql.com/8064])
- `NON_UNIQUE` column from `DBMD.getIndexInfo()` returned inverted value. (Bug#8812 [http://bugs.mysql.com/8812])
- Workaround for server Bug#9098 [http://bugs.mysql.com/9098]: Default values of `CURRENT_*` for `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` columns can't be distinguished from `string` values, so `UpdatableResultSet.moveToInsertRow()` generates bad SQL for inserting default values.
- `EUCKR` charset is sent as `SET NAMES euc_kr` which MySQL-4.1 and newer doesn't understand. (Bug#8629 [http://bugs.mysql.com/8629])
- `DatabaseMetaData.supportsSelectForUpdate()` returns correct value based on server version.
- Use hex escapes for `PreparedStatement.setBytes()` for double-byte charsets including "aliases" `Windows-31J`, `CP934`, `MS932`.
- Added support for the `EUC_JP_Solaris` character encoding, which maps to a MySQL encoding of `eucjpms` (backported from 3.1 branch). This only works on servers that support `eucjpms`, namely 5.0.3 or later.

1.6.17. Changes in MySQL Connector/J 3.0.16-ga (15 November 2004)

- Re-issue character set configuration commands when re-using pooled connections and/or `Connection.changeUser()` when connected to MySQL-4.1 or newer.
- Fixed `ResultSetMetaData.isReadOnly()` to detect non-writable columns when connected to MySQL-4.1 or newer, based on existence of "original" table and column names.
- `ResultSet.updateByte()` when on insert row throws `ArrayOutOfBoundsException`. (Bug#5664 [http://bugs.mysql.com/5664])
- Fixed `DatabaseMetaData.getTypes()` returning incorrect (this is, non-negative) scale for the `NUMERIC` type.
- Off-by-one bug in `Buffer.readString(`*`string`*`)`. (Bug#5664 [http://bugs.mysql.com/5664])
- Made `TINYINT(1)` -> `BIT`/`Boolean` conversion configurable via `tinyInt1isBit` property (default `true` to be JDBC compliant out of the box).
- Only set `character_set_results` during connection establishment if server version >= 4.1.1.
- Fixed regression where `useUnbufferedInput` was defaulting to `false`.

- `ResultSet.getTimestamp()` on a column with `TIME` in it fails. (Bug#5664 [http://bugs.mysql.com/5664])

1.6.18. Changes in MySQL Connector/J 3.0.15-production (04 September 2004)

- `StringUtils.escapeEasternUnicodeByteStream` was still broken for GBK. (Bug#4010 [http://bugs.mysql.com/4010])

- Failover for `autoReconnect` not using port numbers for any hosts, and not retrying all hosts. (**Warning**: This required a change to the `SocketFactory` `connect()` method signature, which is now `public Socket connect(String host, int portNumber, Properties props)`; therefore, any third-party socket factories will have to be changed to support this signature. (Bug#4334 [http://bugs.mysql.com/4334])

- Logical connections created by `MysqlConnectionPoolDataSource` will now issue a `rollback()` when they are closed and sent back to the pool. If your application server/connection pool already does this for you, you can set the `rollbackOnPooledClose` property to `false` to avoid the overhead of an extra `rollback()`.

- Removed redundant calls to `checkRowPos()` in `ResultSet`.

- `DOUBLE` mapped twice in `DBMD.getTypeInfo()`. (Bug#4742 [http://bugs.mysql.com/4742])

- Added FLOSS license exemption.

- Calling `.close()` twice on a `PooledConnection` causes NPE. (Bug#4808 [http://bugs.mysql.com/4808])

- `DBMD.getColumns()` returns incorrect JDBC type for unsigned columns. This affects type mappings for all numeric types in the `RSMD.getColumnType()` and `RSMD.getColumnTypeNames()` methods as well, to ensure that "like" types from `DBMD.getColumns()` match up with what `RSMD.getColumnType()` and `getColumnTypeNames()` return. (Bug#4138 [http://bugs.mysql.com/4138], Bug#4860 [http://bugs.mysql.com/4860])

- "Production" is now "GA" (General Availability) in naming scheme of distributions.

- `RSMD.getPrecision()` returning 0 for non-numeric types (should return max length in chars for non-binary types, max length in bytes for binary types). This fix also fixes mapping of `RSMD.getColumnType()` and `RSMD.getColumnTypeName()` for the `BLOB` types based on the length sent from the server (the server doesn't distinguish between `TINYBLOB`, `BLOB`, `MEDIUMBLOB` or `LONGBLOB` at the network protocol level). (Bug#4880 [http://bugs.mysql.com/4880])

- `ResultSet` should release `Field[]` instance in `.close()`. (Bug#5022 [http://bugs.mysql.com/5022])

- `ResultSet.getMetaData()` should not return incorrectly initialized metadata if the result set has been closed, but should instead throw an `SQLException`. Also fixed for `getRow()` and `getWarnings()` and traversal methods by calling `checkClosed()` before operating on instance-level fields that are nullified during `.close()`. (Bug#5069 [http://bugs.mysql.com/5069])

- Parse new time zone variables from 4.1.x servers.

- Use `_binary` introducer for `PreparedStatement.setBytes()` and `set*Stream()` when connected to MySQL-4.1.x or newer to avoid misinterpretation during character conversion.

1.6.19. Changes in MySQL Connector/J 3.0.14-production (28 May 2004)

- Fixed URL parsing error.

1.6.20. Changes in MySQL Connector/J 3.0.13-production (27 May 2004)

- Using a `MySQLDatasource` without server name fails. (Bug#3848 [http://bugs.mysql.com/3848])
- `No Database Selected` when using `MysqlConnectionPoolDataSource`. (Bug#3920 [http://bugs.mysql.com/3920])
- `PreparedStatement.getGeneratedKeys()` method returns only 1 result for batched insertions. (Bug#3873 [http://bugs.mysql.com/3873])

1.6.21. Changes in MySQL Connector/J 3.0.12-production (18 May 2004)

- Add unsigned attribute to `DatabaseMetaData.getColumns()` output in the `TYPE_NAME` column.
- Added `failOverReadOnly` property, to allow end-user to configure state of connection (read-only/writable) when failed over.
- Backported "change user" and "reset server state" functionality from 3.1 branch, to allow clients of `MysqlConnectionPoolDataSource` to reset server state on `getConnection()` on a pooled connection.
- Don't escape SJIS/GBK/BIG5 when using MySQL-4.1 or newer.
- Allow `url` parameter for `MysqlDataSource` and `MysqlConnectionPool DataSource` so that passing of other properties is possible from inside appservers.
- Map duplicate key and foreign key errors to SQLState of `23000`.
- Backport documentation tooling from 3.1 branch.
- Return creating statement for `ResultSets` created by `getGeneratedKeys()`. (Bug#2957 [http://bugs.mysql.com/2957])
- Allow `java.util.Date` to be sent in as parameter to `PreparedStatement.setObject()`, converting it to a `Timestamp` to maintain full precision. (Bug#103 [http://bugs.mysql.com/103]).
- Don't truncate `BLOB` or `CLOB` values when using `setBytes()` and/or `setBinary/CharacterStream()`. (Bug#2670 [http://bugs.mysql.com/2670]).
- Dynamically configure character set mappings for field-level character sets on MySQL-4.1.0 and newer using `SHOW COLLATION` when connecting.
- Map `binary` character set to `US-ASCII` to support `DATETIME` charset recognition for servers >=

4.1.2.

- Use `SET character_set_results` during initialization to allow any charset to be returned to the driver for result sets.
- Use `charsetnr` returned during connect to encode queries before issuing `SET NAMES` on MySQL >= 4.1.0.
- Add helper methods to `ResultSetMetaData` (`getColumnCharacterEncoding()` and `getColumnCharacterSet()`) to allow end-users to see what charset the driver thinks it should be using for the column.
- Only set `character_set_results` for MySQL >= 4.1.0.
- `StringUtils.escapeSJISByteStream()` not covering all eastern double-byte charsets correctly. (Bug#3511 [http://bugs.mysql.com/3511])
- Renamed `StringUtils.escapeSJISByteStream()` to more appropriate `escapeEasternUnicodeByteStream()`.
- Not specifying database in URL caused `MalformedURL` exception. (Bug#3554 [http://bugs.mysql.com/3554])
- Auto-convert MySQL encoding names to Java encoding names if used for `characterEncoding` property.
- Added encoding names that are recognized on some JVMs to fix case where they were reverse-mapped to MySQL encoding names incorrectly.
- Use `junit.textui.TestRunner` for all unit tests (to allow them to be run from the command line outside of Ant or Eclipse).
- `UpdatableResultSet` not picking up default values for `moveToInsertRow()`. (Bug#3557 [http://bugs.mysql.com/3557])
- Inconsistent reporting of data type. The server still doesn't return all types for *BLOBs *TEXT correctly, so the driver won't return those correctly. (Bug#3570 [http://bugs.mysql.com/3570])
- `DBMD.getSQLStateType()` returns incorrect value. (Bug#3520 [http://bugs.mysql.com/3520])
- Fixed regression in `PreparedStatement.setString()` and eastern character encodings.
- Made `StringRegressionTest` 4.1-unicode aware.

1.6.22. Changes in MySQL Connector/J 3.0.11-stable (19 February 2004)

- Trigger a `SET NAMES utf8` when encoding is forced to `utf8` *or* `utf-8` via the `characterEncoding` property. Previously, only the Java-style encoding name of `utf-8` would trigger this.
- `AutoReconnect` time was growing faster than exponentially. (Bug#2447 [http://bugs.mysql.com/2447])
- Fixed failover always going to last host in list. (Bug#2578 [http://bugs.mysql.com/2578])
- Added `useUnbufferedInput` parameter, and now use it by default (due to JVM issue ht-

tp://developer.java.sun.com/developer/bugParade/bugs/4401235.html)

- Detect `on`/`off` or `1`, `2`, `3` form of `lower_case_table_names` value on server.
- Return `java.lang.Integer` for `TINYINT` and `SMALLINT` types from `ResultSetMetaData.getColumnClassName()`. (Bug#2852 [http://bugs.mysql.com/2852])
- Return `java.lang.Double` for `FLOAT` type from `ResultSetMetaData.getColumnClassName()`. (Bug#2855 [http://bugs.mysql.com/2855])
- Return `[B` instead of `java.lang.Object` for `BINARY`, `VARBINARY` and `LONGVARBINARY` types from `ResultSetMetaData.getColumnClassName()` (JDBC compliance).
- Issue connection events on all instances created from a `ConnectionPoolDataSource`.

1.6.23. Changes in MySQL Connector/J 3.0.10-stable (13 January 2004)

- Don't count quoted IDs when inside a 'string' in `PreparedStatement` parsing. (Bug#1511 [http://bugs.mysql.com/1511])
- “Friendlier” exception message for `PacketTooLargeException`. (Bug#1534 [http://bugs.mysql.com/1534])
- Backported fix for aliased tables and `UpdatableResultSets` in `checkUpdatability()` method from 3.1 branch.
- Fix for `ArrayIndexOutOfBounds` exception when using `Statement.setMaxRows()`. (Bug#1695 [http://bugs.mysql.com/1695])
- Barge blobs and split packets not being read correctly. (Bug#1576 [http://bugs.mysql.com/1576])
- Fixed regression of `Statement.getGeneratedKeys()` and `REPLACE` statements.
- Subsequent call to `ResultSet.updateFoo()` causes NPE if result set is not updatable. (Bug#1630 [http://bugs.mysql.com/1630])
- Fix for 4.1.1-style authentication with no password.
- Foreign Keys column sequence is not consistent in `DatabaseMetaData.getImported/Exported/CrossReference()`. (Bug#1731 [http://bugs.mysql.com/1731])
- `DatabaseMetaData.getSystemFunction()` returning bad function `VResultsSion`. (Bug#1775 [http://bugs.mysql.com/1775])
- Cross-database updatable result sets are not checked for updatability correctly. (Bug#1592 [http://bugs.mysql.com/1592])
- `DatabaseMetaData.getColumns()` should return `Types.LONGVARCHAR` for MySQL `LONGTEXT` type.
- `ResultSet.getObject()` on `TINYINT` and `SMALLINT` columns should return Java type `Integer`. (Bug#1913 [http://bugs.mysql.com/1913])
- Added `alwaysClearStream` connection property, which causes the driver to always empty any

remaining data on the input stream before each query.

- Added more descriptive error message `Server Configuration Denies Access to DataSource`, as well as retrieval of message from server.
- Autoreconnect code didn't set catalog upon reconnect if it had been changed.
- Implement `ResultSet.updateClob()`.
- `ResultSetMetaData.isCaseSensitive()` returned wrong value for `CHAR`/`VARCHAR` columns.
- Connection property `maxRows` not honored. (Bug#1933 [http://bugs.mysql.com/1933])
- Statements being created too many times in `DBMD.extractForeignKeyFromCreateTable()`. (Bug#1925 [http://bugs.mysql.com/1925])
- Support escape sequence {fn convert ... }. (Bug#1914 [http://bugs.mysql.com/1914])
- `ArrayIndexOutOfBounds` when parameter number == number of parameters + 1. (Bug#1958 [http://bugs.mysql.com/1958])
- `ResultSet.findColumn()` should use first matching column name when there are duplicate column names in `SELECT` query (JDBC-compliance). (Bug#2006 [http://bugs.mysql.com/2006])
- Removed static synchronization bottleneck from `PreparedStatement.setTimestamp()`.
- Removed static synchronization bottleneck from instance factory method of `SingleByteCharsetConverter`.
- Enable caching of the parsing stage of prepared statements via the `cachePrepStmts`, `prepStmtCacheSize`, and `prepStmtCacheSqlLimit` properties (disabled by default).
- Speed up parsing of `PreparedStatements`, try to use one-pass whenever possible.
- Fixed security exception when used in Applets (applets can't read the system property `file.encoding` which is needed for `LOAD DATA LOCAL INFILE`).
- Use constants for SQLStates.
- Map charset `ko18_ru` to `ko18r` when connected to MySQL-4.1.0 or newer.
- Ensure that `Buffer.writeString()` saves room for the `\0`.
- Fixed exception `Unknown character set 'danish'` on connect with JDK-1.4.0
- Fixed mappings in SQLError to report deadlocks with SQLStates of `41000`.
- `maxRows` property would affect internal statements, so check it for all statement creation internal to the driver, and set to 0 when it is not.

1.6.24. Changes in MySQL Connector/J 3.0.9-stable (07 October 2003)

- Faster date handling code in `ResultSet` and `PreparedStatement` (no longer uses `Date` methods that synchronize on static calendars).

- Fixed test for end of buffer in `Buffer.readString()`.
- Fixed `ResultSet.previous()` behavior to move current position to before result set when on first row of result set. (Bug#496 [http://bugs.mysql.com/496])
- Fixed `Statement` and `PreparedStatement` issuing bogus queries when `setMaxRows()` had been used and a `LIMIT` clause was present in the query.
- `refreshRow` didn't work when primary key values contained values that needed to be escaped (they ended up being doubly escaped). (Bug#661 [http://bugs.mysql.com/661])
- Support `InnoDB` contraint names when extracting foreign key information in `DatabaseMetaData` (implementing ideas from Parwinder Sekhon). (Bug#517 [http://bugs.mysql.com/517], Bug#664 [http://bugs.mysql.com/664])
- Backported 4.1 protocol changes from 3.1 branch (server-side SQL states, new field information, larger client capability flags, connect-with-database, and so forth).
- Fix `UpdatableResultSet` to return values for `get`*`XXX`*`()` when on insert row. (Bug#675 [http://bugs.mysql.com/675])
- The `insertRow` in an `UpdatableResultSet` is now loaded with the default column values when `moveToInsertRow()` is called. (Bug#688 [http://bugs.mysql.com/688])
- `DatabaseMetaData.getColumns()` wasn't returning `NULL` for default values that are specified as `NULL`.
- Change default statement type/concurrency to `TYPE_FORWARD_ONLY` and `CONCUR_READ_ONLY` (spec compliance).
- Don't try and reset isolation level on reconnect if MySQL doesn't support them.
- Don't wrap `SQLExceptions` in `RowDataDynamic`.
- Don't change timestamp TZ twice if `useTimezone==true`. (Bug#774 [http://bugs.mysql.com/774])
- Fixed regression in large split-packet handling. (Bug#848 [http://bugs.mysql.com/848])
- Better diagnostic error messages in exceptions for “streaming” result sets.
- Issue exception on `ResultSet.get`*`XXX`*`()` on empty result set (wasn't caught in some cases).
- Don't hide messages from exceptions thrown in I/O layers.
- Don't fire connection closed events when closing pooled connections, or on `PooledConnection.getConnection()` with already open connections. (Bug#884 [http://bugs.mysql.com/884])
- Clip +/- INF (to smallest and largest representative values for the type in MySQL) and NaN (to 0) for `setDouble`/`setFloat()`, and issue a warning on the statement when the server does not support +/- INF or NaN.
- Double-escaping of `'\'` when charset is SJIS or GBK and `'\'` appears in non-escaped input. (Bug#879 [http://bugs.mysql.com/879])
- When emptying input stream of unused rows for “streaming” result sets, have the current thread `yield()` every 100 rows in order to not monopolize CPU time.

- `DatabaseMetaData.getColumns()` getting confused about the keyword "set" in character columns. (Bug#1099 [http://bugs.mysql.com/1099])
- Fixed deadlock issue with `Statement.setMaxRows()`.
- Fixed `CLOB.truncate()`. (Bug#1130 [http://bugs.mysql.com/1130])
- Optimized `CLOB.setChracterStream()`. (Bug#1131 [http://bugs.mysql.com/1131])
- Made `databaseName`, `portNumber`, and `serverName` optional parameters for `MysqlDataSourceFactory`. (Bug#1246 [http://bugs.mysql.com/1246])
- `ResultSet.get/setString` mashing char 127. (Bug#1247 [http://bugs.mysql.com/1247])
- Backported authentication changes for 4.1.1 and newer from 3.1 branch.
- Added `com.mysql.jdbc.util.BaseBugReport` to help creation of testcases for bug reports.
- Added property to "clobber" streaming results, by setting the `clobberStreamingResults` property to `true` (the default is `false`). This will cause a "streaming" `ResultSet` to be automatically closed, and any oustanding data still streaming from the server to be discarded if another query is executed before all the data has been read from the server.

1.6.25. Changes in MySQL Connector/J 3.0.8-stable (23 May 2003)

- Allow bogus URLs in `Driver.getPropertyInfo()`.
- Return list of generated keys when using multi-value `INSERTS` with `Statement.getGeneratedKeys()`.
- Use JVM charset with filenames and `LOAD DATA [LOCAL] INFILE`.
- Fix infinite loop with `Connection.cleanup()`.
- Changed Ant target `compile-core` to `compile-driver`, and made testsuite compilation a separate target.
- Fixed result set not getting set for `Statement.executeUpdate()`, which affected `getGeneratedKeys()` and `getUpdateCount()` in some cases.
- Unicode character 0xFFFF in a string would cause the driver to throw an `ArrayOutOfBoundsException`. (Bug#378 [http://bugs.mysql.com/378]).
- Return correct number of generated keys when using `REPLACE` statements.
- Fix problem detecting server character set in some cases.
- Fix row data decoding error when using *very* large packets.
- Optimized row data decoding.
- Issue exception when operating on an already closed prepared statement.
- Fixed SJIS encoding bug, thanks to Naoto Sato.
- Optimized usage of `EscapeProcessor`.

- Allow multiple calls to `Statement.close()`.

1.6.26. Changes in MySQL Connector/J 3.0.7-stable (08 April 2003)

- Fixed `MysqlPooledConnection.close()` calling wrong event type.
- Fixed `StringIndexOutOfBoundsException` in `PreparedStatement.setClob()`.
- 4.1 Column Metadata fixes.
- Remove synchronization from `Driver.connect()` and `Driver.acceptsUrl()`.
- `IOExceptions` during a transaction now cause the `Connection` to be closed.
- Fixed missing conversion for `YEAR` type in `ResultSetMetaData.getColumnTypeName()`.
- Don't pick up indexes that start with `pri` as primary keys for `DBMD.getPrimaryKeys()`.
- Throw `SQLExceptions` when trying to do operations on a forcefully closed `Connection` (that is, when a communication link failure occurs).
- You can now toggle profiling on/off using `Connection.setProfileSql(boolean)`.
- Fixed charset issues with database metadata (charset was not getting set correctly).
- Updatable `ResultSets` can now be created for aliased tables/columns when connected to MySQL-4.1 or newer.
- Fixed `LOAD DATA LOCAL INFILE` bug when file > `max_allowed_packet`.
- Fixed escaping of 0x5c (`'\'`) character for GBK and Big5 charsets.
- Fixed `ResultSet.getTimestamp()` when underlying field is of type `DATE`.
- Ensure that packet size from `alignPacketSize()` does not exceed `max_allowed_packet` (JVM bug)
- Don't reset `Connection.isReadOnly()` when autoReconnecting.

1.6.27. Changes in MySQL Connector/J 3.0.6-stable (18 February 2003)

- Fixed `ResultSetMetaData` to return `""` when catalog not known. Fixes `NullPointerExceptions` with Sun's `CachedRowSet`.
- Fixed `DBMD.getTypeInfo()` and `DBMD.getColumns()` returning different value for precision in `TEXT` and `BLOB` types.
- Allow ignoring of warning for “non transactional tables” during rollback (compliance/usability) by setting `ignoreNonTxTables` property to `true`.
- Fixed `SQLExceptions` getting swallowed on initial connect.
- Fixed `Statement.setMaxRows()` to stop sending `LIMIT` type queries when not needed

(performance).

- Clean up `Statement` query/method mismatch tests (that is, `INSERT` not allowed with `.executeQuery()`).
- More checks added in `ResultSet` traversal method to catch when in closed state.
- Fixed `ResultSetMetaData.isWritable()` to return correct value.
- Add "window" of different `NULL` sorting behavior to `DBMD.nullsAreSortedAtStart` (4.0.2 to 4.0.10, true; otherwise, no).
- Implemented `Blob.setBytes()`. You still need to pass the resultant `Blob` back into an updatable `ResultSet` or `PreparedStatement` to persist the changes, because MySQL does not support "locators".
- Backported 4.1 charset field info changes from Connector/J 3.1.

1.6.28. Changes in MySQL Connector/J 3.0.5-gamma (22 January 2003)

- Fixed `Buffer.fastSkipLenString()` causing `ArrayIndexOutOfBounds` exceptions with some queries when unpacking fields.
- Implemented an empty `TypeMap` for `Connection.getTypeMap()` so that some third-party apps work with MySQL (IBM WebSphere 5.0 Connection pool).
- Added missing `LONGTEXT` type to `DBMD.getColumns()`.
- Retrieve `TX_ISOLATION` from database for `Connection.getTransactionIsolation()` when the MySQL version supports it, instead of an instance variable.
- Quote table names in `DatabaseMetaData.getColumns()`, `getPrimaryKeys()`, `getIndexInfo()`, `getBestRowIdentifier()`.
- Greatly reduce memory required for `setBinaryStream()` in `PreparedStatements`.
- Fixed `ResultSet.isBeforeFirst()` for empty result sets.
- Added update options for foreign key metadata.

1.6.29. Changes in MySQL Connector/J 3.0.4-gamma (06 January 2003)

- Added quoted identifiers to database names for `Connection.setCatalog`.
- Added support for quoted identifiers in `PreparedStatement` parser.
- Streamlined character conversion and `byte[]` handling in `PreparedStatements` for `setByte()`.
- Reduce memory footprint of `PreparedStatements` by sharing outbound packet with `MysqlIO`.

- Added `strictUpdates` property to allow control of amount of checking for "correctness" of updatable result sets. Set this to `false` if you want faster updatable result sets and you know that you create them from `SELECT` statements on tables with primary keys and that you have selected all primary keys in your query.
- Added support for 4.0.8-style large packets.
- Fixed `PreparedStatement.executeBatch()` parameter overwriting.

1.6.30. Changes in MySQL Connector/J 3.0.3-dev (17 December 2002)

- Changed `charsToByte` in `SingleByteCharConverter` to be non-static.
- Changed `SingleByteCharConverter` to use lazy initialization of each converter.
- Fixed charset handling in `Fields.java`.
- Implemented `Connection.nativeSQL()`.
- More robust escape tokenizer: Recognize `--` comments, and allow nested escape sequences (see `testsuite.EscapeProcessingTest`).
- `DBMD.getImported/ExportedKeys()` now handles multiple foreign keys per table.
- Fixed `ResultSetMetaData.getPrecision()` returning incorrect values for some floating-point types.
- Fixed `ResultSetMetaData.getColumnTypeName()` returning `BLOB` for `TEXT` and `TEXT` for `BLOB` types.
- Fixed `Buffer.isLastDataPacket()` for 4.1 and newer servers.
- Added `CLIENT_LONG_FLAG` to be able to get more column flags (`isAutoIncrement()` being the most important).
- Because of above, implemented `ResultSetMetaData.isAutoIncrement()` to use `Field.isAutoIncrement()`.
- Honor `lower_case_table_names` when enabled in the server when doing table name comparisons in `DatabaseMetaData` methods.
- Some MySQL-4.1 protocol support (extended field info from selects).
- Use non-aliased table/column names and database names to fullly qualify tables and columns in `UpdatableResultSet` (requires MySQL-4.1 or newer).
- Allow user to alter behavior of `Statement`/`PreparedStatement.executeBatch()` via `continueBatchOnError` property (defaults to `true`).
- Check for connection closed in more `Connection` methods (`createStatement`, `prepareStatement`, `setTransactionIsolation`, `setAutoCommit`).
- More robust implementation of updatable result sets. Checks that *all* primary keys of the table have been selected.
- `LOAD DATA LOCAL INFILE ...` now works, if your server is configured to allow it. Can be

turned off with the `allowLoadLocalInfile` property (see the `README`).

- Substitute `'?'` for unknown character conversions in single-byte character sets instead of `'\0'`.
- `NamedPipeSocketFactory` now works (only intended for Windows), see `README` for instructions.

1.6.31. Changes in MySQL Connector/J 3.0.2-dev (08 November 2002)

- Fixed issue with updatable result sets and `PreparedStatements` not working.
- Fixed `ResultSet.setFetchDirection(FETCH_UNKNOWN)`.
- Fixed issue when calling `Statement.setFetchSize()` when using arbitrary values.
- Fixed incorrect conversion in `ResultSet.getLong()`.
- Implemented `ResultSet.updateBlob()`.
- Removed duplicate code from `UpdatableResultSet` (it can be inherited from `ResultSet`, the extra code for each method to handle updatability I thought might someday be necessary has not been needed).
- Fixed `UnsupportedEncodingException` thrown when "forcing" a character encoding via properties.
- Fixed various non-ASCII character encoding issues.
- Added driver property `useHostsInPrivileges`. Defaults to true. Affects whether or not `@hostname` will be used in `DBMD.getColumn/TablePrivileges`.
- All `DBMD` result set columns describing schemas now return `NULL` to be more compliant with the behavior of other JDBC drivers for other database systems (MySQL does not support schemas).
- Added SSL support. See `README` for information on how to use it.
- Properly restore connection properties when autoReconnecting or failing-over, including `autoCommit` state, and isolation level.
- Use `SHOW CREATE TABLE` when possible for determining foreign key information for `DatabaseMetaData`. Also allows cascade options for `DELETE` information to be returned.
- Escape `0x5c` character in strings for the SJIS charset.
- Fixed start position off-by-1 error in `Clob.getSubString()`.
- Implemented `Clob.truncate()`.
- Implemented `Clob.setString()`.
- Implemented `Clob.setAsciiStream()`.
- Implemented `Clob.setCharacterStream()`.
- Added `com.mysql.jdbc.MiniAdmin` class, which allows you to send `shutdown` command to MySQL server. This is intended to be used when "embedding" Java and MySQL server together in

an end-user application.

- Added `connectTimeout` parameter that allows users of JDK-1.4 and newer to specify a maxium time to wait to establish a connection.
- Failover and `autoReconnect` work only when the connection is in an `autoCommit(false)` state, in order to stay transaction-safe.
- Added `queriesBeforeRetryMaster` property that specifies how many queries to issue when failed over before attempting to reconnect to the master (defaults to 50).
- Fixed `DBMD.supportsResultSetConcurrency()` so that it returns true for `ResultSet.TYPE_SCROLL_INSENSITIVE` and `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`.
- Fixed `ResultSet.isLast()` for empty result sets (should return `false`).
- `PreparedStatement` now honors stream lengths in setBinary/Ascii/Character Stream() unless you set the connection property `useStreamLengthsInPrepStmts` to `false`.
- Removed some not-needed temporary object creation by smarter use of `Strings` in `EscapeProcessor`, `Connection` and `DatabaseMetaData` classes.

1.6.32. Changes in MySQL Connector/J 3.0.1-dev (21 September 2002)

- Fixed `ResultSet.getRow()` off-by-one bug.
- Fixed `RowDataStatic.getAt()` off-by-one bug.
- Added limited `Clob` functionality (`ResultSet.getClob()`, `PreparedStatemtent.setClob()`, `PreparedStatement.setObject(Clob)`.
- Added `socketTimeout` parameter to URL.
- `Connection.isClosed()` no longer "pings" the server.
- `Connection.close()` issues `rollback()` when `getAutoCommit()` is `false`.
- Added `paranoid` parameter, which sanitizes error messages by removing "sensitive" information from them (such as hostnames, ports, or usernames), as well as clearing "sensitive" data structures when possible.
- Fixed `ResultSetMetaData.isSigned()` for `TINYINT` and `BIGINT`.
- Charsets now automatically detected. Optimized code for single-byte character set conversion.
- Implemented `ResultSet.getCharacterStream()`.
- Added `LOCAL TEMPORARY` to table types in `DatabaseMetaData.getTableTypes()`.
- Massive code clean-up to follow Java coding conventions (the time had come).

1.6.33. Changes in MySQL Connector/J 3.0.0-dev (31 July 2002)

- **!!! LICENSE CHANGE !!!** The driver is now GPL. If you need non-GPL licenses, please contact me <mark@mysql.com>.
- JDBC-3.0 functionality including `Statement/PreparedStatement.getGeneratedKeys()` and `ResultSet.getURL()`.
- Performance enchancements: Driver is now 50-100% faster in most situations, and creates fewer temporary objects.
- Repackaging: New driver name is `com.mysql.jdbc.Driver`, old name still works, though (the driver is now provided by MySQL-AB).
- Better checking for closed connections in `Statement` and `PreparedStatement`.
- Support for streaming (row-by-row) result sets (see `README`) Thanks to Doron.
- Support for large packets (new addition to MySQL-4.0 protocol), see `README` for more information.
- JDBC Compliance: Passes all tests besides stored procedure tests.
- Fix and sort primary key names in `DBMetaData` (SF bugs 582086 and 582086).
- Float types now reported as `java.sql.Types.FLOAT` (SF bug 579573).
- `ResultSet.getTimestamp()` now works for `DATE` types (SF bug 559134).
- `ResultSet.getDate/Time/Timestamp` now recognizes all forms of invalid values that have been set to all zeroes by MySQL (SF bug 586058).
- Testsuite now uses Junit (which you can get from http://www.junit.org.
- The driver now only works with JDK-1.2 or newer.
- Added multi-host failover support (see `README`).
- General source-code cleanup.
- Overall speed improvements via controlling transient object creation in `MysqlIO` class when reading packets.
- Performance improvements in string handling and field metadata creation (lazily instantiated) contributed by Alex Twisleton-Wykeham-Fiennes.

1.6.34. Changes in MySQL Connector/J 2.0.14 (16 May 2002)

- More code cleanup
- PreparedStatement now releases resources on .close() (SF bug 553268)
- Quoted identifiers not used if server version does not support them. Also, if server started with --ansi or --sql-mode=ANSI_QUOTES then "" will be used as an identifier quote, otherwise `` ` `` will be used.
- ResultSet.getDouble() now uses code built into JDK to be more precise (but slower)
- LogicalHandle.isClosed() calls through to physical connection
- Added SQL profiling (to STDERR). Set "profileSql=true" in your JDBC url. See README for

more information.

- Fixed typo for relaxAutoCommit parameter.

1.6.35. Changes in MySQL Connector/J 2.0.13 (24 April 2002)

- More code cleanup.
- Fixed unicode chars being read incorrectly (SF bug 541088)
- Faster blob escaping for PrepStmt
- Added set/getPortNumber() to DataSource(s) (SF bug 548167)
- Added setURL() to MySQLXADataSource (SF bug 546019)
- PreparedStatement.toString() fixed (SF bug 534026)
- ResultSetMetaData.getColumnClassName() now implemented
- Rudimentary version of Statement.getGeneratedKeys() from JDBC-3.0 now implemented (you need to be using JDK-1.4 for this to work, I believe)
- DBMetaData.getIndexInfo() - bad PAGES fixed (SF BUG 542201)

1.6.36. Changes in MySQL Connector/J 2.0.12 (07 April 2002)

- General code cleanup.
- Added getIdleFor() method to Connection and MysqlLogicalHandle.
- Relaxed synchronization in all classes, should fix 520615 and 520393.
- Added getTable/ColumnPrivileges() to DBMD (fixes 484502).
- Added new types to getTypeInfo(), fixed existing types thanks to Al Davis and Kid Kalanon.
- Added support for BIT types (51870) to PreparedStatement.
- Fixed getRow() bug (527165) in ResultSet
- Fixes for ResultSet updatability in PreparedStatement.
- Fixed time zone off by 1-hour bug in PreparedStatement (538286, 528785).
- ResultSet: Fixed updatability (values being set to null if not updated).
- DataSources - fixed setUrl bug (511614, 525565), wrong datasource class name (532816, 528767)
- Added identifier quoting to all DatabaseMetaData methods that need them (should fix 518108)
- Added support for YEAR type (533556)
- ResultSet.insertRow() should now detect auto_increment fields in most cases and use that value in the new row. This detection will not work in multi-valued keys, however, due to the fact that the

MySQL protocol does not return this information.

- ResultSet.refreshRow() implemented.
- Fixed testsuite.Traversal afterLast() bug, thanks to Igor Lastric.

1.6.37. Changes in MySQL Connector/J 2.0.11 (27 January 2002)

- Fixed missing DELETE_RULE value in DBMD.getImported/ExportedKeys() and getCrossReference().
- Full synchronization of Statement.java.
- More changes to fix "Unexpected end of input stream" errors when reading BLOBs. This should be the last fix.

1.6.38. Changes in MySQL Connector/J 2.0.10 (24 January 2002)

- Fixed spurious "Unexpected end of input stream" errors in MysqlIO (bug 507456).
- Fixed null-pointer-exceptions when using MysqlConnectionPoolDataSource with Websphere 4 (bug 505839).

1.6.39. Changes in MySQL Connector/J 2.0.9 (13 January 2002)

- Ant build was corrupting included jar files, fixed (bug 487669).
- Fixed extra memory allocation in MysqlIO.readPacket() (bug 488663).
- Implementation of DatabaseMetaData.getExported/ImportedKeys() and getCrossReference().
- Full synchronization on methods modifying instance and class-shared references, driver should be entirely thread-safe now (please let me know if you have problems)
- DataSource implementations moved to org.gjt.mm.mysql.jdbc2.optional package, and (initial) implementations of PooledConnectionDataSource and XADataSource are in place (thanks to Todd Wolff for the implementation and testing of PooledConnectionDataSource with IBM WebSphere 4).
- Added detection of network connection being closed when reading packets (thanks to Todd Lizambri).
- Fixed quoting error with escape processor (bug 486265).
- Report batch update support through DatabaseMetaData (bug 495101).
- Fixed off-by-one-hour error in PreparedStatement.setTimestamp() (bug 491577).
- Removed concatenation support from driver (the '||' operator), as older versions of VisualAge seem to be the only thing that use it, and it conflicts with the logical '||' operator. You will need to start mysqld with the "--ansi" flag to use the '||' operator as concatenation (bug 491680)
- Fixed casting bug in PreparedStatement (bug 488663).

1.6.40. Changes in MySQL Connector/J 2.0.8 (25 November 2001)

- Batch updates now supported (thanks to some inspiration from Daniel Rall).
- XADataSource/ConnectionPoolDataSource code (experimental)
- PreparedStatement.setAnyNumericType() now handles positive exponents correctly (adds "+" so MySQL can understand it).
- DatabaseMetaData.getPrimaryKeys() and getBestRowIdentifier() are now more robust in identifying primary keys (matches regardless of case or abbreviation/full spelling of Primary Key in Key_type column).

1.6.41. Changes in MySQL Connector/J 2.0.7 (24 October 2001)

- PreparedStatement.setCharacterStream() now implemented
- Fixed dangling socket problem when in high availability (autoReconnect=true) mode, and finalizer for Connection will close any dangling sockets on GC.
- Fixed ResultSetMetaData.getPrecision() returning one less than actual on newer versions of MySQL.
- ResultSet.getBlob() now returns null if column value was null.
- Character sets read from database if useUnicode=true and characterEncoding is not set. (thanks to Dmitry Vereshchagin)
- Initial transaction isolation level read from database (if avaialable) (thanks to Dmitry Vereshchagin)
- Fixed DatabaseMetaData.supportsTransactions(), and supportsTransactionIsolationLevel() and getTypeInfo() SQL_DATETIME_SUB and SQL_DATA_TYPE fields not being readable.
- Fixed PreparedStatement generating SQL that would end up with syntax errors for some queries.
- Fixed ResultSet.isAfterLast() always returning false.
- Fixed time zone issue in PreparedStatement.setTimestamp() (thanks to Erik Olofsson)
- Captialize type names when "captializeTypeNames=true" is passed in URL or properties (for WebObjects, thanks to Anjo Krank)
- Updatable result sets now correctly handle NULL values in fields.
- PreparedStatement.setDouble() now uses full-precision doubles (reverting a fix made earlier to truncate them).
- PreparedStatement.setBoolean() will use 1/0 for values if your MySQL version is 3.21.23 or higher.

1.6.42. Changes in MySQL Connector/J 2.0.6 (16 June 2001)

- Fixed PreparedStatement parameter checking

- Fixed case-sensitive column names in ResultSet.java

1.6.43. Changes in MySQL Connector/J 2.0.5 (13 June 2001)

- Fixed ResultSet.getBlob() ArrayIndex out-of-bounds
- Fixed ResultSetMetaData.getColumnTypeName for TEXT/BLOB
- Fixed ArrayIndexOutOfBounds when sending large BLOB queries (Max size packet was not being set)
- Added ISOLATION level support to Connection.setIsolationLevel()
- Fixed NPE on PreparedStatement.executeUpdate() when all columns have not been set.
- Fixed data parsing of TIMESTAMPs with 2-digit years
- Added Byte to PreparedStatement.setObject()
- ResultSet.getBoolean() now recognizes '-1' as `true`
- ResultSet has +/-Inf/inf support
- ResultSet.insertRow() works now, even if not all columns are set (they will be set to "NULL")
- DataBaseMetaData.getCrossReference() no longer ArrayIndexOOB
- getObject() on ResultSet correctly does TINYINT->Byte and SMALLINT->Short

1.6.44. Changes in MySQL Connector/J 2.0.3 (03 December 2000)

- Implemented getBigDecimal() without scale component for JDBC2.
- Fixed composite key problem with updatable result sets.
- Added detection of -/+INF for doubles.
- Faster ASCII string operations.
- Fixed incorrect detection of MAX_ALLOWED_PACKET, so sending large blobs should work now.
- Fixed off-by-one error in java.sql.Blob implementation code.
- Added "ultraDevHack" URL parameter, set to "true" to allow (broken) Macromedia UltraDev to use the driver.

1.6.45. Changes in MySQL Connector/J 2.0.1 (06 April 2000)

- Fixed RSMD.isWritable() returning wrong value. Thanks to Moritz Maass.
- Cleaned up exception handling when driver connects
- Columns that are of type TEXT now return as Strings when you use getObject()

- DatabaseMetaData.getPrimaryKeys() now works correctly wrt to key_seq. Thanks to Brian Slesinsky.
- No escape processing is done on PreparedStatements anymore per JDBC spec.
- Fixed many JDBC-2.0 traversal, positioning bugs, especially wrt to empty result sets. Thanks to Ron Smits, Nick Brook, Cessar Garcia and Carlos Martinez.
- Fixed some issues with updatability support in ResultSet when using multiple primary keys.

1.6.46. Changes in MySQL Connector/J 2.0.0pre5 (21 February 2000)

- Fixed Bad Handshake problem.

1.6.47. Changes in MySQL Connector/J 2.0.0pre4 (10 January 2000)

- Fixes to ResultSet for insertRow() - Thanks to Cesar Garcia
- Fix to Driver to recognize JDBC-2.0 by loading a JDBC-2.0 class, instead of relying on JDK version numbers. Thanks to John Baker.
- Fixed ResultSet to return correct row numbers
- Statement.getUpdateCount() now returns rows matched, instead of rows actually updated, which is more SQL-92 like.

10-29-99

- Statement/PreparedStatement.getMoreResults() bug fixed. Thanks to Noel J. Bergman.
- Added Short as a type to PreparedStatement.setObject(). Thanks to Jeff Crowder
- Driver now automagically configures maximum/preferred packet sizes by querying server.
- Autoreconnect code uses fast ping command if server supports it.
- Fixed various bugs wrt. to packet sizing when reading from the server and when alloc'ing to write to the server.

1.6.48. Changes in MySQL Connector/J 2.0.0pre (17 August 1999)

- Now compiles under JDK-1.2. The driver supports both JDK-1.1 and JDK-1.2 at the same time through a core set of classes. The driver will load the appropriate interface classes at runtime by figuring out which JVM version you are using.
- Fixes for result sets with all nulls in the first row. (Pointed out by Tim Endres)
- Fixes to column numbers in SQLExceptions in ResultSet (Thanks to Blas Rodriguez Somoza)
- The database no longer needs to specified to connect. (Thanks to Christian Motschke)

1.6.49. Changes in MySQL Connector/J 1.2b (04 July 1999)

- Better Documentation (in progress), in doc/mm.doc/book1.html
- DBMD now allows null for a column name pattern (not in spec), which it changes to '%'.
- DBMD now has correct types/lengths for getXXX().
- ResultSet.getDate(), getTime(), and getTimestamp() fixes. (contributed by Alan Wilken)
- EscapeProcessor now handles \{ \} and { or } inside quotes correctly. (thanks to Alik for some ideas on how to fix it)
- Fixes to properties handling in Connection. (contributed by Juho Tikkala)
- ResultSet.getObject() now returns null for NULL columns in the table, rather than bombing out. (thanks to Ben Grosman)
- ResultSet.getObject() now returns Strings for types from MySQL that it doesn't know about. (Suggested by Chris Perdue)
- Removed DataInput/Output streams, not needed, 1/2 number of method calls per IO operation.
- Use default character encoding if one is not specified. This is a work-around for broken JVMs, because according to spec, EVERY JVM must support "ISO8859_1", but they don't.
- Fixed Connection to use the platform character encoding instead of "ISO8859_1" if one isn't explicitly set. This fixes problems people were having loading the character- converter classes that didn't always exist (JVM bug). (thanks to Fritz Elfert for pointing out this problem)
- Changed MysqlIO to re-use packets where possible to reduce memory usage.
- Fixed escape-processor bugs pertaining to {} inside quotes.

1.6.50. Changes in MySQL Connector/J 1.2a (14 April 1999)

- Fixed character-set support for non-Javasoft JVMs (thanks to many people for pointing it out)
- Fixed ResultSet.getBoolean() to recognize 'y' & 'n' as well as '1' & '0' as boolean flags. (thanks to Tim Pizey)
- Fixed ResultSet.getTimestamp() to give better performance. (thanks to Richard Swift)
- Fixed getByte() for numeric types. (thanks to Ray Bellis)
- Fixed DatabaseMetaData.getTypeInfo() for DATE type. (thanks to Paul Johnston)
- Fixed EscapeProcessor for "fn" calls. (thanks to Piyush Shah at locomotive.org)
- Fixed EscapeProcessor to not do extraneous work if there are no escape codes. (thanks to Ryan Gustafson)
- Fixed Driver to parse URLs of the form "jdbc:mysql://host:port" (thanks to Richard Lobb)

1.6.51. Changes in MySQL Connector/J 1.1i (24 March 1999)

- Fixed Timestamps for PreparedStatements
- Fixed null pointer exceptions in RSMD and RS
- Re-compiled with jikes for valid class files (thanks ms!)

1.6.52. Changes in MySQL Connector/J 1.1h (08 March 1999)

- Fixed escape processor to deal with unmatched { and } (thanks to Craig Coles)
- Fixed escape processor to create more portable (between DATETIME and TIMESTAMP types) representations so that it will work with BETWEEN clauses. (thanks to Craig Longman)
- MysqlIO.quit() now closes the socket connection. Before, after many failed connections some OS's would run out of file descriptors. (thanks to Michael Brinkman)
- Fixed NullPointerException in Driver.getPropertyInfo. (thanks to Dave Potts)
- Fixes to MysqlDefs to allow all *text fields to be retrieved as Strings. (thanks to Chris at Leverage)
- Fixed setDouble in PreparedStatement for large numbers to avoid sending scientific notation to the database. (thanks to J.S. Ferguson)
- Fixed getScale() and getPrecision() in RSMD. (contrib'd by James Klicman)
- Fixed getObject() when field was DECIMAL or NUMERIC (thanks to Bert Hobbs)
- DBMD.getTables() bombed when passed a null table-name pattern. Fixed. (thanks to Richard Lobb)
- Added check for "client not authorized" errors during connect. (thanks to Hannes Wallnoefer)

1.6.53. Changes in MySQL Connector/J 1.1g (19 February 1999)

- Result set rows are now byte arrays. Blobs and Unicode work bidriectonally now. The useUnicode and encoding options are implemented now.
- Fixes to PreparedStatement to send binary set by setXXXStream to be sent untouched to the MySQL server.
- Fixes to getDriverPropertyInfo().

1.6.54. Changes in MySQL Connector/J 1.1f (31 December 1998)

- Changed all ResultSet fields to Strings, this should allow Unicode to work, but your JVM must be able to convert between the character sets. This should also make reading data from the server be a bit quicker, because there is now no conversion from StringBuffer to String.
- Changed PreparedStatement.streamToString() to be more efficient (code from Uwe Schaefer).

- URL parsing is more robust (throws SQL exceptions on errors rather than NullPointerExceptions)
- PreparedStatement now can convert Strings to Time/Date values via setObject() (code from Robert Currey).
- IO no longer hangs in Buffer.readInt(), that bug was introduced in 1.1d when changing to all byte-arrays for result sets. (Pointed out by Samo Login)

1.6.55. Changes in MySQL Connector/J 1.1b (03 November 1998)

- Fixes to DatabaseMetaData to allow both IBM VA and J-Builder to work. Let me know how it goes. (thanks to Jac Kersing)
- Fix to ResultSet.getBoolean() for NULL strings (thanks to Barry Lagerweij)
- Beginning of code cleanup, and formatting. Getting ready to branch this off to a parallel JDBC-2.0 source tree.
- Added "final" modifier to critical sections in MysqlIO and Buffer to allow compiler to inline methods for speed.

9-29-98

- If object references passed to setXXX() in PreparedStatement are null, setNull() is automatically called for you. (Thanks for the suggestion goes to Erik Ostrom)
- setObject() in PreparedStatement will now attempt to write a serialized representation of the object to the database for objects of Types.OTHER and objects of unknown type.
- Util now has a static method readObject() which given a ResultSet and a column index will re-instantiate an object serialized in the above manner.

1.6.56. Changes in MySQL Connector/J 1.1 (02 September 1998)

- Got rid of "ugly hack" in MysqlIO.nextRow(). Rather than catch an exception, Buffer.isLastDataPacket() was fixed.
- Connection.getCatalog() and Connection.setCatalog() should work now.
- Statement.setMaxRows() works, as well as setting by property maxRows. Statement.setMaxRows() overrides maxRows set via properties or url parameters.
- Automatic re-connection is available. Because it has to "ping" the database before each query, it is turned off by default. To use it, pass in "autoReconnect=true" in the connection URL. You may also change the number of reconnect tries, and the initial timeout value via "maxReconnects=n" (default 3) and "initialTimeout=n" (seconds, default 2) parameters. The timeout is an exponential backoff type of timeout; for example, if you have initial timeout of 2 seconds, and maxReconnects of 3, then the driver will timeout 2 seconds, 4 seconds, then 16 seconds between each re-connection attempt.

1.6.57. Changes in MySQL Connector/J 1.0 (24 August 1998)

- Fixed handling of blob data in Buffer.java
- Fixed bug with authentication packet being sized too small.
- The JDBC Driver is now under the LPGL

8-14-98

- Fixed Buffer.readLenString() to correctly read data for BLOBS.
- Fixed PreparedStatement.stringToStream to correctly read data for BLOBS.
- Fixed PreparedStatement.setDate() to not add a day. (above fixes thanks to Vincent Partington)
- Added URL parameter parsing (?user=... and so forth).

1.6.58. Changes in MySQL Connector/J 0.9d (04 August 1998)

- Big news! New package name. Tim Endres from ICE Engineering is starting a new source tree for GNU GPL'd Java software. He's graciously given me the org.gjt.mm package directory to use, so now the driver is in the org.gjt.mm.mysql package scheme. I'm "legal" now. Look for more information on Tim's project soon.
- Now using dynamically sized packets to reduce memory usage when sending commands to the DB.
- Small fixes to getTypeInfo() for parameters, and so forth.
- DatabaseMetaData is now fully implemented. Let me know if these drivers work with the various IDEs out there. I've heard that they're working with JBuilder right now.
- Added JavaDoc documentation to the package.
- Package now available in .zip or .tar.gz.

1.6.59. Changes in MySQL Connector/J 0.9 (28 July 1998)

- Implemented getTypeInfo(). Connection.rollback() now throws an SQLException per the JDBC spec.
- Added PreparedStatement that supports all JDBC API methods for PreparedStatement including InputStreams. Please check this out and let me know if anything is broken.
- Fixed a bug in ResultSet that would break some queries that only returned 1 row.
- Fixed bugs in DatabaseMetaData.getTables(), DatabaseMetaData.getColumns() and DatabaseMetaData.getCatalogs().
- Added functionality to Statement that allows executeUpdate() to store values for IDs that are automatically generated for AUTO_INCREMENT fields. Basically, after an executeUpdate(), look at the SQLWarnings for warnings like "LAST_INSERTED_ID = 'some number', COMMAND = 'your SQL query'". If you are using AUTO_INCREMENT fields in your tables and are executing a lot of executeUpdate()s on one Statement, be sure to clearWarnings() every so often to save memory.

1.6.60. Changes in MySQL Connector/J 0.8 (06 July 1998)

- Split MysqlIO and Buffer to separate classes. Some ClassLoaders gave an IllegalAccess error for some fields in those two classes. Now mm.mysql works in applets and all classloaders. Thanks to Joe Ennis <jce@mail.boone.com> for pointing out the problem and working on a fix with me.

1.6.61. Changes in MySQL Connector/J 0.7 (01 July 1998)

- Fixed DatabaseMetadata problems in getColumns() and bug in switch statement in the Field constructor. Thanks to Costin Manolache <costin@tdiinc.com> for pointing these out.

1.6.62. Changes in MySQL Connector/J 0.6 (21 May 1998)

- Incorporated efficiency changes from Richard Swift <Richard.Swift@kanatek.ca> in `MysqlIO.java` and `ResultSet.java`:
- We're now 15% faster than gwe's driver.
- Started working on `DatabaseMetaData`.
- The following methods are implemented:
  - `getTables()`
  - `getTableTypes()`
  - `getColumns`
  - `getCatalogs()`