MySQL Connector/J Documentation

by Mark Matthews

MySQL Connector/J Documentation

by Mark Matthews

Table of Contents

List of Tables

List of Examples

Chapter 1. Introduction

What is MySQL Connector/J?

MySQL Connector/J is an implementation of Sun's JDBC 3.0 API for the MySQL relational database server. It strives to conform as much as possible to the JDBC API as specified by JavaSoft. It is known to work with many third-party products, including:

Application Servers

- Apache Tomcat [http://jakarta.apache.org/]
- JBoss [http://www.jboss.org/]
- Weblogic [http://www.beasys.com/]
- IBM WebSphere [http://www-3.ibm.com/software/info1/websphere/]

Object Relational Mapping Tools

- Hibernate [http://hibernate.sourceforge.net/]
- Apache ObjectRelationalBridge [http://db.apache.org/ojb/]
- CocoBase [http://www.thoughtinc.com/]
- Kodo [http://www.solarmetric.com/]

Development Environments

- Eclipse [http://www.eclipse.org/]
- Borland JBuilder [http://www.borland.com/jbuilder/]
- IBM WebSphere Studio [http://www-3.ibm.com/software/awdtools/studioappdev/]

Release Notes

Known Issues

Implementation Notes (By java.sql and javax.sql Interface/Class)

MySQL Connector/J passes all of the tests in Sun's JDBC compliance testsuite except for tests requiring stored procedures (which MySQL does not have at this time). 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(), etc) 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 table types 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. Also, any tables referenced by the query that created the streaming result will be locked until all of the results have been read or the connection closed.

- ResultSetMetaData

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

- Statement

Un-implemented Functionality

The following methods in the JDBC API have not been implemented yet. They rely on functionality that at this time is not present in the MySQL server:

- Blob.truncate()

- Connection.setSavePoint() is not supported in versions earlier than 3.1.1
- Connection.prepareCall(String) is not supported in versions earlier than 3.1.1
- Connection.releaseSavepoint(Savepoint) is not supported in versions earlier than 3.1.1
- Connection.rollback(Savepoint) is not supported in versions earlier than 3.1.1

- PreparedStatement.setArray(int, Array)
- PreparedStatement.setRef()
- PreparedStatement.getParameterMetaData()

- ResultSet.getArray(int)
- ResultSet.getArray(colName)
- ResultSet.getRef(int)
- ResultSet.getRef(String)
- ResultSet.rowDeleted()
- ResultSet.rowInserted()
- ResultSet.rowUpdated()
- ResultSet.updateArray(int, Array)
- ResultSet.updateArray(String, Array)
- ResultSet.updateRef(int, Ref)
- ResultSet.updateRef(String, Ref)

Chapter 2. Installation

This chapter contains instructions for installing the MySQL Connector/J drivers on Microsoft Windows and UNIX platforms. If you are on another platform that supports Java and the JDBC API then substitute commands appropriate for your system.

System Requirements

Java Versions Supported

MySQL Connector/J supports Java-2 JVMs, including JDK-1.2.x, JDK-1.3.x and JDK-1.4.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.

MySQL Server Versions Supported

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

Installing MySQL Connector/J

MySQL Connector/J is distributed as a .zip or .tar.gz archive containing the sources, the class files and a class-file only "binary" .jar archive named "mysql-connector-java-[version]-bin.jar". You will need to use the appropriate gui or command-line utility to un-archive the distribution (for example, WinZip for the .zip archive, and "tar" for the .tar.gz archive).

Setting the CLASSPATH (For Standalone Use)

Once you have un-archived the distribution archive, you can install the driver in one of two ways: Either copy the "com" and "org" subdirectories and all of their contents to anywhere you like, and put the directory holding the "com" and "org" subdirectories in your classpath, or put mysql-connector-java-[version]-bin.jar in your classpath, either by adding the FULL path to it to your CLASSPATH enviornment variable, or by copying the .jar file to $JAVA_HOME/jre/lib/ext. 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 2.1. 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.

Example 2.2. Setting the CLASSPATH Under Microsoft Windows 9X

The following is an example of setting the CLASSPATH under Microsoft Windows 95, 98, ME:

```
C:\> set CLASSPATH=\path\to\mysql-connector-java-[version]-bin.jar;%CLASSPATH%
```

This command can be added as the last line in AUTOEXEC.BAT. If this is done the MySQL Connector/J driver will be made available to all Java applications that run on the Windows 9x system. This setting will require the computer to be rebooted before the changes will take effect.

Driver Class Name and JDBC URL Format

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 backwards-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][=propertyValue1][&propertyNa
```

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.

Starting with version 3.0.12 and 3.1.2, Connector/J supports multiple hosts with ports, separated by commas:

```
jdbc:mysql://[host:port],[host:port].../[database][?propertyName1][=propertyValue1][&propertyName2][
```

If the database is not specified, the connection will be made with no 'current' database. In this case, you will need to either call the 'setCatalog()' method on the Connection instance, issue a 'USE dbname' query 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 good 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).

You may specify additional properties to the JDBC driver, either by placing them in a java.util.Properties instance and passing that instance to the DriverManager when you connect, or by adding them to the end of your JDBC URL as name-value pairs. The first property needs to be preceeded with a '?' character, and additional name-value pair properties are separated by an '&' character. The properties, their definitions and their default values are covered in the following table:

Table 2.1. Connection Properties

Property Name	Definition	Required?	Default Value	Since Version
user	The user to connect as	No		All

Property Name	Definition	Required?	Default Value	Since Version
password	The password to use when connecting	No		All
gatherPerfMetrics	Should the driver gather performance metrics, and report them via the configured logger every 'reportMetricsIntervalMillis' milliseconds?	false	false	3.1.2
reportMetricsIntervalMillis	If 'gatherPerfMetrics' is enabled, how often should they be logged (in ms)?	false	30000	3.1.2
logSlowQueries	Should queries that take longer than 'slowQueryThresholdMillis' be logged?	false	false	3.1.2
slowQueryThresholdMillis	If 'logSlowQueries' is enabled, how long should a query (in ms) before it is logged as 'slow'?	false	2000	3.1.2
failOverReadOnly	When failing over in autoReconnect mode, should the connection be set to 'read-only'?	false	true	3.0.12
allowLoadLocalInfile	Should the driver allow use of 'LOAD DATA LOCAL INFILE...' (defaults to 'true').	false	true	3.0.3
allowMultiQueries	Allow the use of ';' to delimit multiple queries during one statement (true/false, defaults to 'false'	false	false	3.1.1
autoReconnect	Should the driver try to re-establish bad connections?	false	false	1.1
autoReconnectForPools	Use a reconnection strategy appropriate for connection pools (defaults to 'false')	false	false	3.0.8
cacheCallableStmts	Should the driver cache the parsing stage of CallableStatements	false	false	3.1.2
cachePrepStmts	Should the driver cache the parsing stage of PreparedStatements?	false	false	3.0.10
cacheResultSet-	Should the driver	false	false	3.1.1

Property Name	Definition	Required?	Default Value	Since Version
Metadata	cache ResultSet-MetaData for Statements and Prepared-Statements? (Req. JDK-1.4+, true/false, default 'false')			
capitalizeTypeNames	Capitalize type names in Database-MetaData? (usually only useful when using WebObjects, true/false, defaults to 'false')	false	false	2.0.7
clobberStreamin-gResults	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.	false	false	3.0.9
continueBatchOnErr-or	Should the driver continue processing batch commands if one statement fails. The JDBC spec allows either way (defaults to 'true').	false	true	3.0.3
useServerPrepStmts	Use server-side prepared statements if the server supports them? (defaults to 'true').	false	true	3.1.0
emulateLocators	N/A	false	false	3.1.0
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?	false	false	3.1.2
ignoreNonTxTables	Ignore non-transactional table warning for rollback? (defaults to 'false').	false	false	3.0.9
interactiveClient	Set the CLI-ENT_INTERACTIV E flag, which tells MySQL to timeout connections based on	false	false	3.1.0

Property Name	Definition	Required?	Default Value	Since Version
	INTERACT-IVE_TIMEOUT instead of WAIT_TIMEOUT			
paranoid	Take measures to prevent exposure sensitive information in error messages and clear data structures holding sensitive data when possible? (defaults to 'false')	false	false	3.0.1
pedantic	Follow the JDBC spec to the letter.	false	false	3.0.0
profileSQL	Trace queries and their execution/fetch times to the configured logger (true/false) defaults to 'false'	false	false	3.1.0
reconnectAtTxEnd	If autoReconnect is set to true, should the driver attempt reconnectionsat the end of every transaction?	false	false	3.0.10
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')?	false	false	2.0.13
requireSSL	Require SSL connection if useSSL=true? (defaults to 'false').	false	false	3.1.0
roundRobinLoadBalance	When autoReconnect is enabled, and failoverReadonly is false, should we pick hosts to connect to on a round-robin basis?	false	false	3.1.2
strictFloatingPoint	Used only in older versions of compliance test	false	false	3.0.0
strictUpdates	Should the driver do strict checking (all primary keys selected) of updatable result sets (true, false, defaults to 'true')?	false	true	3.0.4
useCompression	Use zlib compression	false	false	3.1.0

Property Name	Definition	Required?	Default Value	Since Version
	when communicating with the server (true/false)? Defaults to 'false'.			
useHostsInPrivileges	Add '@hostname' to users in DatabaseMetaData.getColumn/TablePrivileges() (true/false), defaults to 'true'.	false	true	3.0.2
useNewIO	Should the driver use the java.nio.* interfaces for network communication (true/false), defaults to 'false'	false	false	3.1.0
useSSL	Use SSL when communicating with the server (true/false), defaults to 'false'	false	false	3.0.2
useStreamLengthsInPrepStmts	Honor stream length parameter in PreparedStatement/ResultSet.setXXXStream() method calls (true/false, defaults to 'true')?	false	true	3.0.2
useTimezone	Convert time/date types between client and server timezones (true/false, defaults to 'false')?	false	false	3.0.2
ultraDevHack	Create PreparedStatements for prepareCall() when required, because UltraDev is broken and issues a prepareCall() for _all_ statements? (true/false, defaults to 'false')	false	false	2.0.3
useUnbufferedInput	Don't use BufferedInputStream for reading data from the server	false	true	3.0.11
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	false	false	1.1g

Property Name	Definition	Required?	Default Value	Since Version
	character set that MySQL either doesn't natively support (such as UTF-8), true/false, defaults to 'true'			
useUsageAdvisor	Should the driver issue 'usage' warnings advising proper and efficient usage of JDBC and MySQL Connector/J to the log (true/false, defaults to 'false')?	false	false	3.1.1
callableStmtCacheSize	If 'cacheCallableStmts' is enabled, how many callable statements should be cached?	false	100	3.1.2
connectTimeout	Timeout for socket connect (in milliseconds), with 0 being no timeout. Only works on JDK-1.4 or newer. Defaults to '0'.	false	0	3.0.1
initialTimeout	If autoReconnect is enabled, the initial time to wait between re-connect attempts (in seconds, defaults to '2').	false	2	1.1
maxReconnects	Maximum number of reconnects to attempt if autoReconnect is true, default is '3'.	false	3	1.1
maxRows	The maximum number of rows to return (0, the default means return all rows).	false	-1	all versions
metadataCacheSize	The number of queries to cacheResultSetMetadata for if cacheResultSetMetaData is set to 'true' (default 50)	false	50	3.1.1
prepStmtCacheSize	If prepared statement caching is enabled, how many prepared statements should be cached?	false	25	3.0.10
prepStmtCacheSqlLimit	If prepared statement caching is enabled, what's the largest SQL the driver will	false	256	3.0.10

Property Name	Definition	Required?	Default Value	Since Version
	cache the parsing for?			
queriesBeforeRetry-Master	Number of queries to issue before falling back to master when failed over (when using multi-host failover). Whichever condition is met first, 'queriesBeforeRetry-Master' or 'secondsBeforeRetryMaster' will cause an attempt to be made to reconnect to the master. Defaults to 50.	false	50	3.0.2
secondsBeforeRetry-Master	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	false	30	3.0.2
socketTimeout	Timeout on network socket operations (0, the default means no timeout).	false	0	3.0.1
characterEncoding	If 'useUnicode' is set to true, what character encoding should the driver use when dealing with strings? (defaults is to 'autodetect')	false		1.1g
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)	false	com.mysql.jdbc.log.StandardLogger	3.1.1
profileSql	Deprecated, use 'profileSQL' instead. Trace queries and their execution/fetch	false		2.0.14

Property Name	Definition	Required?	Default Value	Since Version
	times on STDERR (true/false) defaults to 'false'			
serverTimezone	Override detection/ mapping of timezone. Used when timezone from server doesn't map to Java timezone	false		3.0.2
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.	false	com.mysql.jdbc.StandardSocketFactory	3.0.3

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.

Installing MySQL Connector/J for Use With Servlets/JSP/EJB

If you want to use MySQL Connector/J with a servlet engine or 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.

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

You can also use the MysqlDataSource, MysqlConnectionPoolDataSource or MysqlXADataSource classes in the com.mysql.jdbc.jdbc2.optional and com.mysql.jdbc.jdbc2.optional.xa packages, if your J2EE application server supports or requires them. 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

What's Next?

Once the driver has been installed into your CLASSPATH or application server, the driver is ready for use! Taking a look at the next chapter containing programming information might be a good idea, however.

Chapter 3. Developing Applications with MySQL Connector/J

...

Basic Functionality

Registering MySQL Connector/J With the JDBC DriverManager

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.

Example 3.1. Registering the Driver With the DriverManager

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
        }
}
```

Opening a Connection to MySQL

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 3.2. 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/test?user=monty&pa

            // 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 Statements and PreparedStatements, as well as retrieve metadata about the database. This is explained in the following sections.

Creating a Statement Instance

Statements allow you to execute basic SQL queries and retrieve the results through the ResultSet class which is described later. To get a Statement object, you call the createStatement() method on the Connection object you have retrieved via the DriverManager.getConnection() method. Once you have a Statement object, you can execute a SELECT query by calling the executeQuery(String SQL) 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 an UPDATE/INSERT/DELETE query. If the query was a SELECT query, you can retrieve the results by calling the getResultSet() method. If the query was an UPDATE/INSERT/DELETE query, you can retrieve the affected rows count by calling getUpdateCount() on the Statement instance.

Executing a SELECT Query

Example 3.3. Using java.sql.Statement to Execute a SELECT Query

```
// assume 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;
    }
}
```

Advanced Functionality

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()`, `setAsiiStream()`, `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 configuration variable '*`character_set`*' for server versions older than 4.1.0 and '*`character_set_server`*' for server versions 4.1.0 and newer. See the "Server Character Set and Collation [http://www.mysql.com/doc/en/Charset-server.html]" section in the MySQL server manual for more information.

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 3.1. MySQL to Java Encoding Name Translations

MySQL Character Set Name	Java-Style Character Encoding Name
usa7	US-ASCII
big5	Big5
gbk	GBK

MySQL Character Set Name	Java-Style Character Encoding Name
sjis	SJIS
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
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.

Stored Procedures

Starting with MySQL 5.0 and Connector/J 3.1.1, 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 3.4. 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 3.5. 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 per-forms 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 created the stored procedure), JDBC requires that they be specified before statement execution using the various `registerOutputParameter()` methods in the `CallableStatement` interface:

   Example 3.6. 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
    //

    cStmt.registerOutParameter(2);

    //
    // 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'
    //

    cStmt.registerOutParameter("inOutParam");

    //
    // 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 3.7. 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.

   While `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 3.8. 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(1); // index-based

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

    ...
```

Connecting over 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 truststore
```

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 keystore
```

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, 168, 55, 103, 215,
 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, 185, 178, 100, 215,
Session ID:  {163, 227, 84, 53, 81, 127, 252, 254, 178, 179, 68, 63, 182, 158, 30, 11, 150, 79, 170
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.

Chapter 4. How to Report 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 than 'jdbc:mysql:///test', then 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/.

Chapter 5. Troubleshooting

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. If you have further issues, see the "SUPPORT" section.

5.1.

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.

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 or not 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 security problems with your MySQL server.

5.2.

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

One of two things are happening. Either the driver is not in your CLASSPATH (see the "INSTALLATION" section above), or your URL format is incorrect (see "Developing Applications with MySQL Connector/J").

5.3.

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
```

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).

5.4.

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

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 "Developing Applications with MySQL 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 5.1. 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 41000 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) || "41000".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));
    }
```

5.5.
I'm trying to use JDBC-2.0 updatable result sets, and I get an exception saying my result set is not updatable.

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 all of the primary key(s) and the query can only span one table (i.e. no joins). This is outlined in the JDBC specification.

Appendix A. Reference

Type Conversions Supported by MySQL Connector/J

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.

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

Table A.1. 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

Appendix B. ChangeLog

```
# Changelog
# $Id: CHANGES,v 1.38.4.62 2004/02/11 02:15:16 mmatthew Exp $

xx-xx-04 - Version 3.1.2-alpha (not yet released)

    - Fixed stored procedure parameter parsing info when size was
      specified for a parameter (i.e. 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 process).

    - 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.

02-14-04 - Version 3.1.1-alpha

    - 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 multibyte 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 WARNINGS').

    - Default result set type changed to TYPE_FORWARD_ONLY
      (JDBC compliance).

    - Centralized setting of result set type and concurrency.

    - Re-factored 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 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).

  - Fix for BUG#1673, where DatabaseMetaData.getColumns() is not
    returning correct column ordinal info for non '%' column name patterns.

  - 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 > 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.

    - Fixed BUG#2359, NULL values for numeric types in binary
      encoded result sets causing NullPointerExceptions.

    - 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 BUG#2502 -- charset conversion issue in getTables().

    - Implemented multiple result sets returned from a statement
      or stored procedure.

    - Fixed BUG#2606 -- Server side prepared statements not returning
      datatype 'YEAR' correctly.

    - Enabled streaming of result sets from server-side prepared
      statements.

    - Fixed BUG#2623 -- Class-cast exception when using
      scrolling result sets and server-side prepared statements.

    - Merged unbuffered input code from 3.0.

    - Fixed ConnectionProperties that weren't properly exposed
      via accessors, cleaned up ConnectionProperties code.

    - Fixed BUG#2671, NULL fields not being encoded correctly in
      all cases in server side prepared statements.

    - Fixed rare buffer underflow when writing numbers into buffers
      for sending prepared statement execution requests.

02-18-03 - Version 3.1.0-alpha

    - 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).

05-27-04 - Version 3.0.14-production
    - Fixed BUG#3923 - URL parsing error

05-27-04 - Version 3.0.13-production
    - Fixed BUG#3848 - Using a MySQLDatasource without server name fails

    - Fixed BUG#3920 - "No Database Selected" when using
      MysqlConnectionPoolDataSource.
    - Fixed BUG#3873 - PreparedStatement.getGeneratedKeys() method returns only
      1 result for batched insertions

05-18-04 - Version 3.0.12-production
    - 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)

    - Allow java.util.Date to be sent in as parameter to
      PreparedStatement.setObject(), converting it to a Timestamp
      to maintain full precision (BUG#3103).

    - Don't truncate BLOBs/CLOBs when using setBytes() and/or
      setBinary/CharacterStream() (BUG#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.

    - Fixed BUG#3511, StringUtils.escapeSJISByteStream() not covering all
      eastern double-byte charsets correctly.

    - Renamed StringUtils.escapeSJISByteStream() to more appropriate
      escapeEasternUnicodeByteStream().

    - Fixed BUG#3554 - Not specifying database in URL caused MalformedURL
      exception.

    - 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).

    - Fixed BUG#3557 - UpdatableResultSet not picking up default values
      for moveToInsertRow().

    - Fixed BUG#3570 - inconsistent reporting of column type. The server
      still doesn't return all types for *BLOBs *TEXT correctly, so the
      driver won't return those correctly.

    - Fixed BUG#3520 -- DBMD.getSQLStateType() returns incorrect value.

    - Fixed regression in PreparedStatement.setString() and eastern character
      encodings.

    - Made StringRegressionTest 4.1-unicode aware.

02-19-04 - Version 3.0.11-stable

    - 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).

    - Fixed failover always going to last host in list (BUG#2578)

    - Added 'useUnbufferedInput' parameter, and now use it by default
      (due to JVM issue
      http://developer.java.sun.com/developer/bugParade/bugs/4401235.html)

    - Detect 'on/off' or '1','2','3' form of lower_case_table_names on
      server.

    - Return 'java.lang.Integer' for TINYINT and SMALLINT types from
      ResultSetMetaData.getColumnClassName() (fix for BUG#2852).

    - Return 'java.lang.Double' for FLOAT type from ResultSetMetaData.
      getColumnClassName() (fix for BUG#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.

01-13-04 - Version 3.0.10-stable

    - Don't count quoted id's when inside a 'string' in PreparedStatement
      parsing (fix for BUG#1511).

    - 'Friendlier' exception message for PacketTooLargeException
       (BUG#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).

    - Fixed BUG#1576, dealing with large blobs and split packets not being
      read correctly.

    - Fixed regression of Statement.getGeneratedKeys() and REPLACE statements.

    - Fixed BUG#1630, subsequent call to ResultSet.updateFoo() causes NPE if
      result set is not updatable.

    - Fix for 4.1.1-style auth with no password.

    - Fix for BUG#1731, Foreign Keys column sequence is not consistent in
```

```
  DatabaseMetaData.getImported/Exported/CrossReference().

- Fix for BUG#1775 - DatabaseMetaData.getSystemFunction() returning
  bad function 'VResultsSion'.

- Fix for BUG#1592 -- cross-database updatable result sets
  are not checked for updatability correctly.

- DatabaseMetaData.getColumns() should return Types.LONGVARCHAR for
  MySQL LONGTEXT type.

- ResultSet.getObject() on TINYINT and SMALLINT columns should return
  Java type 'Integer' (BUG#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.

- Fix for BUG#1933 -- Connection property "maxRows" not honored.

- Fix for BUG#1925 -- Statements being created too many times in
  DBMD.extractForeignKeyFromCreateTable().

- Fix for BUG#1914 -- Support escape sequence {fn convert ... }

- Fix for BUG#1958 -- ArrayIndexOutOfBounds when parameter number ==
  number of parameters + 1.

- Fix for BUG#2006 -- ResultSet.findColumn() should use first matching
  column name when there are duplicate column names in SELECT query
  (JDBC-compliance).

- 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 w/ 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.

10-07-03 - Version 3.0.9-stable

    - 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 (bugs.mysql.com BUG#496)

    - Fixed Statement and PreparedStatement issuing bogus queries
      when setMaxRows() had been used and a LIMIT clause was present
      in the query.

    - Fixed BUG#661 - refreshRow didn't work when primary key values
      contained values that needed to be escaped (they ended up being
      doubly-escaped).

    - Support InnoDB contraint names when extracting foreign key info
      in DatabaseMetaData BUG#517 and BUG#664
      (impl. ideas from Parwinder Sekhon)

    - Backported 4.1 protocol changes from 3.1 branch (server-side SQL
      states, new field info, larger client capability flags,
      connect-with-database, etc).

    - Fix UpdatableResultSet to return values for getXXX() when on
      insert row (BUG#675).

    - The insertRow in an UpdatableResultSet is now loaded with
      the default column values when moveToInsertRow() is called
      (BUG#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)

    - Fixed regression in large split-packet handling (BUG#848).

    - Better diagnostic error messages in exceptions for 'streaming'
      result sets.

    - Issue exception on ResultSet.getXXX() 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).

    - 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.

    - Fix for BUG#879, double-escaping of '\' when charset is SJIS or GBK and '\'
      appears in non-escaped input.

    - 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.

    - Fixed BUG#1099, DatabaseMetaData.getColumns() getting confused about the
      keyword 'set' in character columns.

    - Fixed deadlock issue with Statement.setMaxRows().

    - Fixed CLOB.truncate(), BUG#1130

    - Optimized CLOB.setChracterStream(), BUG#1131

    - Made databaseName, portNumber and serverName optional parameters
      for MysqlDataSourceFactory (BUG#1246)

    - Fix for BUG#1247 -- ResultSet.get/setString mashing char 127

    - Backported auth. 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.

05-23-03 - Version 3.0.8-stable

    - 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)

    - Return correct amount 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()

04-08-03 - Version 3.0.7-stable

    - 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 (i.e. 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.

02-18-03 - Version 3.0.6-stable

    - 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/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 (i.e. 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, as MySQL does
      not support 'locators'.

    - Backported 4.1 charset field info changes from Connector/J 3.1

01-22-03 - Version 3.0.5-gamma

    - 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.


01-06-03 - Version 3.0.4-gamma

    - 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 SELECTs 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.

12-17-02 - Version 3.0.3-dev

    - 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.

11-08-02 - Version 3.0.2-dev

    - 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 databases (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...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 only work when the connection is in a
      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 using Strings
      smarter in EscapeProcessor, Connection and DatabaseMetaData classes.

09-21-02 - Version 3.0.1-dev

    - 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() == false

    - Added "paranoid" parameter...sanitizes error messages removing
```

```
      "sensitive" information from them (i.e. hostnames, ports,
      usernames, etc.), 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)


07-31-02 - Version 3.0.0-dev

    - !!! 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 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


05-16-02 - Version 2.0.14
```

```
    - 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.

04-24-02 - Version 2.0.13

    - 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)

04-07-02 - Version 2.0.12

    - 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 timezone 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.

01-27-02 - Version 2.0.11

    - 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.

01-24-02 - Version 2.0.10

     - Fixed spurious "Unexpected end of input stream" errors in
       MysqlIO (bug 507456).

     - Fixed null-pointer-exceptions when using
       MysqlConnectionPoolDataSource with Websphere 4 (bug 505839).

01-13-02 - Version 2.0.9

     - 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).

11-25-01 - Version 2.0.8

     - 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).

10-24-01 - Version 2.0.7

     - 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 timezone 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 >= 3.21.23.

06-16-01 - Version 2.0.6

     - Fixed PreparedStatement parameter checking

     - Fixed case-sensitive column names in ResultSet.java

06-13-01 - Version 2.0.5

     - 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

12-03-00 - Version 2.0.3

     - Implemented getBigDecimal() without scale component
       for JDBC2.

     - Fixed composite key problem with updateable 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.

04-06-00 - Version 2.0.1

     - 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.

02-21-00 - Version 2.0pre5

     - Fixed Bad Handshake problem.

01-10-00 - Version 2.0pre4

     - 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.

08-17-99 - Version 2.0pre

      - 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)

07-04-99 - Version 1.2b

      - 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.

 04-14-99 - Version 1.2a

      - 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)

 03-24-99 - Version 1.1i

      - Fixed Timestamps for PreparedStatements

      - Fixed null pointer exceptions in RSMD and RS

      - Re-compiled with jikes for valid class files (thanks ms!)

 03-08-99 - Version 1.1h

      - Fixed escape processor to deal with un-matched { 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)

02-19-99 - Version 1.1g

     - 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 un-touched to the MySQL server.

     - Fixes to getDriverPropertyInfo().

12-31-98 - Version 1.1f

     - 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)

11-03-98 - Version 1.1b

     - 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.

9-02-98 - Vesion 1.1

     - 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, e.g. 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.

8-24-98 - Version 1.0

     - 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=... etc).


8-04-98 - Version 0.9d

     - 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, etc.
     - 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.

7-28-98 - Version 0.9

     - 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.

7-06-98 - Version 0.8

     - 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.

7-01-98 - Version 0.7

     - 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.

5-21-98 - Version 0.6

     - 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()
```