Mysql++
A C++ API for Mysql
ver 1.7.9

Kevin Atkinson
<kevinatk@home.com>
Sinisa Milivojevic
<sinisa@mysql.com>
Michael Widenius
<monty@mysql.com>

1st May 2001

Contents

Introductory Material

Chapter 1

Introduction

1.1 What is Mysql++

Mysql++ is a complex C++ API for Mysql (And other SQL Databases Soon). The goal of this API is too make working with Queries as easy as working with other STL Containers. See the Overview chapter (**??**) for the components that make up Mysql++.

1.2 Getting Mysql++

The latest version of Mysql++ can be found at the mysql++ web site at http://www.mysql.com/download_mysql++.html

1.3 Mysql++ Mailing List

Instructions for joining the mailing list (and an archive of the mailing list) can be found off the Mysql++ home page at http://www.mysql.com/download_mysql++.html. If you just wish to ask questions, you can mail to mysql-plusplus@lists.mysql.com.

Chapter 2

Overview

The Mysql++ API has developed into a very complex and powerful being. With many different ways to accomplish the same task. Unfortunately this means that figuring out how to perform a simple task can be frustrating for new users of my library. In this section we will attempt to provide an overview of the many different components of the library.

Like working with most other SQL API the process for executing queries is the same. 1) You open the connection, 2) You form and execute the queries, 3) You iterate through the result set. It not much different in my C++ API. However there is a lot of extra functionality along each step of the way.

The Main Database Handle

This is a class that handles the connection to the Mysql server. You always need at least one of these objects to do anything. It can either create a separate queries object or directly execute queries. The separate query object is the recommended way as it gives you far more power.

The Query Object

This object is the recommended way of executing queries. It is subclassed from strstream which means you can write to it like any other stream to aid in the formation of queries.

You can also set up Template queries with this class. Template queries are a way of setting up queries with replaceable parameters that you can change throughout your program.

You can also use specialized structures and even the dramatic result sets to aid in creating queries however more on that latter.

The Query object returns an object with information about the success of a query for non-select queries (queries that don't return a result set).

The Result Sets

For queries that return a result set you have essentially two different ways of handling the results: in a dramatic result set, or in a static one.

The Dramatic Result Set

The Dramatic Result set is a result set in which the names of the columns and the type of information of the columns does not need to be determined at compile time. The result set can be completely constant in which the data is returned to you in a constant string link class, semi-constant in which you can modify the data one row at a time, or a truly mutable in which in you can modify the data in any way you like.

The constant result set is a result set that is closely bound to the result set in the C API and is the one that provides the most functionality. With this result set you can find out detailed information about the type of information stored in each of the columns. This is also the fastest because the data does not need to be copied at all.

The semi-constant result set is like the constant result set except you can modify the data one row at a time. The data you modify is actually a copy of the data returned by the server. This means that modifying the data does not change the actual result set at all.

The semi-constant result set is actually the same thing as the constant result set. The only difference is that when you request a row from the result set you specifically declare the row as a mutable one. This means that you can get some rows back as constant rows and others as mutable ones.

The truly mutable result set is a result set similar to the constant one except that the data is truly mutable in the sense that you can change the data in the actual result set. However unlike the first one this result set is not bound to the C API result set. Instead it containers a copy of the data returned by the C API in a two-dimensional vector. Because of this the detailed information about each of the columns is not currently available, only the column names and the C++ type that most closely matches the original SQL type. Also, because it makes a copy of the data returned from the C API, there is a little bit of performance penalty to using this one.

The rows in all the dramatic result sets are very close to an Standard Template Library (STL) random access container. This means that they have an iterator which can be used for STL algorithms. There is even couple of specialized utility function to aid in the use of the result sets in STL algorithms.

The columns in all the dramatic result are also very close to an STL random access container. However, in addition to accessing the columns by there index number you can also access the columns via there field names.

In addition, because both the rows and the columns are STL like containers, you can also treat the result set as a two- dimensional array. For example you can get the 5th item on the 2nd row by simply saying result[2][5]. Because you can also use the field names you can substitute the column number by a field name and say result[2]["price"] to get "price" of the item on the 2nd row, for example.

The actual data that all the dramatic result sets return is stored in a special string like class that has some additional magic too it. The magic is that the column data will automatically convert itself into all of the basic data types as well as some additional types types that are designed to handle mysql types which includes types for handling dates, times, sets, and types with a null value. If there is a problem in the conversion it will either set a warning flag or throw an exception depending on how it is configured. Regarding exceptions, MySQL++ supports two different methods of tracing exceptions. One is by the fixed type (the old one) and one is standard C++ type by the usage of what() method. A choice of methods has to be done in building a library. If configure script is run with –enable-exception option , then new method will be used. If no option is provided, or –disable-exception is used, old MySQL++ exceptions will be enforced.

The drastic result sets can even be used to help form queries with the help of some additional method. There is a method for returns: 1) A comma separated list of the data (for example: 1.5, 10, "Dog, "Brown"), 2) A comma separated list of the field names (for example: age, weight, what, color), and 3) An equal list (for example: age = 1.5 AND weight = 10 AND what = "Dog" AND color = "Brown").

Mutable result sets can be created with out an actual query so that you can take advantage of these methods to aid in inserting data into the database with out having to first create an unnecessary query.

The Static Result Sets

The results from an query can also be stored statically in what we call a specialized SQL structure. These structures are then stored in some STL container such a vector or list, or even a set or multiset as the the specialized structures can also be made less-than-comparable. Unlike the dramatic result sets it is assumed that the programmer knows what the result set is going to look like. Because of this all the information about the columns, including the names, are lost.

These Specialized Structures are exactly that C++ 'structs'. Each member item is stored with a unique name within the structure. You can in no way use STL algorithms or anything else STL to work with the individual elements of the structures. However naturally because these structures are then stored in STL containers you can use STL algorithms on the containers of these structures. The containers represent the rows, and the individual elements of the structure represent the columns. For example you can access the item named "price" on the second row by saying result[2].price. With the dramatic result set you would have probably needed to say result[2]["price"] to accomplish the same result.

If there is a problem in converting from the result set returned by the server to the specialized structures an exception is thrown.

To aid in the creating of queries using these specialized structures, the same query aiding methods are available to use that are available for the dramatic result sets. This includes methods for returning a comma separated list of the data, a comma separated list of the field names, and an equal list.

The Dynamic Fully Mutable Sets

This result set will be implemented when server-side cursors are implemented in MySQL. But, based on so far acquired knowledge and experience, from designing and implementing both MySQL++ and MySQLGUI, a preliminiary layout and design of the most advanced result set so far has been achieved. It's implementation is postponed, however, from the above reasons. This result set will be fully dynamic and dramatic. This result set is fully dynamic in a sense that entire result set is stored in a dynamic C++ container. This container will be only a cache , a dynamic cache, to the entire result set, and will have a default size. This dynamic container will be a window of M rows into an entire result set of N rows, where N is limited by MySQL server , Operating System and File System only. This result set will also be fully dramatic in a sense that the names of the columns and the type of information of the columns will not need to be determined at compile time. But all existing functionality of static , mutable sets will be available in this set too. However as this set will be dramatic, no advance information on result set structure will be necessary, which will thus aleviate need for the usage of specialized macros for the construction of classes. This set will also have methods for updating, deleting and inserting rows in a manner that will be almost identical for use as methods for the existing fully mutable sets.

In addition

In addition to the material mentioned there are also many generic classes that can be used with other programs. Examples of this include a special const string class, a const random access adapter that will make a random access container out of a class with nothing but the size() method and the subscript ([]) operator defined and a generic SQL query class that can be used any SQL C or C++ API.

As from version 1.7, there is a new addtion to the libraries. Several very usefull functions for STL strings can be added, which can be used in any C++ aplication, MySQL++ related or not. Those functions are contained in source files string_util.hh and string_util.cc.

Chapter 3

Important Changes

3.1 Current Changes

- Changed the syntax of equal_list for SSQLS from **equal_list (cchar *, Manip, cchar *)** to **equal_list (cchar *, cchar *, Manip)**.
- Since version 1.3 of mysql++, it can no longer be compiled and built by GNU compilers older then 2.95. Since version 1.3, mysql++ has been changed to accomodate changes in 2.95 on various aspects of C++. This prevents it's building with earlier versions of GNU , like 2.7.xx, 2.8.xx. Also building by egcs compilers 1.x.x is not supported any more. Although with some changes in code mysql++ could be built with earlier compilers, running of such programs would result in their crashing.
- At the writting of the present version (1.7.9) g++ 2.95.3 and g++ 2.96 can not be used either
- There are separate versions for Borland C++, VC++ and Compaq compilers on Tru64
- It was also noted that on certain SPARC Solaris installation, C++ exceptions did not work with gcc 2.95.2. This case was tested and it is established that mysql++ builds and runs flawlessly with a following version of gcc on Solaris: gcc version 2.95 19990728 (release)
- The "connection" constructors for **Connection** and **Connection::connect** (formally known as **Mysql**) has changed so that the data base name is the first paremeter. The same is valid for connect method.
- All new client feature implemented in 3.22.xx as various options on connect are mplemented in a new constructor and real_connect method
- Also new configuration constructs in 3.23.xx are strictly followed
- Mysql++ now can be compiled on Win32 with use of Cygwin compiler from Cygnus Inc.
- Autoconf and Automake are fully implemented
- The Specialized SQL Structures (formally known as Custom Mysql Structures) changed from mysql_ to sql_.

- Changed all of the functions that return zero (false) on success and non-zero (true) other wise to bool. This means that they now return true on success and false on faillier. This means that you now need to negate your expressions that test the output of these functions.
- Almost all methods that returned MysqlString now return a normal string. Because **Mysql-String** objects will convert them selves into string when needed there should not be a problem with:

  ```
  MysqlString s = mysql.host_info
  ```

- The data type **MysqlRow** now returns (**MysqlString** or **MysqlColData**) is now subclassed from a special string class that we wrote to handle working with a const string in a much more efficient way (ie it doesn't copy it). Unfortunitlly this data type is also more limited see the section on "const_string" for more info.
- All necessary methods pertaining to the administration funcitons have been added
- Mysql++ now quotes and escapes objects automatically, if column data is used with '<<' operator
- Mysql++ now has much more secure execution of INSERT, UPDATE and DELETE with a new exec() method
- Mysql++ now has much better configuration
- There is a new method of fetching strings
- Mysql++ can now work with binary data
- Standard C++ exceptions handling with what() method has been introduced
- All 64 int handling of string conversions have been moved to libmysqlclient
- Programs written with MySQL++ will now automatically read all relevant MySQL configuration files

3.2 Future Changes

- To implement fully mutable result sets
- The behavior of MysqlString when used with binary operators is going to change in a future version (if we can figure out how to pull it off) Instead of converting to the type on the other side of the operator the MysqlString will it convert to the type the Mysql server said it originally was. This will be a lot safer and more predictable.

Usage

Chapter 4

Tutorial by Example

4.1 Introduction

This tutorial is meant to give you a jump start into using my API. My Mysql++ API is a very complicated being with a lot of advance features that you can due without if all you want to do is execute simple queries.

4.2 Assumptions

This tutorial assumes you know C++ fairly well, inparticuler it assumes you know about the Standard Template Library (STL) and exceptions.

4.3 Running the Examples

All of the example code form complete running programs. However in order to use them you need to first compile them my switching to the examples directory and typing in `make`. Then you need to set up the database by running reset-db. The usage of the reset-db program is as follows.

```
reset-db [host [user [password]]]
```

If you leave off host localhost is assumed. If you leave off user your current username is assumed. If you leave of the password it is assumed that you don't need one.

When you first run the program you need to give it an account with permission to create databases. Once the database is created you can use any account that has permission full permission to the database mysql_cpp_data.

You should also run the reset-db program between examples that modify the data or else things might not work right.

4.4 The Basics

4.4.1 A Simple Example

The following example demonstrates how to open a connection, execute a simple query, and display the results. The code can be found in the file `simple1.cc` which is located in the examples directory.

```
#include <iostream>
#include <iomanip>
#include <sqlplus.hh>

int main() {
Connection con("mysql_cpp_data");
// The full format for the Connection constructor is
// Connection(cchar *db, cchar *host="",
//            cchar *user="", cchar *passwd="")
// You may need to specify some of them if the database is not on
// the local machine or you database username is not the same as your
// login name, etc..

Query query = con.query();
// This creates a query object that is bound to con.

query << "select * from stock";
// You can write to the query object like you would any other ostrem

Result res = query.store();
// Query::store() executes the query and returns the results

cout << "Query: " << query.preview() << endl;
// Query::preview() simply returns a string with the current query
// string in it.

cout << "Records Found: " << res.size() << endl << endl;

Row row;
cout.setf(ios::left);
cout << setw(17) << "Item"
<< setw(4)  << "Num"
<< setw(7)  << "Weight"
<< setw(7)  << "Price"
<< "Date" << endl
<< endl;

Result::iterator i;
// The Result class has a read-only Random Access Iterator
for (i = res.begin(); i != res.end(); i++) {
row = *i;
cout << setw(17) << row[0]
```

```
<< setw(4) << row[1]
<< setw(7) << row["weight"]
// you can use either the index number or column name when
// retrieving the colume data as demonstrated above.
<< setw(7) << row[3]
<< row[4] << endl;
}
return 0;
}
```

Everything here should be fairly obvious. Take particular notice of how we used an iterator with the result set.

4.4.2 A slightly more complicated example

This example is almost like the previous one however it uses exceptions and the automatic conversion feature of **ColData**. Pay particular notice to how exceptions are used. This file for this code is named `complic1.cc`.

```
#include <iostream>
#include <iomanip>
#include <sqlplus.hh>

int main() {
try { // its in one big try block

Connection con(use_exceptions);
con.connect("mysql_cpp_data");
// Here we broke making the connection into two calls.
// The first one creates the Connection object with the
// use exceptions option turned on and the second one
// makes the connection

Query query = con.query();

query << "select * from stock";
Result res = query.store();

cout << "Query: " << query.preview() << endl;
cout << "Records Found: " << res.size() << endl << endl;

Row row;
cout.setf(ios::left);
cout << setw(17) << "Item"
<< setw(4) << "Num"
<< setw(7) << "Weight"
<< setw(7) << "Price"
<< "Date" << endl
<< endl;
```

```
Result::iterator i;

cout.precision(3);
for (i = res.begin(); i != res.end(); i++) {
row = *i;
cout << setw(17) << row["item"] << setw(4) << row[1]
<< setw(7)  << (double)row[2]
// This is converting the row to a double so that we
// can set the precision of it.
// ColData has the nice feature that it will convert to
// any of the basic c++ types.  if there is a problem
// in the conversion it will throw an exception (which we
// cache below).  To test it try changing the 2 in row[2]
// to row[0]
<< setw(7) << (double)row[3];
Date date = row["sdate"];
// The ColData is implicitly converted to a date here.
cout.setf(ios::right);
cout.fill('0');
cout << setw(2) << date.month << "-" << setw(2) << date.day << endl;
cout.fill(' ');
cout.unsetf(ios::right);
}
return 0;
} catch (BadQuery er) { // handle any connection or
// query errors that may come up
cerr << "Error: " << er.error << endl;
return -1;
} catch (BadConversion er) { // handle bad conversions
cerr << "Error: Tried to convert \"" << er.data << "\" to a \""
<< er.type_name << "\"." << endl;
return -1;
}
}
```

Everything should be fairly obvious. A few notes about exceptions, however:

1. When the **use_exceptions** flag is set for a parent object it is also set for all of its children the it created after the flag is set. For example when the **use_exceptions** flag is set for the `con` object, it is also set for the `query` object. Please note that the **use_exceptions** flag is not linked, it is copied. This means that when you change the **use_exceptions** flag only its new children are affected, *not* the ones it already created.

2. **ColData** will always throw an exception when it encounters a bad conversion. A bad conversion is defined as a conversion in which: a) All the charters from the string are not read in and b) The remaining characters are something other than whitespace, zeros (0), or periods (.). This means that when "1.25" is converted into an int an exception will be thrown however

not when “1.00” is converted into an int as the remaining characters are the period and the zero.

To see how the exception work try creating an error. Some good things to try would be misspelling the table name or changing the double to an int.

4.4.3 Getting Info about the Fields

The following example demonstrates how to get some basic information about the fields, including the name of the field and the SQL type. The file is called fieldinfo1.cc.

```
#include <iostream>
#include <iomanip>
#include <sqlplus.hh>

int main() {
try { // its in one big try block

Connection con(use_exceptions);
con.connect("mysql_cpp_data");
Query query = con.query();
query << "select * from stock";
Result res = query.store();

cout << "Query: " << query.preview() << endl;
cout << "Records Found: " << res.size() << endl << endl;

cout << "Query Info:\n";
cout.setf(ios::left);

for (unsigned int i = 0; i < res.size(); i++) {
cout << setw(2) << i
<< setw(15) << res.names(i).c_str()
// this is the name of the field
<< setw(15) << res.types(i).sql_name()
// this is the SQL identifier name
// Result::types(unsigned int) returns a mysql_type_info which in many
// ways is like type_info except that it has additional sql type
// information in it. (with one of the methods being sql_name())
<< setw(20) << res.types(i).name()
// this is the C++ identifier name which most closely resembles
// the sql name (its is implementation defined and often not very readable)
<< endl;
}

cout << endl;

if (res.types(0) == typeid(string))
```

```
cout << "Field 'item' is of an sql type which most closely resembles a\n"
<< "the c++ string type\n";
// this is demonstrating how a mysql_type_info can be compared with a c++
// type_info.

if (res.types(1) == typeid(short int))
cout << "Field 'num' is of an sql type which most closely resembles a\n"
<< "the c++ short int type\n";
else if (res.types(1).base_type() == typeid(short int))
cout << "Field 'num' base type is of an sql type which most closely \n"
<< "resembles a the c++ short int type\n";
// However you have to be careful as if it can be null the actual type is
// Null<TYPE> not TYPE.  So you should always use the base_type method
// to get at the underlying type.  If the type is not null than this base
// type would be the same as its type.

return 0;
} catch (BadQuery er) {
cerr << "Error: " << er.error << endl;
return -1;
} catch (BadConversion er) { // handle bad conversions
cerr << "Error: Tried to convert \"" << er.data << "\" to a \""
<< er.type_name << "\"." << endl;
return -1;
}
}
```

4.5 Specialized SQL Structures

4.5.1 Retrieving Data

The next example demonstrates a fairly interesting concept known as Specialized SQL Structures (SSQLS). The file name for this code is `custom1.cc`.

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <sqlplus.hh>
#include <custom.hh>

sql_create_5 (stock,          // struct name,
1, 5,            // I'll explain these latter
string, item,     // type, id
int, num,
double, weight,
double, price,
Date, sdate)
```

```
// this is calling a very complex macro which will create a custom
// struct "stock" which has the variables:
//   string item
//    int num
//    ...
//    Date sdate
// defined as well methods to help populate the class from a mysql row
// among other things that I'll get too in a latter example

int main () {
try {                          // its in one big try block
Connection con (use_exceptions);
con.connect ("mysql_cpp_data");
Query query = con.query ();
query << "select * from stock";

vector < stock > res;
query.storein (res);
// this is storing the results into a vector of the custom struct
// "stock" which was created my the macro above.

cout.setf (ios::left);
cout << setw (17) << "Item"
<< setw (4) << "Num"
<< setw (7) << "Weight"
<< setw (7) << "Price"
<< "Date" << endl
<< endl;

// Now we we iterate through the vector using an iterator and
// produce output similar to that using Row
// Notice how we call the actual variables in i and not an index
// offset.  This is because the macro at the begging of the file
// set up an *actual* struct of type stock which contains the
// variables item, num, weight, price, and data.

cout.precision(3);
vector <stock>::iterator i;
for (i = res.begin (); i != res.end (); i++) {
cout << setw (17) << i->item.c_str ()
// unfortunally the gnu string class does not respond to format
// modifers so we have to convert it to a conat char *.
<< setw (4) << i->num
<< setw (7) << i->weight
<< setw (7) << i->price
<< i->sdate
<< endl;
```

```
}
return 0;

} catch (BadQuery er){ // handle any connection
// or query errors that may come up
cerr << "Error: " << er.error << endl;
return -1;

} catch (BadConversion er) {
// we still need to cache bad conversions incase something goes
// wrong when the data is converted into stock
cerr << "Error: Tried to convert \"" << er.data << "\" to a \""
<< er.type_name << "\"." << endl;
return -1;
}
}
```

As you can see. SSQLS are very powerful things.

4.5.2 Adding Data

SSQLS can also be used to add data to a table. The file name for this code is custom2.cc

```
#include <iostream>
#include <vector>
#include <sqlplus.hh>
#include <custom.hh>
#include "util.hh"
// util.hh/cc contains the print_stock_table function

sql_create_5(stock, 1, 5, string, item, int, num,
double, weight, double, price, Date, sdate)

int main() {
try { // its in one big try block

Connection con(use_exceptions);
con.connect("mysql_cpp_data");
Query query = con.query();

stock row;
// create an empty stock object

/*    row.item = "Hot Dogs";
row.num = 100;
row.weight = 1.5;
row.price = 1.75;
row.sdate = "1998-09-25"; */
row.set("Hot Dogs", 100, 1.5, 1.75, "1998-09-25");
```

```
// populate stock

query.insert(row);
// form the query to insert the row
// the table name is the name of the struct by default
cout << "Query : " << query.preview() << endl;
// show the query about to be executed
query.execute();
// execute a query that does not return a result set

print_stock_table(query);
// now print the new table;

} catch (BadQuery er) {
cerr << "Error: " << er.error << endl;
return -1;
} catch (BadConversion er) {
cerr << "Error: Tried to convert \"" << er.data << "\" to a \""
<< er.type_name << "\"." << endl;
return -1;
}
}
```

That's all there is to it. Because this example modifies the data you should run `reset-db` after running the example code.

4.5.3 Modifying Data

And it almost as easy to modify data with SSQLS. The file name is custom3.cc.

```
#include <iostream>
#include <vector>
#include <sqlplus.hh>
#include <custom.hh>
#include "util.hh"
// util.hh/cc contains the print_stock_table function

sql_create_5(stock, 1, 5, string, item, int, num,
double, weight, double, price, Date, sdate)

int main() {
try { // its in one big try block

Connection con(use_exceptions);
con.connect("mysql_cpp_data");
Query query = con.query();

query << "select * from stock where item = \"Hotdogs' Buns\" ";
```

```
Result res = query.store();
if (res.empty())
throw BadQuery("Hotdogs' Buns not found in table, run reset-db");
// here we are testing if the query was successful, if not throw a bad query
stock row = res[0];
// because there should only be one row in this query we don't
// need to use a vector.  Just store the first row directly in
// "row".  We can do this because one of the constructors for
// stock takes a Row as an parameter.

stock row2 = row;
// Now we need to create a copy so that the replace query knows
// what the original values are.

row.item = "Hotdog Buns"; // now change item

query.update(row2, row);
// form the query to replace the row
// the table name is the name of the struct by default
cout << "Query : " << query.preview() << endl;
// show the query about to be executed
query.execute();
// execute a query that does not return a result set

print_stock_table(query);
// now print the new table;

} catch (BadQuery er) {
cerr << "Error: " << er.error << endl;
return -1;
} catch (BadConversion er) {
cerr << "Error: Tried to convert \"" << er.data << "\" to a \""
<< er.type_name << "\"." << endl;
return -1;
}
}
```

When you run the example you will notice that in the where clause only the *item* field is checked for. This is because SSQLS also also less-than-comparable.

Don't forget to run `reset-db` after running the example.

4.5.4 Less-Than-Comparable

SSQLS are can also be made less-than-comparable. This means that they can be sorted and stored in sets as demonstrated in the next example. The file name is custom4.cc

```
#include <iostream>
#include <iomanip>
```

```
#include <vector>
#include <sqlplus.hh>
#include <custom.hh>

sql_create_5(stock,
1, // This number is used to make a SSQLS less-than-comparable.
// If this number is n then if the first n elements are the
// same the two SSQLS are the same.
// In this case if two two stock's "item" are the same then
// the two stock are the same.
5, // this number should generally be the same as the number of
// elements in the list unless you have a good reason not to.

string,item, int,num, double,weight, double,price, Date,sdate)

int main() {
try { // its in one big try block

Connection con(use_exceptions);
con.connect("mysql_cpp_data");
Query query = con.query();

query << "select * from stock";

set<stock> res;
query.storein(res);
// here we are storing the elements in a set not a vector.

cout.setf (ios::left);
cout << setw (17) << "Item"
<< setw (4) << "Num"
<< setw (7) << "Weight"
<< setw (7) << "Price"
<< "Date" << endl
<< endl;

// Now we we iterate through the set. Since it is a set the list will
// naturally be in order.

set<stock>::iterator i;
cout.precision(3);
for (i = res.begin (); i != res.end (); i++) {
cout << setw (17) << i->item.c_str ()
<< setw (4) << i->num
<< setw (7) << i->weight
<< setw (7) << i->price
<< i->sdate
<< endl;
```

```
}

i = res.find(stock("Hamburger Buns"));
if (i != res.end())
cout << "Hamburger Buns found.  Currently " << i->num << " in stock.\n";
else
cout << "Sorry no Hamburger Buns found in stock\n";

// Now we are using the set's find method to find out how many
// Hamburger Buns are in stock.

return 0;

} catch (BadQuery er) {
cerr << "Error: " << er.error << endl;
return -1;
} catch (BadConversion er) {
cerr << "Error: Tried to convert \"" << er.data << "\" to a \""
<< er.type_name << "\"." << endl;
return -1;
}
}
```

4.6 Let us be usefull

Beginning with MySQl++ 1.6 we have introduced three new examples, whose aim is to demonstrate some of the strongest features of MySQL++, whose primary objective is not just to demonstrate power and ease of use of MySQL++, but also to provide a solution to some of the most frequent problems presented by MySQL users. These examples exemplify a superiority of C++ over other existing languages. Those examples take very few effective MySQL++ / C++ commands to produce highly efficient code, such that each of those examples resovles some very common problems that face MySQL users, especially beginners.

As these examples are meant to tbe applied, and are applied by many MySQL users, constants that can differ from one case to another have been grouped in order to simplify editing. Also , all of this examples contain full error checking code. This is one of the areaa where C++ exception handling, fully applied in MySQL++, truly shines.

4.6.1 Loading binary file in a BLOB column

This function is solved in MySQL version 3.23 , but as of this writing many users are still using older versions. Beside that this examples demonstrates several features of MySQL++. This program requires one argument, which is a full path of the binary file.

```
#include <sys/stat.h>
#include <fstream>
#include <mysql++>
extern int errno;
```

```
const char MY_DATABASE[]="telcent";
const char MY_TABLE[]="fax";
const char MY_HOST[]="localhost";
const char MY_USER[]="root";
const char MY_PASSWORD[]="";
const char MY_FIELD[]="fax"; // BLOB field
int main(int argc, char *argv[]) {
if (argc < 2) {
cerr << "Usage : load_file full_file_path" << endl << endl;
return -1;
}
Connection con(use_exceptions);
try {
con.real_connect (MY_DATABASE,MY_HOST,MY_USER,MY_PASSWORD,3306,(int)0,60,NULL);
Query query = con.query(); ostrstream strbuf;
ifstream In (argv[1],ios::in | ios::binary); struct stat for_len;
if ((In.rdbuf())->is_open()) {
if (stat (argv[1],&for_len) == -1) return -1;
unsigned int blen = for_len.st_size; if (!blen) return -1;
char *read_buffer = new char[blen]; In.read(read_buffer,blen); string fill(read_buffer,blen);
strbuf << "INSERT INTO " << MY_TABLE << " (" << MY_FIELD << ") VALUES(\"" <<
escape << fill << "\")";
query.exec(strbuf.str());
delete[] read_buffer;
}
else
cerr << "Your binary file " << argv[1] << "could not be open, errno = " << errno;
return 0;
} catch (BadQuery er) {
cerr << "Error: " << er.error << " " << con.errnum() << endl;
return -1;
}
}
```

One of the features that is displayed in this example is escape manipulator. Although automatic quoting and escaping is introduced in version 1.6, it is applicable to ColData classes only, as they contain info on data type. We could also make quoting and escaping on general data type string, but it would require to scan entire string to find out if quoting and escaping is applicable. As this feature would slow down code, we DEFINITELY NEED USER'S FEEDBACK on this matter.

4.6.1 Displaying images in HTML from BLOB column

This example is also very short one, considering a function that it performs. Although since 3.23.3, there is a command that dumps data from BLOB column in a binary file, this program can be used not only by users still utilizing older versions, but by users that do not wish to have this middle step of saving image to disk.

```
#include <sqlplus.hh>
```

```
#define MY_DATABASE "telcent"
#define MY_TABLE "fax"
#define MY_HOST "localhost"
#define MY_USER "root"
#define MY_PASSWORD ""
#define MY_FIELD "fax" // BLOB field
#define MY_KEY "datet" // PRIMARY KEY
int main (int argc, char *argv[]) {
if (argc < 2) {
cerr << "Usage : cgi_image primary_key_value" << endl << endl;
return -1;
}
cout << "Content-type: image/jpeg" << endl;
Connection con(use_exceptions);
try {
con.real_connect (MY_DATABASE,MY_HOST,MY_USER,MY_PASSWORD,3306,(int)0,60,NULL);
Query query = con.query();
query << "SELECT " << MY_FIELD << " FROM " << MY_TABLE << " WHERE " <<
MY_KEY << " = " << argv[1];
ResUse res = query.use(); Row row=res.fetch_row(); long unsigned int *jj = res.fetch_lengths();
cout << "Content-length: " << *jj << endl << endl;
fwrite(row.raw_data(0),1,*jj,stdout); return 0;
} catch (BadQuery er) {
cerr << "Error: " << er.error << " " << con.errnum() << endl;
return -1;
}
}
```

This example demonstrates MySQL++ handling of binary data, which was introduced in version 1.6. Flexible usage of streams enables utilization of this program in many applications.

4.6.2 Delete or Update from Select

This feature is asked for by many users, but until it is done, this program can be used instead. it is a small program, which also demonstrates few MySQL++ features.

```
#include <sqlplus.hh>
#define MY_DATABASE "telcent"
#define MY_TABLE "nazivi"
#define MY_HOST "localhost"
#define MY_USER "root"
#define MY_PASSWORD ""
#define MY_FIELD "naziv"
#define MY_QUERY "SELECT URL from my_table as t1, my_table as t2 where t1.field =
t2.field"
int main (void) {
Connection con(use_exceptions);
try {
```

```
ostrstream strbuf; unsigned int i=0;
con.real_connect (MY_DATABASE,MY_HOST,MY_USER,MY_PASSWORD,3306,(int)0,60,NULL);
Query query = con.query(); query << MY_QUERY;
ResUse res = query.use(); Row row;
strbuf << "delete from " << MY_TABLE << " where " << MY_FIELD << " in (";
// for UPDATE just replace the above DELETE FROM with UPDATE statement
for(;row=res.fetch_row();i++) strbuf << row[0] << ","; if (!i) return 0;
string output(strbuf.str()); output.erase(output.size()-1,1); output += ")";
query.exec((const string&)output); // cout << output << endl;
return 0;
} catch (BadQuery er) {
cerr << "Error: " << er.error << " " << con.errnum() << endl;
return -1;
}
}
```

Please do notify that in query construction command field MY_FIELD list of values is inserted unquoted and unescaped. This way a new feature, introduced in MySQL++ since version 1.6, is exemplified. Namely field values will be quoted or not depending on it's type. Users should not explicitely quote or quote + escape, as this will result in error. This way some more burden is taken from a programmer. Programmer may disable this feature by setting a corresponding global veriable to false. **??**This example is written to perform DELETE. UPDATE requires some changes.

All users of this examples should beware that one more check is required in order to run this query safely. Namely, in some extreme cases, size of query might grow larger then max_allowed packet. Thterefore this check should be added.

4.6.2 And There's More

This is only scratching the surface of what SSQLS can do for more information see the chapter on them (**??**).

4.7 Template Queries

Another powerful feature of Mysql++ is being able to set up template queries. The following example demonstrates how to use them. This code is the actual code used to set up and/or reset the sample database. It can be found under reset-db.cc. we hope to come up with some better examples soon.

```
#include <iostream>
#include <sqlplus.hh>

int main (int argc, char *argv[]) {
Connection connection(use_exceptions);
try { // the entire main block is one big try block;

if (argc == 1) connection.connect("");
```

```
else if (argc == 2) connection.connect("",argv[1]);
else if (argc == 3) connection.connect("",argv[1],argv[2]);
else if (argc <= 4) connection.connect("",argv[1],argv[2],argv[3]);
// create a new object and connect based on any (if any) arguments
// passed to main();

try {
connection.select_db("mysql_cpp_data");
} catch (BadQuery er) {
// if it couldn't connect to the database assume that it doesn't exist
// and try created it.  If that does not work exit with an error.
connection.create_db("mysql_cpp_data");
connection.select_db("mysql_cpp_data");
}

Query query = connection.query();  // create a new query object

try { // ignore any errors here
// we hope to make this simpler soon
query.execute("drop table stock");
} catch (BadQuery er) {}

query << "create table stock  (item char(20) not null, num smallint,"
<< "weight double, price double, sdate date)";
query.execute(RESET_QUERY);
// send the query to create the table and execute it.  The
// RESET_QUERY tells the query object to reset it self after
// execution

query << "insert into %5:table values (%q0, %q1, %2, %3, %q4)";
query.parse();
// set up the template query we will use to insert the data.  The
// parse method call is important as it is what lets the query
// know that this is a template and not a literal string

query.def["table"] = "stock";
// This is setting the parameter named table to stock.

query.execute ("Hamburger Buns", 56, 1.25, 1.1, "1998-04-26");
query.execute ("Hotdogs' Buns"   ,65, 1.1 , 1.1, "1998-04-23");
query.execute ("Dinner Roles"  , 75,  .95, .97, "1998-05-25");
query.execute ("White Bread"   , 87, 1.5, 1.75, "1998-09-04");
// The last parameter "table" is not specified here.  Thus
// the default value for "table" is used which is "stock".

} catch (BadQuery er) { // handle any errors that may come up
cerr << "Error: " << er.error << endl;
return -1;
```

```
}
}
```

Chapter 5

Class Reference

This chapter documents all of the classes that are meant to be used. If it is not documented here **don't use it** because it is a internal method or class meant to only be used by other related classes.

5.1 Manipulators

The following manipulators modify only the next item to the right of it in an `<<` chain. They can be used with any ostream (which includes **SQLQuery** and **Query** because they are also ostreams) or **SQLQueryParms**. When used with **SQLQueryParms** they will override any settings set by the Template Query for that particular item.

quote Quote and escape the next item. Can be used with **ostream** or **SQLQueryParms**.

quote_only Quote but don't escape the next item. Can be used with **ostream** or **SQLQueryParms**.

quote_only_double Quote, but don't escape the next item, with " instead of '.

escape Escape the next item.

do_nothing Does exactly what it says nothing. Used as a dummy manipulator when you are required to use some manipulator. When used with **SQLQueryParms** it will make sure that it does not get formatted in any way overriding any setting set by the template query.

ignore Only valid when used with **SQLQueryParms**. Like **do_nothing** however this one will not override formatting set by the template query, thus it is ignored.

Since version 1.6, automatic quoting and escaping has been added to manipulators. This mechanism is applied to mysql_ColData only, iso epse to the class very frequentrly uitlized as a return object of Row[] index. Automatic quoting or escaping is used with $<<$ operator only, and on all stream derived classes and objects, including strstream, query objects, but excepting cout, cerr and clog. This has been designed so intentionally, as streaming out values to those objects does not require quoting or escaping. But this feature comes handy when you construct query's by streaming values to query object or to strstream class object.

This feature can be glibally turned of by setting value *dont_quote_auto* in your code to true.

Chapter 6

Template Queries

The idea of template queries is too provide a query with replaceable parameters that can be changed between query calls with out having to reform the queries.

6.1 Setting Them Up

To set up a template query simply enter the query like it is a normal query. For example:

```
query << "select (%2:field1, %3:field2) from stock where %1:wheref = %q0:what"
```

And then execute the Query::parse() method. For example:

```
query.parse()
```

6.2 Template Format

An example template looks like this

```
select (%2:field1, %3:field2) from stock where %1:wheref = %q0:what
```

The numbers represent the element number in **SQLQueryParms** (see the next section).

The format of the substation parameter is:

```
%(modifier)##(:name)(:)
```

Where Modifier can be any one of the following:

% Print an actual "%"

"" "" means nothing. Don't quote or escape no matter what.

q This will quote and escape it using mysql_escape_string if it is a string or char *, or another Mysql specific types that needs to be quoted.

Q Quote but don't escape based on the same rules. This can save a bit of time if you know the strings will never need quoting

r Always quote and escape even if it is a number.

R Always quote but don't escape even if it is a number.

represents a number up to two digits

":name" is for an optional name which aids in filling SQLQueryParms. Name can contain any alpha-numeric characters or the underscore. If you use name it must be proceeded by non-alpha-numeric charter. If this is not the case add a column after the name. If you need to represent an actual colon after the name follow the name by two-columns. The first one will end the name and the second one won't be processed.

6.3 Setting the Parameters

The parameters can either be set when the query is executed or ahead of time by using default parameters.

6.3.1 At Execution Time

To specify the parameters when you want to execute a query simply use **Query::store(const SQLString &parm0, [..., const SQLString &parm11])** (or **Query::use** or **Query::execute**). Where **parm0** corresponds to parameter number 0, etc. You may specify from 1 to 12 different parameters. For example:

```
Result res = query.store("Dinner Roles", "item", "item", "price")
```

with the template query provided in section **??** would produce:

```
select (item, price) from stock where item = "Dinner Roles"
```

The reason for why *we didn't* make the template the more logical:

```
select (%0:field1, %1:field2) from stock where %2:wheref = %q3:what
```

will become apparent shortly.

6.3.2 Using Defaults

You can also set the parameters one at a time by means of the public data member **def**. To change the values of the **def** simply use the subscript operator. You can refer to the parameters either by number or by name. For example:

```
query.def[0] = "Dinner Roles";
query.def[1] = "item";
query.def[2] = "item";
query.def[3] = "price";
```

and

```
query.def["what"] = "Dinner Roles";
query.def["wheref"] = "item";
query.def["field1"] = "item";
query.def["field2"] = "price";
```

would both have the same effect.

Once all the parameters are set simply execute as you would have executed the query before you knew about template queries. For example:

```
Result res = query.store()
```

6.3.3 Combining the Two

You can also combine the use of setting the parameters at execution time and setting them by use of the **def** object by simply using the extended form of **Query::store** (or **use** or **execute**) without all of necessary parameters specified. For example:

```
query.def["field1"] = "item";
query.def["field2"] = "price";
Result res1 = query.store("Hamburger Buns", "item");
Result res2 = query.store(1.25, "price");
```

Would store the query:

```
select (item, price) from stock where item = "Hamburger Buns"
```

for `res1` and

```
select (item, price) from stock where price = 1.25
```

for `res2`.

Because the extended form of **Query::store** can only effect the beginning (by number not by location) parameters the more logical template query:

```
select (%0:field1, %1:field2) from stock where %2:wheref = %q3:what
```

would *not* of worked in this case. Thus the more twisted ordering of

```
select (%2:field1, %3:field2) from stock where %1:wheref = %q0:what
```

was needed so that we can specify **wheref** and **what** each time.

One thing to watch out for, however, is that **Query::store(const char* q)** is also defined for executing the query `q`. For this reason when you use the **Query::store** (or **use**, or **execute**) with only one item and that item is a **const char*** you need to explicitly convert it into a SQLString. For example:

```
Result res = query.store(SQLString("Hamburger Buns")).
```

6.3.4 Error Handling

If for some reason you did not specify all the parameters when executing the query *and* the remaining parameters do not have there values set via `def` the query object will throw a **SQLQueryNEParms** object. In which case you you can find out what happened by checking the value of **SQLQueryNEParms::string**.

For example:

```
query.def["field1"] = "item";
query.def["field2"] = "price";
Result res = query.store(1.25);
```

would throw **SQLQueryNEParms** because the `wheref` is not specified.

In theory this exception should never be thrown. If the exception is thrown it probably a logic error on you part. (Like in the above example)

6.3.5 More Advanced Stuff

To be written. However, for now see the class **SQLQuery** (**??**) and **SQLQueryParms** (**??**) for more information.

Chapter 7

Specialized SQL Structures

The Specialized SQL Structures (SSQLS) allows you create structures to hold data for mysql queries with extra functionality to make your life easier. These structures are in no way related to any Standard Template Library (STL) type of containers. These structures are exactly that **structs**. Each member item is stored with a unique name within the structure. You can in no way use STL algorithms are anything else STL to work with the individual structures. However you CAN use these structures as the **value_type** for STL containers. (They would be pretty useless if you couldn't.)

7.1 sql_create_basic

The following command will create a basic mysql query for use with the sample database.

```
sql_create_basic_5(stock, 0, 0,
                   string, item, // type, id,
                   int, num,
                   double, weight,
                   double, price,
                   MysqlDate, date)
```

This will set up the following structure:

```
struct stock {
  stock () {}
  stock (const MysqlRow &row);
  set (const MysqlRow &row);

  string item;
  int num;
  double weight;
  double price;
  MysqlDate date;
};
```

As you can see this is nothing fancy. The main advantage of this simple structure is the **stock (MysqlRow &row)** constructor which allows you to easily populate a vector of stocks like so:

```
vector<stock> result;
query.storein(result);
```

That's all there is two it. The requirements are that the query returns elements in the same order as you specified them in the custom structure.

The general format is:

```
sql_create_basic_#(NAME, 0, 0, TYPE1, ITEM1, ... TYPE#, ITEM#)
```

Where # is the number of valuables in the vector, NAME is the name of the structure you wish to create, and TYPE1 is the type name for first item and ITEM1 is the valuables name for the first item etc..

7.2 sql_create_basic with compare

You can also make the structure comparable by changing the first 0 in the previous example to a non zero number. This number, lets call it n, will tell c++ that if the first n number or the same then the two structures are the same.

For example:

```
sql_create_basic_5(stock, 1, 0,
                   string, item, // type, id,
                   int, num,
                   double, weight,
                   double, price,
                   MysqlDate, date)
```

will create a structure where only the item valuable is checked to see if two different stocks are the same. It also allows you to compare one structure to another based on the value of item. (If n is greater than one it will compare the structures in a Lexicographic order. For example if it was 2 it would first compare `item` and if item was the same it would then compare `num`. If num was the same it would declare the two structures the same.)

In addition what the previous example defines it also defines the following:

```
struct stock
  ...
  stock (const string &p1);
  set (const string &p1);
  bool operator == (const stock &other) const;
  bool operator != (const stock &other) const;
  bool operator > (const stock &other) const;
```

```
    bool operator < (const stock &other) const;
    bool operator >= (const stock &other) const;
    bool operator <= (const stock &other) const;
    int cmp (const stock &other) const;
    int compare (const stock &other) const;
  }

  int compare (const stock &x, const stock &y);
```

int compare (const stock &x, const stock &y) compares x to y and return <0 if $x < y$, 0 if $x = y$, and >0 if $x > y$. stock::cmp and stock::compare are the same thing as compare(*this, other).

stock::stock is a constructor that will set item to p1 and leave the other variables undefined. This is useful for creating temporary objects to use for comparisons like x <= stock("Hotdog").

Because **stock** is now less-then-comparable you can store the query results in a set:

```
set<stock> result;
query.storein(result);
```

And you can now use it like any other set, for example:

```
cout << result.lower_bound(stock("Hamburger"))->item << endl;
```

will return the first item that begins with Hamburger.

You can also now use it will any STL algorithm that require the values to be less-then-comparable.

The general format so far is:

```
sql_create_base_#(NAME, CMP, 0, TYPE1, ITEM1, ... TYPE#, ITEM#)
```

where CMP is that the number that tells c++ that if the first cmp variables are the same then the two structures are the same.

7.3 sql_create_basic with Additional Constructor

The last zero in the last example if for creating another constructor. Let this zero be m then it will create a constructor which will populate the first n variables. For example:

```
sql_create_basic_5(stock, 1, 5,
                   string, item, // type, id,
                   int, num,
                   double, weight,
                   double, price,
                   MysqlDate, date)
```

will also define:

```
struct stock {
  ...
  stock(const string&, const int&, const double&,
        const double&, const MysqlDate&);
  set(const string&, const int&, const double&,
      const double&, const MysqlDate&);
}
```

7.4 sql_create_basic General Format

Thus the general format for sql_create_basic is

```
sql_create_basic_#(NAME, CMP, CNST, TYPE1, ITEM1, ..., TYPE#, ITEM#)
```

Where:

- # is the number of valuables in the vector
- NAME is the name of the structure you wish to create
- CMP is the number that tells c++, if not set to 0, that if the first cmp variables are the same then the two structures are the same.
- CNST is the number, if not set to 0, that will create a constructor which will populate the first n variables.
- TYPE1 is the type name for first item and ITEM1 is the valuables name for the first item etc..

7.5 sql_create_basic_c_order

You can also specify an alternate order for when mysql populates the structure. For example:

```
sql_create_basic_c_order_5(stock, 2, 5,
                           MysqlDate, date, 5, // type, id, order
                           double, price, 4,
                           string, item, 1,
                           int, num, 2,
                           double, weight, 3)
```

This will create a similar structure as in the previous example except that that the order of the data items will be different and c++ will use the first two items to compare with (date, price). However because a custom order is specified you can use the same query to populate the set. It will fill `date` with the first 5th item of the query result set, `price` with the 4th, etc...

7.6 sql_create_basic_c_order General Format

Thus the general format for sql_create_basic is

```
sql_create_basic_c_order_# (NAME, CMP, CNST,
                            TYPE1, ITEM1, ORDER1,
                            ...
                            TYPE#, ITEM#, ORDER#)
```

Where:

- # is the number of valuables in the vector
- NAME is the name of the structure you wish to create
- CMP is the number that tells c++, if not set to 0, that if the first cmp variables are the same then the two structures are the same.
- CNST is the number, if not set to 0, that will create a constructor which will populate the first n variables.
- TYPE1 is the type name for first item, ITEM1 is the valuable name for the first item, ORDER1 is the order number for the first item ...etc...

7.7 sql_create

In addition to the basic structures you can set up enhanced structures that also have methods defined to aid in the creation of queries and in the insertion of data in tables.

For example:

```
sql_create_5(stock, 1, 5,
             string, item, // type, id,
             int, num,
             double, weight,
             double, price,
             MysqlDate, date)
```

which will, in addition to that which is defined in sql_create_basic with Additional Constructor, define the equivalent to:

```
struct stock {
  ...
  static char *names[];
  static char *table;
  template <class Manip>
  stock_value_list<Manip> value_list(cchar *d = ",",  // basic form
```

```
                                   Manip m = mysql_quote) const;
      template <class Manip>
      stock_field_list<Manip> field_list(cchar *d = ",",
                                   Manip m = mysql_do_nothing) const;
      template <class Manip>
      stock_equal_list<Manip> equal_list(cchar *d = ",",
                                   cchar *e = " = ",
                                   Manip m = mysql_quote,
                                   ) const;

      template <class Manip>                            // bool form
      stock_cus_value_list<Manip> value_list([cchar *d, [Manip m,] ]
                                       bool i1,
                                       bool i2 = false, ... ,
                                       bool i5 = false) const;
      template <class Manip>                            // list form
      stock_cus_value_list<Manip> value_list([cchar *d, [Manip m,] ]
                                         stock_enum i1,
                                         stock_enum i2 = stock_NULL, ...,
                                         stock_enum i5 = stock_NULL) const;
      template <class Manip>                            // vector form
      stock_cus_value_list<Manip> value_list([cchar *d, [Manip m,] ]
                                         vector<bool> *i) const;

      ...(The logical equivalent for field_list and equal_list)...
    };
```

value_list() returns a special class that when used with the << operator with an ostream on the left will return a comma separated list with values properly quoted and escaped when needed.

field_list() return a special class than does the same thing but returns a list of fields that the structure holds which in this case is the same thing as the valuable names. The field names are not escaped or quoted

equal_list() returns a comma separated list with the format **field name** = **value**. The field name is not quoted or escaped and value is escaped or quoted as needed.

For example:

```
stock s("Dinner Roles",75,0.95,0.97,"1998-05-25");
cout << "Value List: " << s.comma_list() << endl;
cout << "Field List: " << s.field_list() << endl;
cout << "Equal List: " << s.equal_list() << endl;
```

Would return something like (with a little extra hand formating):

```
Value List: 'Dinner Roles',75,0.95,0.97,'1998-05-25'
Field List: item,num,weight,price,date
Equal List: item = 'Dinner Roles',num = 75,weight = 0.95,
            price = 0.97,date = '1998-05-25'
```

A combination of the field and value list can be used for insert or replace queries. For example:

```
query << "insert into stock (" << s.field_list() ") values "
      << s.value_list();
```

will insert `s` into table stock.

You can also use SQLQuery::insert or SQLQuery::replace (and thus Query::insert or Query::replace) as a short cut to accomplish the same task like so:

```
query.insert(s);
```

It will use s.table for the table name which defaults to the name of the structure.

You can also specify an different delimiter "d". If none is specified it defaults to ",". With this you can use the delimiter " AND " for equal_list to aid in update and select queries. For example:

```
stock s2 = s;
s2.item = "6 Dinner Roles";
query << "UPDATE TABLE stock SET " << s2.equal_list()
      << " WHERE " << s.equal_list(" AND ");
```

would produce the query:

```
UPDATE TABLE stock SET item = '6 Dinner Roles',num = 75,weight = 0.95,
                       price = 0.97,date = '1998-05-25'
                 WHERE item = 'Dinner Roles' AND num = 75
                       AND weight = 0.95 AND price = 0.97
                       AND date = '1998-05-25'
```

which will change the entree in the table so that item is now "6 Dinner Roles" instead of "Dinner Roles"

You can use **SQLQuery::update** (and thus **Query::update**) as a short cut to accomplishing the same task like so:

```
stock s2 = s;
s2.item = "6 Dinner Roles";
query.update(s,s2);
```

Like **SQLQuery::insert**, it will use s.table for the table name which defaults to the name of the structure.

You can also specify an different manipulator which will effect the way c++ quotes or escapes the values. This may be any valid stream manipulator that only effects the item to the right of manipulator. **value_list** and **equal_list** defaults to **escape** and **field_list** defaults to **do_nothing**. For **equal_list** the manipulator only effects the **value** part and not the **field name** part.

This can be useful creating exporting to a file where you don't want quotes around strings for example.

```
table_out << q.value_list("\ t", mysql_escape) << endl;
```

will append data to the file handle table_out.

The three non-basic forms allow you to specify which items are returned. For example:

```
cout << q.value_list(false,false,true,true,false) << endl; //bool form
cout << q.value_list(stock_weight, stock_price) << endl;   //list form
```

will both return:

```
0.95,0.97
```

The *bool form* excepts boolean arguments where each true/false represents an wether to show a valuable. False means not to show it while true means to show it. If you leave of some they are assumed to be false. For example:

```
cout << q.value_list(false,false,true,true) << endl;
```

is the same as the above example.

The *list form* allows you to specify which items to show. An enum values are created for each valuable with the name of struct plus the underscore character prefixed before it. For example: item becomes stock_item.

These forms can be useful is select queries. For example:

```
query << "SELECT * FROM stock WHERE "
      << q.equal_list(" AND ",stock_weight,stock_price);
```

would produce the query:

```
SELECT * FROM stock WHERE weight=0.95 AND price=0.97
```

which will select all rows from stock which have the same weight and price as *q*.

The *vector form* (not shown above) allows you to pass a boolean vector which is a time saver if you use the some pattern more than once as it avoids having to create the vector from the arguments each time. If `a` is a boolean vector then `a[0]` will hold wether to include the first variable `a[1]` the second etc... For example:

```
vector<bool> a;
a[0] = false; a[1] = false; a[2] = true; a[3] = true; a[4] = false;
query << "SELECT * FROM stock WHERE " << q.equal_list(" AND ", a);
```

will produce the same query as in the above example.

7.8 sql_create_c_names

You can also specify alternate field names like so:

```
sql_create_c_names_5(stock, 1, 5,
                     string, item, "item", // type, id, column name
                     int, num, "quantity",
                     double, weight, "weight",
                     double, price, "price"
                     MysqlDate, date, "shipment")
```

When **field_list** or **equal_list** is used it will use the given field names rather than the variable names for example:

```
stock s("Dinner Roles",75,0.95,0.97,"1998-05-25");
cout << "Field List: " << s.field_list() << endl;
cout << "Equal List: " << s.equal_list() << endl;
```

Would return something like (with a little extra hand formating):

```
Field List: item,quantity,weight,price,shipment
Equal List: item = 'Dinner Roles',quantity = 75,weight = 0.95,
            price = 0.97,shipment = '1998-05-25'
```

7.9 sql_create_c_names General Format

The general format is:

```
sql_create_c_names_# (NAME, CMP, CNST,
                      TYPE1, ITEM1, NAME1,
                      ...
                      TYPE#, ITEM#, NAME#)
```

where NAME1 is the name of the first field, etc. Everything else is the same as it is the same as in sql_create_basic_c_order General Format.

7.10 sql_create_c_order

As in sql_create_basic_c_order you may specify a custom order. The general from is:

```
sql_create_c_order_# (NAME, CMP, CNST,
                      TYPE1, ITEM1, ORDER1,
                      ...
                      TYPE#, ITEM#, ORDER#)
```

where everything is the same as in sql_create_basic_c_order General Format.

7.11 sql_create_complete

You can also specify both a custom order and custom field names. The general from is.

```
sql_create_complete_# (NAME, CMP, CNST,
                        TYPE1, ITEM1, NAME1, ORDER1,
                        ...
                        TYPE#, ITEM#, NAME#, ORDER#)
```

Where everything is the same as in sql_create_c_order General Format and sql_create_c_names General Format.

7.12 Changing the table name

In order to avoid having even more forms we decided not to allow you to specify a different table name in the actual macro call. The table name is used by **SQLQuery::insert**, **replace**, and **update**. However you can easeally change the default table name, which is the same as the struct name, by changing the reference **NAME::table()** returns to a different **const char *** For example:

```
stock::table() = "in_stock"
```

Will change the table name to "in_stock" in the examples used through out this guide.

7.13 Seeing the actual code

To see the actual code that the macro inserts use sql++pretty. For example:

```
sql++pretty < test.cc | less
```

7.14 Adding functionality

The best way to add functionality is through inheritance. Even though you could paste the code outputted from pretty.pl and modify it this is not recommended because it won't reflect future enhancements.

7.15 Other notes

Macros are defined for structures with up to 25 items. If you need more modify the underlying perl script custom.pl. This perl script is used to generate the header file. It in no way tries to parse C++ code.

The header file that the script custom.pl creates is close to a meg. However, please note that the 1 meg header file (custom-macros.hh) is NOTHING but macros. Therefor the compiler has to do very little work when reading is.

Also, everything included by the macro call is done in such a way that you can safely include the macro call in a header file and not have to worry about duplicate function calls or anything of the like.

Chapter 8

Long Names

By default the Mysql++ API uses both short names with out the **Mysql** or **mysql_** prefix and long names with the **Mysql** or **mysql_** prefix. If this causes name problems define the macro **MYSQL_NO_SHORT_NAMES** before including **mysql++**. This will force the use of long names only. The short and long names are mapped as follows:

Short Name	**Long Name**
BadQuery	MysqlBadQuery
Connection	MysqlConnection
ResNSel	ResNSel
ResUse	ResUse MysqlResUse
Result	MysqlRes
Field	MysqlField
Fields	MysqlFields
ResIter	MysqlResIter
ResultIter	MysqlResIter
Row	MysqlRow
MutableRow	MysqlMutableRow
FieldNames	MysqlFieldNames
Query	MysqlQuery
BadConversion	MysqlBadConversion
ColData	MysqlColData
MutableColData	MysqlMutableColData
quote	mysql_quote
quote_only	mysql_quote_only
quote_double_only	mysql_quote_double_only
escape	mysql_escape
do_nothing	mysql_do_nothing
ignore	mysql_ignore
Date	MysqDate
Time	MysqlTime
DateTime	MysqlDateTime
Set	MysqlSet
Null	MysqlNull
null_type	mysql_null_type
null	mysql_null
NullisNull	MysqlNullisNull
NullisZero	MysqlNullisZero
NullisBlank	MysqlNullisBlank

Appendices

Appendix A

Changelog

1.7.9 (May 1 2001) Sinisa Milivojevic <sinisa@mysql.com>

- Fixed a serious bug in Connection constructor when reading MySQL options
- Improved copy constructor and some other methods in Result / ResUse
- Many other minor improvements
- Produced a complete manual with chapter 5 included
- Updated documentation, including a Postscript format

1.7.8 (November 14 2000) Sinisa Milivojevic <sinisa@mysql.com>

- Introduced a new, standard way of dealing with C++ exceptions. MySQL++ now supports two different methods of tracing exceptions. One is by the fixed type (the old one) and one is standard C++ type by the usage of what() method. A choice of methods has to be done in building a library. If configure script is run with –enable-exception option , then new method will be used. If no option is provided, or –disable-exception is used, old MySQL++ exceptions will be enforced. This innovation is a contribution of Mr. Ben Johnson <ben@blarg.net>
- MySQL++ now automatically reads at connection all standard MySQL configuration files
- Fixed a bug in sql_query::parse to enable it to parse more then 99 char's
- Added an optional client flag in connect, which will enable usage of this option, e.g. for getting matched and not just affected rows. This change does not require any changes in existing programs
- Fixed some smaller bugs
- Added better handling of NULL's. Programmers will get a NULL string in result set and should use is_null() method in ColData to check if value is NULL
- Further improved configuration
- Updated documentation, including a Postscript format

1.7.6 (September 22 2000) Sinisa Milivojevic <sinisa@mysql.com>

- This release contains some C++ coherency improvements and scripts enhacements
- result_id() is made available to programmers to fetch LAST_INSERT_ID() value
- Connection constroctur ambiguity resolved, thanks to marc@mit.edu
- Improved cnnfigure for better finding out MySQL libraries and includes
- Updated documentation, including a Postscript format

1.7.5 (July 30 2000) Sinisa Milivojevic <sinisa@mysql.com>

- This release has mainl bug fixes and code improvements
- A bug in FieldNames::init has been fixed, enabling a bug free usage of this class with in what ever a mixture of cases that is required
- Changed behaviour of ResUse, Result and Row classes, so that they could be re-used as much as necessary, without any memory leaks, nor with any re-initializations necessary
- Fixed all potential leaks that could have been caused by usage of delete instead of delete[] after memory has been allocated with new[]
- Deleted all unused classes and macros. This led to a reduction of library size to one half of the original size. This has furthermore brought improvements in compilation speed
- Moved all string manipulation from system libraries to libmysqlclient, thus enabling uniformity of code and usage of 64 bit integers on all platforms, including Windows, without reverting to conditional compilation. This changes now requires usage of mysql 3.23 client libraries, as mandatory
- Changed examples to reflect above changes
- Configuration scripts have been largely changed and further changes shall appear in consecutive sub-releases. This changes have been done and shall be done by our MySQL developer Thimble Smith <tim@mysql.com>
- Changed README, TODO and text version of manual. Other versions of manual have not been updated
- Fixed .version “bug”. This is only partially fixed and version remains 1.7.0 due to some problems in current versions of libtool. This shall be finally fixed in a near future
- Several smaller fixes and improvements
- Added build.sh script to point to the correct procedure of building of this library. Edit it to add configure options of your choice

1.7 (May17 2000) Sinisa Milivojevic <sinisa@mysql.com>

- This is mainly a release dealing with bug fixes, consistency improvements and easier configure on some platforms
- A bug in fetch_row() method of ResUse class has been fixed. Beside changes that existed in a distributed patch, some additional error checking has been introduced
- A bug in escape manipulator has been fixed that could cause an error if all characters had to be escaped
- An inconsistency in column indexing has been fixed. Before this version, column names in row indexing with strings, i.e. row[<string>] , has been case sensitive, which was inconsistent with MySQL server handling of column names
- An inconsistency in conversion from strings to integers or floats has been fixed. In prior version a space found in data would cause a BadConversion exception. This has been fixed, but 100% consistency with MySQL server has not been targeted, so that other non-numeric characters in data will still cause BadConversion exception or error. As this API is used in applications, users should provide feedback if full compatibility with MySQL server is desired, in which case BadConversion exception or error would be abolished in some of future versions
- A new method in ColData class has been introduced. is_null() method returns a boolean to denote if a column in a row is NULL. Finally, as of this release, testing for NULL values is possible. Those are columns with empty strings for which is_null() returns true.
- Some SPARC Solaris installations had C++ exception problems with g++ 2.95.2 This was a bug that was fixed in GNU gcc, as from release 2.95 19990728. This version was thoroughly tested and is fully functional on SPARC Solaris 2.6 with the above version of gcc.
- A 'virtual destructor ' warning for Result class has been fixed
- Several new functions for STL strings have been added. Those functions (see string_util.hh) add some of the functionality missing in existing STL libraries
- Conversion for 64 bit integers on FreeBSD systems has been added. On those systems _FIX_FOR_BSD_ should be defined in CXXFLAGS prior to configuring. Complete conversion to the usage of functions for integer conversion found in mysqlclient library is planned for one of the next releases
- A completely new, fully dynamic, dramatic and fully mutable result set has been designed and will be implemented in some of 2.x releases
- Several smaller fixes and improvements, including defaulting exceptions to true, instead of false, as of this version
- An up-to-date and complete Postscript version of documentation is included in this distribution
- Large chunks of this manual are changed, as well as README and TODO files.

1.6 (Feb 3 2000) Sinisa Milivojevic <sinisa@mysql.com>

- This is a major release as it includes new features and major rewrites
- Automatic quoting and escaping with streams. It works automatically , depending on the column type. It will work with < < on all ostream derived types. it is paricularly handy with query objects and strstreams. Automatic quoting and escaping on cout, cerr and clog stream objects is intentionally left out, as quoting / escaping on those stream objects is not necessary. This feature can be turned of by setting global boolean dont_quote_auto to true.
- Made some major changes in code, so that now execute method should be used only with SSQL and template queries, while for all other query execution of UPDATE's, INSERT's, DELETE's, new method exec() should be used. It is also faster.
- New method get_string is inroduced for easier handling / casting ColData into C++ strings.
- Major rewrite of entire code, which led to it's reduction and speed improvement. This also led to removal of several source files.
- Handling of binary data is introduced. No application program changes are required. One of new example programs demonstrates handling of binary data
- Three new example programs have been written and thoroughly tested. Their intention is to solve some problems addressed by MySQL users.
- Thorough changes is Makefile system has been made
- Better configuration scripts are written, thanks to D.Hawkins <dhawkins@cdrgts.com>
- Added several bug fixes
- Changed Manual and Changelog

1.5 (Dec 1 1999) Sinisa Milivojevic <sinisa@mysql.com>

- Fixed bug in template queries, introduced in 1.4 (!)
- Fixed connect bug
- Fixed several bug in type_info classes
- Added additional robustness in classes
- Added additional methods for SQL type info
- Changed Changelog and README

1.4 (Nov 25 1999) Sinisa Milivojevic <sinisa@mysql.com>

- Fixed bug in store and storein methods
- Fixed one serious memory leak
- Fixed a very serious bug generated by gcc 2.95.xx !!
- Added robustness in classes, so that e.g. same query and row objects can be re-used
- Changed sinisa_ex example to reflect and demonstrate this stability
- Changed Changelog and README
- Few other bug fixes and small improvements and speed-ups

1.3 (Nov 10 1999) Sinisa Milivojevic <sinisa@mysql.com>

- Fixed several erronous definitions
- Further changed source to be 2.95.2 compatible
- Expunged unused statements, especially dubious ones, like use of pointer_tracker
- Corrected bug in example file fieldinf1
- Finally fixed mysql_init in Connection constructor, which provided much greater stability !
- Added read and get options, so that clients, like mysqlgui can use it
- Changed Changelog and README
- Many other bug fixes.

1.2 (Oct 15 1999) Sinisa Milivojevic <sinisa@mysql.com>

- First offical release. Version 1.0 and 1.1 were releases by Sinisa before I (Kevin Atkinson) made him the offical maintainer,
- Many manual fixes.
- Changed README and Changelog
- Changed source to be compilable by gcc 2.95.xx, tribute to Kevin Atkinson <kevinatk@home.com>
- Added methods in Connection class which are necessary for fullfilling administrative functions with MySQL
- Added many bug fixes in code pertaining to missing class initializers , as notified by Michael Rendell <michael@cs.mun.ca>
- Sinisa Milivojevic <sinisa@mysql.com> is now the offical maintainer.

1.1 (Aug 2 1999) Sinisa Milivojevic <sinisa@mysql.com>

- Added several bug fixes
- Fixed memory leak problems and variables overlapping problems.
- Added automake and autoconf support by loic@ceic.com
- Added Makefile for manual
- Added support for cygwin
- Added example sinisa_ex (let modesty prevail) which used to crash a lot when memory allocation, memory leak and overlap problems were present. Smooth running of this example proves that all those bugs are fixed
- Corrected bugs in sql_query.cc regarding delete versus delete[] and string length in manip.cc
- Changed manual
- Changed README
- Many other smaller things

1.0 (June 9 1999) Michael Widenius <monty@monty.pp.sci.fi>

- Added patches from Orion Poplawski <orion@bvt.com> to support the UnixWare 7.0 compiler

.64.1.1a (Sep 27 1998)

- Fixed several bugs that caused my library to fail to compile with egcs 1.1. Hopefully it will still compile with egcs 1.0 however I have not been able to test it with egcs 1.0.
- Removed some problem causing debug output in sql++pretty.

.64.1a (Aug 1 1998)

- Added an (almost) full guide to using Template Queries.
- Fixed it so the SQLQuery will throw an exception when all the template parameters are not provided.
- Proofread and speedchecked the manual (it really needed it).
- Other minor document fixes.

.64.0.1a (July 31 1998)

- Reworked the Class Reference section a bit.
- Minor document fixes
- Added more examples for SSQLS.
- Changed the syntax of equal_list for SSQLS from equal_list (cchar *, Manip, cchar *) to (cchar *, cchar *, Manip).
- Added set methods to SSQLS. These new methods do the same thing as there corresponding constructors.
- Added methods for creating a mysql_type_info from a C++ type_info.

.64.a (July 24 1998)

- Changed the names of all the classes so they no longer have to have Mysql in the begging of it. However if this creates a problem you can define a macro to *only* use the old names instead.
- The Specialized SQL Structures (formally known as Custom Mysql Structures) changed from mysql_ to sql_.
- Added the option of using exceptions thoughout the API.
- ColData (formally known as MysqlStrings) will now throw an exception if there is a problem in the conversion.
- Added a null adapter.
- Added Mutable Result Sets
- Added a very basic runtime type identification for SQL types
- Changed the document format from POD to LyX .
- Am now using a modified version of Perceps to extract the class information directly from the code to make my life easier.
- Added an option of defining a macro to avoid using the automatic conversion with binary operators.
- Other small fixed I probully forgot to mentune.

.63.1.a

- Added Custom Mysql Structures.
- Fixed the Copy constructor of class Mysql
- Started adding code so that class Mysql lets it children now when it is leaving

- Attempted to compile it into a library but still need help. As default it will compile as a regular program.
- Other small fixes.

.62.a (May 3 1998)

- Added Template Queries
- Created s separate SQLQuery object that is independent of an SQL connection.
- You no longer have to import the data for the test program as the program creates the database and tables it needs.
- Many small bug fixes.

.61.1.a (April 28 1998)

- Cleaned up the example code in test.cc and included it in the manual.
- Added an interface layout plan to the manual.
- Added a reverse iterator.
- Fixed a bug with row.hh (It wasn't being included because of a typo).

.61.0.a

- Major interface changes. I warned you that the interface may change while it is in pre-alpha state and I wasn't kidding.
- Created a new and Separate Query Object. You can no longer execute queries from the Mysql object instead you have to create a query object with Mysql::query() and use it to execute queries.
- Added the comparison operators to MysqlDate, MysqlTime and MysqlDateTime. Fixed a few bugs in the MysqlDate... that effected the stream output and the conversion of them to strings.
- Reflected the MysqlDate... changes in the manual.
- Added a new MysqlSet object and a bunch of functions for working with mysql set strings.

.60.3a (April 24 1998)

- Changed strtoq and strtouq to strtoll and strtull for metter compatibility Minor Manual fix.
- Changed makefile to make it more compatible with Solaris (Thanks Chris H)
- Fixed bug in comparison functions so that they would compare in he right direction.
- Added some items to the to do list be sure to have a look.

Appendix B

To Do

These are in the order I plan on implementing them.

- Improve the runtime type identification system for the sql types which will be needed for the mutable results sets and the binary operators in particular.
- To move properly all 64 int handling to libmysqlclient
- To implement dynamic, fully mutable result sets
- To improve configure for better detection of mysql includes and libs
- Improve the mutable results sets so that they can be assigned types that are not strings setting the sql type apropprestly. Also allow the SQLtype to be fixed so that when the programmer assigned a type to the data field that is not compatible with that sql type it will through an exception. For example setting an string to an int.
- Change the behavior of MysqlString when used with binary operators. Instead of converting to the type on the other side of the operator have it convert to the type the Mysql server said it originally was.
- Improve the Null adapters to make them more intelligently.
- Be able to store the result set in an assignable container that stores the results in the format they were originally stored in on the server. (Not sure what the best way to go about this is. If you have any ideas let me know.)
- Better handling of the destruction of the Mysql class. Have it first tell all its children that its parent is getting destroys and have then respond appropreatly. (Partly implemented as of version .63.1.a)
- Create a container to hold Mysql enums and sets as a bit set as opposed to a list of STL set.

If you have anything else you want to see let us know at sinisa@mysql.com or monty@mysql.com.

Appendix C

Credits

The following is an informal list of programs and people I would like to thank.

- Cygnus - for the great compiler (egcs.cygnus.com)
- Perl - for making my life in general a lot easier (www.perl.com)
- Lyx - as a great tool for helping me with this manual (www.lyx.org)
- perceps - As a great tool for extracting documentation from the source (friga.mer.utexas.edu/mark/perl/perceps/)
- latex2html - For making the html version of this document possible (www-dsed.llnl.gov/files/programs/unix/latex2html/)
- lynx - For manking the text version of this document possible.
- Mysql - for obvious reasons (www.tcx.se)
- Xemacs - the editor of choice
- Debian/GNU Linux - The platform I developed this on (www.debian.org)
- Chris Halverson - For helping me get it to compile under Solaris.
- Fredric Fredricson - For a long talk about automatic conversions.
- Michael Widenius - Mysql developer who has been very supportive of my efforts.
- Paul J. Lucas -For the original idea of treating the query object like a stream.
- Scott Barron - For helping me with the shared libraries.
- Jools Enticknap - For giving me the Template Queries Idea.
- M. S. Sriram - For a detailed dission of how the Template Queries should be implemented, the suggestion to throw exceptions on bad queries, and the idea of having a back-end independent query object (ie SQLQuery).
- Sinisa Milivojevic - For becoming the new offical maintainer.
- D. Hawkins and E. Loic for their autoconf + automake contribution.

Appendix D

Copyright

The Mysql++ API is copyright 1998 by Kevin Atkinson and 1999 by MySQL and is released under the LGPL license .

The intent of doing this is allow developers to use my library to develop commercial programs and to allow it be distributed with commercial databases.

D.1 LGPL

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

GNU LIBRARY GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does. 1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

> a) The modified work must itself be a software library.
> b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
> c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
> d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

> a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
> b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
> c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
> d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

> a) Accompany the combined library with a copy of the same work based on the Library,

uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that

distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the library's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.
>
> You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.
>
> <signature of Ty Coon>, 1 April 1990 Ty Coon, President of Vice

That's all there is to it!

Appendix E

Feedback

Since October 1999, all maintenance has been transferred to Sinisa Milivojevic (sinisa@mysql.com) and Michael Widenius (monty@mysql.com).

Send your feedback to any of these addresses, or even better to the mailing list mysql-plusplus@lists.mysql.com.