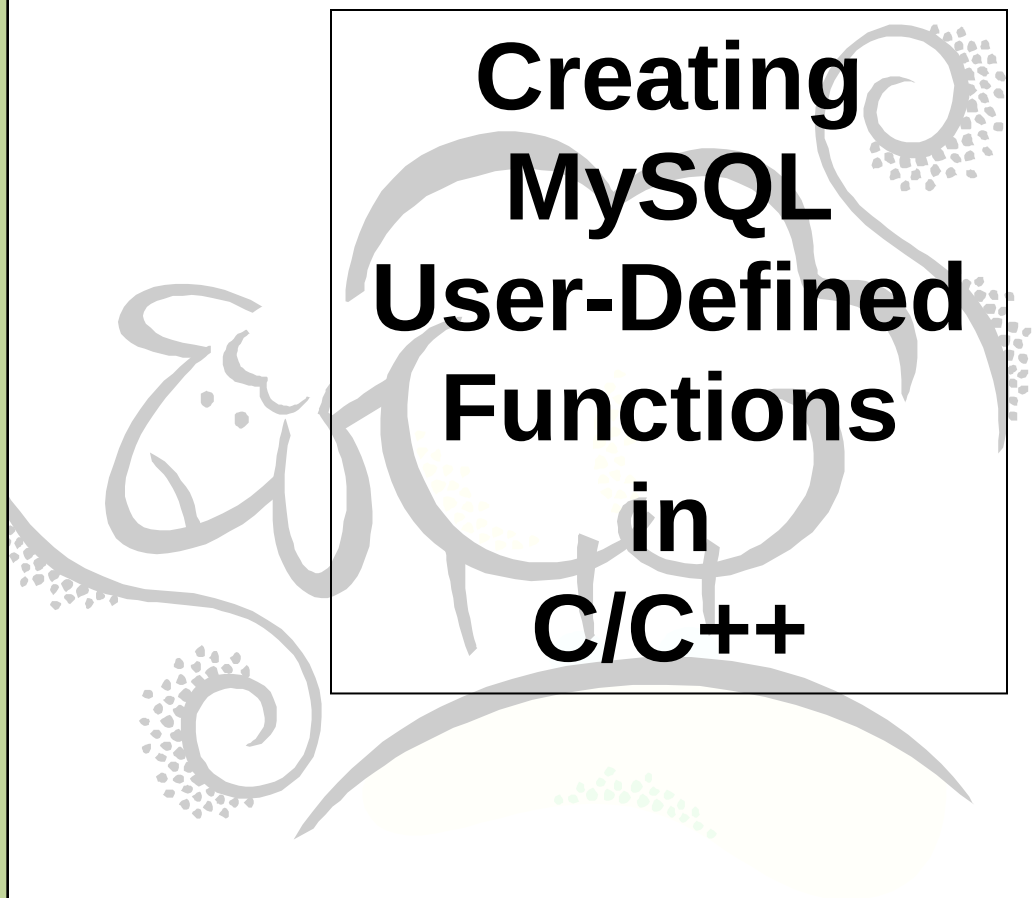
www.sheepsqueezers.com



Creating MySQL User-Defined Functions in C/C++

This presentation as well as other presentations and documents found on the sheepsqueezers.com website may contain quoted material from outside sources such as books, articles and websites. It is our intention to diligently reference all outside sources. Occasionally, though, a reference may be missed. No copyright infringement whatsoever is intended, and all outside source materials are copyright of their respective author(s).

Table of Contents

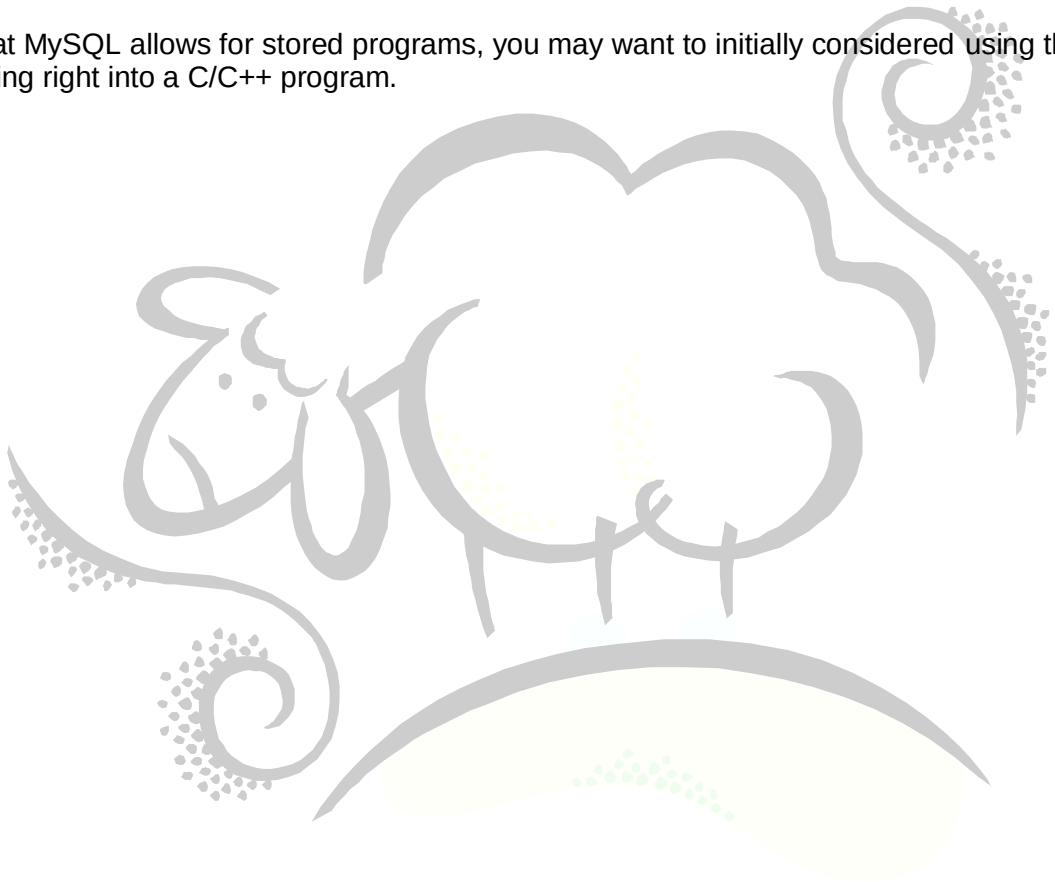
Introduction

This document describes how to create a user-defined function, whether a simple function or an aggregate function used with a SQL GROUP BY, to be used within your MySQL session. This document describes the creation process using a Linux machine with the native C/C++ compiler, although the process is similar on a Windows-based platform.

Please see the MySQL documentation *Chapter 22: Extending MySQL* for more information than is presented in this document. Much of this document was taken from this chapter.

Note that in order to create user-defined functions, the version of MySQL you are running must have been linked dynamically rather than statically. If you are going to write or use UDFs, you must compile MySQL yourself using dynamic linking (or possibly download a dynamically-linked version to install).

Now that MySQL allows for stored programs, you may want to initially considered using that instead of jumping right into a C/C++ program.



Pre-Requisites

Before you start, you need the following set up in your Linux account:

1. An account on a Linux machine (preferably root access)
2. A userid and password to a MySQL database
3. Create a directory call `lib` in your `$HOME` account and add these lines to your `.bash_profile`:

```
LD_LIBRARY_PATH=$HOME/lib
export LD_LIBRARY_PATH
```

4. The following C header files in the directory where you are going to write your programs and create your shared object file: `dbug.h`, `m_string.h`, `my_config.h`, `my_global.h`, `mysql_com.h`, `mysql_version.h`, `raid.h`, `m_ctype.h`, `my_alloc.h`, `my_dir.h`, `my_pthread.h`, `mysql.h`, `my_sys.h`. These header files can be found in the MySQL directory or in the tarball that comes with MySQL for Linux (for example, mysql-max-4.0.20-pc-linux-i686.tar.gz).
5. The following compiler command-line at the ready:

```
c++ -I . -o My_DLL_Name.so -shared My_DLL_Name.cpp 2>compiler_results.log
```

6. The ability to copy `My_DLL_Name.so` from your `lib` subdirectory to `/usr/lib`. If you are logged in as root you can do this, otherwise you will have to ask your system administrator to move it to this location. Note that there may be other directories you can write to, but MySQL must have these directories in its load library path.
7. You should be using MySQL-Max which supports dynamic loading. If not, see the MySQL manual on how to get this to work.
8. When you define the function prototypes – See Appendix A – make sure that you state `extern "C"` and not `extern "C" __declspec( dllexport )`.

Writing a Simple User-Defined Function

A simple user-defined function (UDF) is any function that returns a result for each row passed to it. In contrast, an aggregate UDF works in conjunction with the GROUP BY statement and returns a result based on computations on the rows which make up a group.

First, you will need to include a standard piece of code into your UDF C++ program. See Appendix A for example code. You will need to include the following functions in your C++ program to get it to work in MySQL:

1. `myudf()` – required function
2. `myudf_init()` – optional initialization function
3. `myudf_deinit()` – optional de-initialization function

Note that `myudf()` is the name of the C++ function, `MYUDF()` will be the name of the SQL function you use with the SELECT statement, and the `MYUDF.so` is the shared object library you must copy over to `/usr/lib` in order for MySQL to find your function(s).

Based on your **return type**, choose one of these `myudf()` functions :

For STRING functions, choose this:

```
char *myudf(UDF_INIT *initid, UDF_ARGS *args,char *result, unsigned long
*length,char *is_null, char *error);
```

For INTEGER functions, choose this:

```
long long myudf(UDF_INIT *initid, UDF_ARGS *args,char *is_null, char *error);
```

For REAL functions, choose this:

```
double myudf(UDF_INIT *initid, UDF_ARGS *args,char *is_null, char *error);
```

The initialization and de-initialization functions are defined as:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
void xxx_deinit(UDF_INIT *initid);
```

Note that the initialization function should fill in any members that it wishes to change:

`my_bool maybe_null`
: `myudf_init()` should set maybe_null to 1 if `myudf()` can return `NULL`. The default value is 1 if any of the arguments are declared `maybe_null`.

`unsigned int decimals`
: Number of decimals to pass back.

`unsigned int max_length`
: The maximum length of the string result. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimals indicated by the initid->decimals. For numeric functions, the length includes any sign or decimal point characters. If you want to return a blob, you can set this to 65K or 16M.

`char *ptr`

A pointer that the function can use for its own purposes. For example, functions can use initid->ptr to communicate allocated memory between functions. In myudf_init(), allocate the memory and assign it to this pointer:

```
initid->ptr = allocated_memory;
```

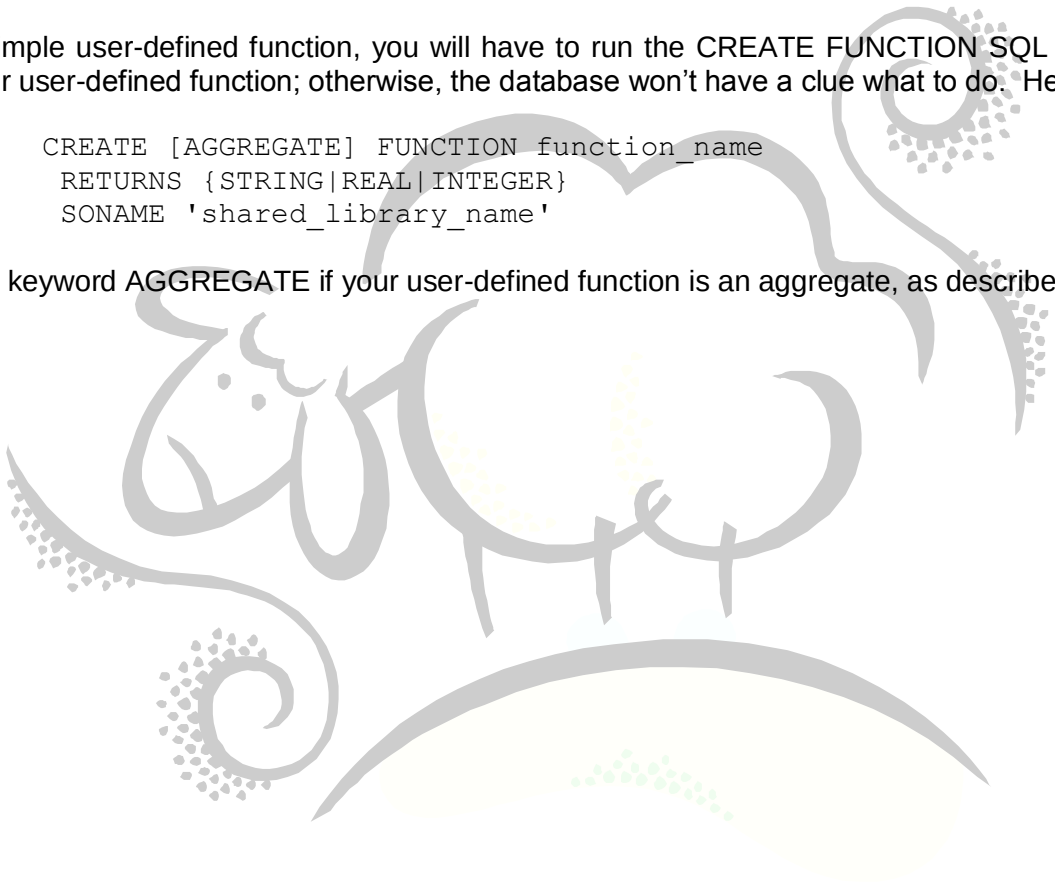
In myudf() and myudf_deinit(), refer to initid->ptr to use or deallocate the memory.

When an SQL statement invokes `MYUDF()`, MySQL calls the initialization function `myudf_init()` to let it perform any required setup, such as argument checking or memory allocation. If `myudf_init()` returns an error, the SQL statement is aborted with an error message and the main and de-initialization functions are not called. Otherwise, the main function `myudf()` is called once for each row. After all rows have been processed, the deinitialization function `myudf_deinit()` is called so that it can perform any required cleanup.

For a simple user-defined function, you will have to run the CREATE FUNCTION SQL code in order to then use your user-defined function; otherwise, the database won't have a clue what to do. Here is the syntax:

```
CREATE [AGGREGATE] FUNCTION function_name
 RETURNS {STRING|REAL|INTEGER}
 SONAME 'shared_library_name'
```

Use the keyword AGGREGATE if your user-defined function is an aggregate, as described in the next section.



Writing an Aggregate User-Defined Function

For an aggregate UDF, you must also provide the following functions:

`myudf_reset()` – required. Reset sum and insert the argument as the initial value for a new group.
`Myudf_add()` – required. Add the argument to the old sum.

When using aggregate UDFs, MySQL works in this way:

1. Call `myudf_init()` to let the aggregate function allocate the memory it will need to store results.
2. Sort the table according to the GROUP BY expression.
3. For the first row in a new group, call the `myudf_reset()` function.
4. For each new row that belongs in the same group, call the `myudf_add()` function.
5. When the group changes or after the last row has been processed, call `myudf()` to get the result for the aggregate.
6. Repeat 3-5 until all rows have been processed.
7. Call `myudf_deinit()` to let the UDF free any memory it has allocated.

All functions must be thread-safe, meaning that you are not allowed to allocate any global or static variables that change. If you need memory, you should allocate it in `myudf_init()` and free it in `myudf_deinit()`.

For a simple user-defined function, you will have to run the CREATE FUNCTION SQL code in order to then use your user-defined function; otherwise, the database won't have a clue what to do. Here is the syntax:

```
CREATE [AGGREGATE] FUNCTION function_name
 RETURNS {STRING|REAL|INTEGER}
 SONAME 'shared_library_name'
```

Use the keyword AGGREGATE if your user-defined function is an aggregate, as described above.

Appendix A – Example Using the Aggregate String Concatenation Function GROUP_CONCAT

```
/* Copyright © 2002 MySQL AB

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.

   This program 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 General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA */

/*
** Aggregate String Concat file of UDF (user definable functions) that are dynamically loaded
** into the standard mysqld core.
**
** The functions name, type and shared library is saved in the new system
** table 'func'.  To be able to create new functions one must have write
** privilege for the database 'mysql'. If one starts MySQL with
** --skip-grant, then UDF initialization will also be skipped.
**
** Syntax for the new commands are:
** create function <function_name> returns {string|real|integer}
**             soname <name_of_shared_library>
** drop function <function_name>
**
** Each defined function may have a xxxx_init function and a xxxx_deinit
** function.  The init function should alloc memory for the function
** and tell the main function about the max length of the result
** (for string functions), number of decimals (for double functions) and
** if the result may be a null value.
**
** If a function sets the 'error' argument to 1 the function will not be
** called anymore and mysqld will return NULL for all calls to this copy
** of the function.
**
** All strings arguments to functions are given as string pointer + length
** to allow handling of binary data.
** Remember that all functions must be thread safe. This means that one is not
** allowed to alloc any global or static variables that changes!
** If one needs memory one should alloc this in the init function and free
** this on the __deinit function.
**
** Note that the init and __deinit functions are only called once per
** SQL statement while the value function may be called many times
**
** Function 'group_concat' returns the concatenation of strings in a SELECT query with "GROUP BY" clause (AGGREGATION).
**
** A dynamicly loadable file should be compiled shared.
** (something like: gcc -shared -o my_func.so myfunc.cc).
** You can easily get all switches right by doing:
** cd sql ; make udf_example.o
** Take the compile line that make writes, remove the '-c' near the end of
** the line and add -shared -o udf_example.so to the end of the compile line.
** The resulting library (udf_example.so) should be copied to some dir
** searched by ld. (/usr/lib ?)
** If you are using gcc, then you should be able to create the udf_example.so
** by simply doing 'make udf_example.so'.
**
** After the library is made one must notify mysqld about the new
** functions with the commands:
**
** CREATE AGGREGATE FUNCTION group_concat RETURNS STRING SONAME "MyGroupConcat.dll";
**
** After this the functions will work exactly like native MySQL functions.
** Functions should be created only once.
**
** The functions can be deleted by:
**
** DROP FUNCTION group_concat;
**
** The CREATE FUNCTION and DROP FUNCTION update the func@mysql table. All
** Active function will be reloaded on every restart of server
** (if -skip-grant-tables is not given)
**
** If you ge problems with undefined symbols when loading the shared
** library, you should verify that mysqld is compiled with the -rdynamic
** option.
```

```
**
** If you can't get AGGREGATES to work, check that you have the column
** 'type' in the mysql.func table.  If not, run 'mysql_fix_privilege_tables'.
**
*/

#ifdef STANDARD
#include <stdio.h>
#include <string.h>
#ifdef __WIN__
typedef unsigned __int64 ulonglong;    /* Microsofts 64 bit types */
typedef __int64 longlong;
#else
typedef unsigned long long ulonglong;
typedef long long longlong;
#endif /*__WIN__*/
#else
#include <my_global.h>
#include <my_sys.h>
#endif
#include <mysql.h>
#include <m_ctype.h>
#include <m_string.h>          // To get strmov()

/*
** Aggregate Function.  MUST CHANGE THE NAME MYUDFNAME to your own UDF function name!!
*/

/* These must be right or mysqld will not find the symbol! */

extern "C" my_bool MYUDFNAME_init( UDF_INIT* initid, UDF_ARGS* args, char* message );
extern "C" void MYUDFNAME_deinit( UDF_INIT* initid );
extern "C" void MYUDFNAME_reset( UDF_INIT* initid, UDF_ARGS* args, char* is_null, char *error );
extern "C" void MYUDFNAME_add( UDF_INIT* initid, UDF_ARGS* args, char* is_null, char *error );
extern "C" char * MYUDFNAME(UDF_INIT * initid, UDF_ARGS *args, char *result, unsigned long *length, char *is_null, char *
/*error*/ );

/*
** Syntax for the new aggregate commands are:
** create aggregate function <function_name> returns {string|real|integer}
**             soname <name_of_shared_library>
**
*/
```

Appendix B – References

Click on the links below to find out more on Amazon.com's website.

1. MySQL by Paul DuBois (ISBN-13: 978-0672329388)
2. MySQL Stored Procedure Programming by Guy Harrison and Steven Feuerstein (ISBN-13: 978-0596100896)

Support sheepsqueezers.com

If you found this information helpful, please consider supporting sheepsqueezers.com. There are several ways to support our site:

- ☐ Buy me a cup of coffee by clicking on the following link and donate to my PayPal account: Buy Me A Cup Of Coffee?.
- ☐ Visit my Amazon.com Wish list at the following link and purchase an item: http://amzn.com/w/3OBK1K4EIWIR6

Please let me know if this document was useful by e-mailing me at comments@sheepsqueezers.com.