5 Database handling with MySQL

5.1 On not reinventing the wheel

It is pretty sensible to not start from scratch at every project so we build on work done by others who came this way. All of the C libraries we are using were written by someone from even lower level bits and pieces and we can access and modify this code should we want to. This is what Open Systems is all about. In practice we simply want to use reliable services, and the first two we are going to use are ubiquitous – databases access and graphical image generation.

5.2 MySQL C API

To access the set of MySQL functions we make sure our compiler can access the header and libraries of MySQL and simply call some mysql functions we have not had to write ourselves.

In this next example we join the database code onto the last work we did and insert the decoded data into a database, and read out what we have stored there so far.

In a new terminal window we type:

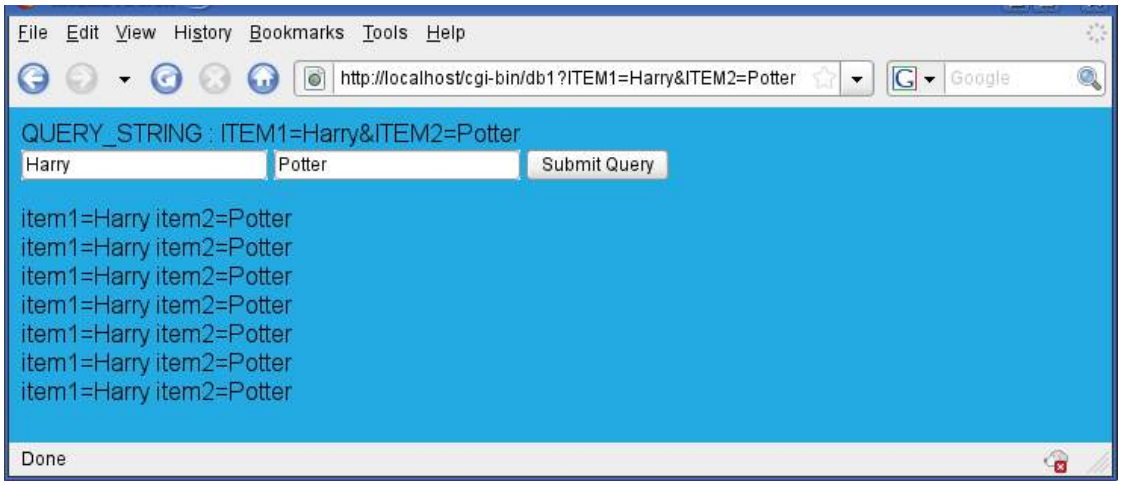
mysql test

then:

create table CIL (ITEM1 varchar(255),ITEM2 varchar(255));

This creates a table for the program to write to in the test database that should already be created in your MySQL setup.

It looks like this if it all works out OK.



The data types declared **MYSQL**, **MYSQL_RES**, and **MYSQL_ROW** are **preprocessor definitions** that stand for more complex C declarations in mysql.h and all we need to know is how to call them.

The documentation at MySQL provides all the information you will need to do more complex operations.

```
/*****************************************************************
* C Programming in Linux (c) David Haskins 2008
* chapter5_1.c                                                  *
*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <mysql/mysql.h>
#include "c_in_linux.h"

int main(int argc, char *argv[], char *env[])
{
        char value1[255] = "",value2[255] = "",SQL[1024]="";
        int rc = 0;

        MYSQL *conn = NULL;
        MYSQL_RES *result = NULL;
        MYSQL_ROW row;

        printf("Content-type:text/html\n\n<html><body bgcolor=#23abe2>\n");
        strncpy(value1,(char *) getenv("QUERY_STRING"),255);
        printf("QUERY_STRING : %s<BR>\n", value1 );
        printf("<form>\n");
        //call the decode_value function to get value of "ITEM1"
        decode_value( "ITEM1=", (char *) &value1, 255);
        if(strlen(value1) > 0 )
                printf("<input type=\"TEXT\" name=\"ITEM1\" value=\"%s\">\n",value1);
        else
                printf("<input type=\"TEXT\" name=\"ITEM1\">\n");
        //call the decode_value function to get value of "ITEM2"
        decode_value( "ITEM2=", (char *) &value2, 255);
        if(strlen(value2) > 0 )
                printf("<input type=\"TEXT\" name=\"ITEM2\"    value=\"%s\">\n",value2);
        else
                printf("<input type=\"TEXT\" name=\"ITEM2\">\n");
        printf("<input type=\"SUBMIT\">");
        printf("</form></body></html>\n");
        //OPEN DATABASE
        conn = mysql_init((MYSQL *) 0);
        mysql_options(conn,MYSQL_READ_DEFAULT_GROUP,"mysqlcapi");
        mysql_real_connect(conn, "localhost","","","test",0, NULL, 0);
        //INSERT IF THERE IS ANY DATA
        if(strlen(value1) > 0 || strlen(value2) > 0)
        {
        sprintf(SQL,"insert into CIL values ('%s','%s')",value1,value2);
                rc = mysql_query(conn,SQL);
        }
        //READ
        rc = mysql_query(conn,"select * from CIL");
        result = mysql_use_result(conn);
        while( (row = mysql_fetch_row(result)) != NULL)
        {
                        printf("item1=%s   item2=%s<br>",row[0],row[1]);
        }
        mysql_free_result(result);
        mysql_close(conn);
 return 0;  }
```

Understood all that?

In barely 10 lines (highlighted) of C we have inserted our data into a database table and can read it out again, which is pretty painless. With **mysql_init** we obtain a pointer to a data structure of type MYSQL, we can then se this to connect to the test database with **mysql_options** and **mysql_real_connect,** then we execute SQL statements just as we would in a terminal session. The results of a query can be retrieved as a sequence of MYSQL_ROW arrays of strings with **mysql_use_results**. We free up the memory used with **mysql_free_result** and close the database with **mysql_close**.

As is usual with C libraries, you need to be able understand the usually sparse documentation to understand the function calls, and for MySQL 5.1 we can find all this information at:

http://dev.mysql.com/doc/refman/5.1/en/index.html

The MySQL 5.1 Reference Manual / Connectors and APIs / MySQL C API includes:

mysql_affected_rows()	mysql_insert_id()
mysql_autocommit()	mysql_kill()
mysql_change_user()	mysql_library_end()
mysql_character_set_name()	mysql_library_init()mysql_list_dbs()
mysql_close()	mysql_list_fields(
mysql_commit()	mysql_list_processes()
mysql_connect()	mysql_list_tables()
mysql_create_db()	mysql_more_results()
mysql_data_seek()	mysql_next_result()
mysql_debug()	mysql_num_fields()
mysql_drop_db()	mysql_num_rows()
mysql_dump_debug_info()	mysql_options()
mysql_eof()	mysql_ping()
mysql_errno()	mysql_query()
mysql_error()	mysql_real_connect()
mysql_escape_string()	mysql_real_escape_string()
mysql_fetch_field()	mysql_real_query()
mysql_fetch_field_direct()	mysql_refresh(
mysql_fetch_fields()	mysql_reload()
mysql_fetch_lengths()	mysql_rollback()
mysql_fetch_row()	mysql_row_seek()
mysql_field_count()	mysql_row_tell()
mysql_field_seek()	mysql_select_db()
mysql_field_tell()	mysql_set_character_set()
mysql_free_result()	mysql_set_local_infile_default()
mysql_get_character_set_info()	mysql_set_local_infile_handler()
mysql_get_client_info()	mysql_set_server_option()
mysql_get_client_version()	mysql_shutdown()
mysql_get_host_info()	mysql_sqlstate()
mysql_get_proto_info()	mysql_ssl_set()
mysql_get_server_info()	mysql_stat()
mysql_get_server_version()	mysql_store_result()
mysql_get_ssl_cipher(	mysql_thread_id()
mysql_hex_string()	mysql_use_result()
mysql_info()	mysql_warning_count()
mysql_init()	