MySQL C Connector

Ing. Alessandro Pellegrini, PhD
pellegrini@diag.uniroma1.it

Requisiti di sistema

- Stiamo scrivendo un programma in C:
  - abbiamo bisogno di un compilatore C!

- Stiamo scrivendo un programma che usa le librerie di MySQL:
  - devono essere installate!
  - gli header devono essere installati!

- Come compilare:
  - gcc -g *.c -o program `mysql_config --cflags --include --libs`

Makefile di esempio

```
CC = gcc
INCLUDES = -I/usr/local/include/mysql
LIBS = -L/usr/local/lib/mysql -lmysqlclient
all: myclient
main.o: main.c myclient.h
    $(CC) -c $(INCLUDES) main.c
aux.o: aux.c myclient.h
    $(CC) -c $(INCLUDES) aux.c
myclient: main.o aux.o
    $(CC) -o myclient main.o aux.o $(LIBS)
clean:
    rm -f myclient main.o aux.o
```

Come connettersi al DBMS

- La connessione è di tipo client/server, anche se in locale
  - Ci si connette tramite socket
  - Si deve fornire l'endpoint
- Il DBMS opera solo su connessioni autenticate
  - Dobbiamo dire qual è l'username e la password
  - Possiamo specificare (opzionale) un database su cui operare nella sessione

Come connettersi al DBMS

```
#include <stdlib.h>
#include <my_global.h>
#include <my_sys.h>
#include <mysql.h>

static char *opt_host_name = NULL; /* host (default=localhost) */
static char *opt_user_name = NULL; /* username (default=login name)*/
static char *opt_password = NULL; /* password (default=none) */
static unsigned int opt_port_num = 0; /* port number (use built-in) */
static char *opt_socket_name = NULL; /* socket name (use built-in) */
static char *opt_db_name = NULL; /* database name (default=none) */
static unsigned int opt_flags = 0; /* connection flags (none) */
static MYSQL *conn; /* pointer to connection handler */
```

Come connettersi al DBMS

```
int main (int argc, char **argv)
{
    /* initialize connection handler */
    conn = mysql_init(NULL);
    if(conn == NULL) {
        fprintf(stderr, "mysql_init() failed\n");
        exit(EXIT_FAILURE);
    }
    /* connect to server */
    if (mysql_real_connect (conn, opt_host_name, opt_user_name,
       opt_password, opt_db_name, opt_port_num, opt_socket_name,
       opt_flags) == NULL) {
        fprintf(stderr, "mysql_real_connect() failed\n");
        mysql_close(conn);
        exit(EXIT_FAILURE);
    }
```

Come connettersi al DBMS

```
    /* disconnect from server */
    mysql_close (conn);
    exit(EXIT_SUCCESS);
}
```

Gestione degli errori

```
if (...some MySQL function fails...) {
    print_error (conn, "...some error message...");
}
```

equivalente a:

```
if (...some MySQL function fails...) {
    fprintf (stderr, "...some error message...:\nError %u (%s): %s\n",
    mysql_errno (conn), mysql_sqlstate(conn), mysql_error (conn));
}
```

Esecuzione di statement SQL

- Schema di base:

```
if (mysql_query (conn, stmt_str) != 0) {
    /* failure; report error */
} else {
    /* success; what is the effect? */
}
```

Statement senza result set

```
if(mysql_query(conn, "INSERT INTO my_tbl SET name = 'My Name'")
       != 0) {
       print_error (conn, "INSERT statement failed");
} else {
       printf ("INSERT statement succeeded: %lu rows affected\n",
              (unsigned long)mysql_affected_rows (conn));
}
```

Statement con result set

- Quando una query restituisce dei risultati, questi devono essere processati dal client secondo uno schema ben preciso
  - I DBMS lavorano *sempre* a tabelle
- I passi fondamentali sono:
  1. Far generare alla libreria delle strutture dati che consentano di operare sul result set
     - `mysql_store_result()`
     - `mysql_use_result()` (un solo result set alla volta)
  2. Iterare su ciacuna riga del result set
     - `mysql_fetch_row()`
  3. Liberare le strutture dati necessarie alle operazioni sul result set
     - `mysql_free_result()`

Iterazione sul result set di base

```
MYSQL_RES *res_set;
MYSQL_ROW row;
unsigned int i;

res_set = mysql_store_result (conn);

/* error handling omitted */

while ((row = mysql_fetch_row (res_set)) != NULL) {
    for (i = 0; i < mysql_num_fields (res_set); i++) {
        printf ("%s\t", row[i] != NULL ? row[i] : "NULL");
    }
    printf("\n");
}
```

Metadati del result set

- `mysql_num_rows()`: numero delle righe nel result set
- and `mysql_num_fields()`: numero delle colonne nel result set
- `mysql_fetch_lengths()`: la lunghezza di ciascun valore nelle colonne, nella riga corrente
- `mysql_fetch_field()`: nome delle colonne, tipo delle colonne, dimensione massima di tutti gli elementi nelle colonne, informazioni sulla tabella di origine dei dati
  - struttura `MYSQL_FIELD`

- Un esempio è il programma 'metadata' contenuto in: http://www.kitebird.com/mysql-book/sampdb-5ed/sampdb.tar.gz

Problemi con caratteri speciali

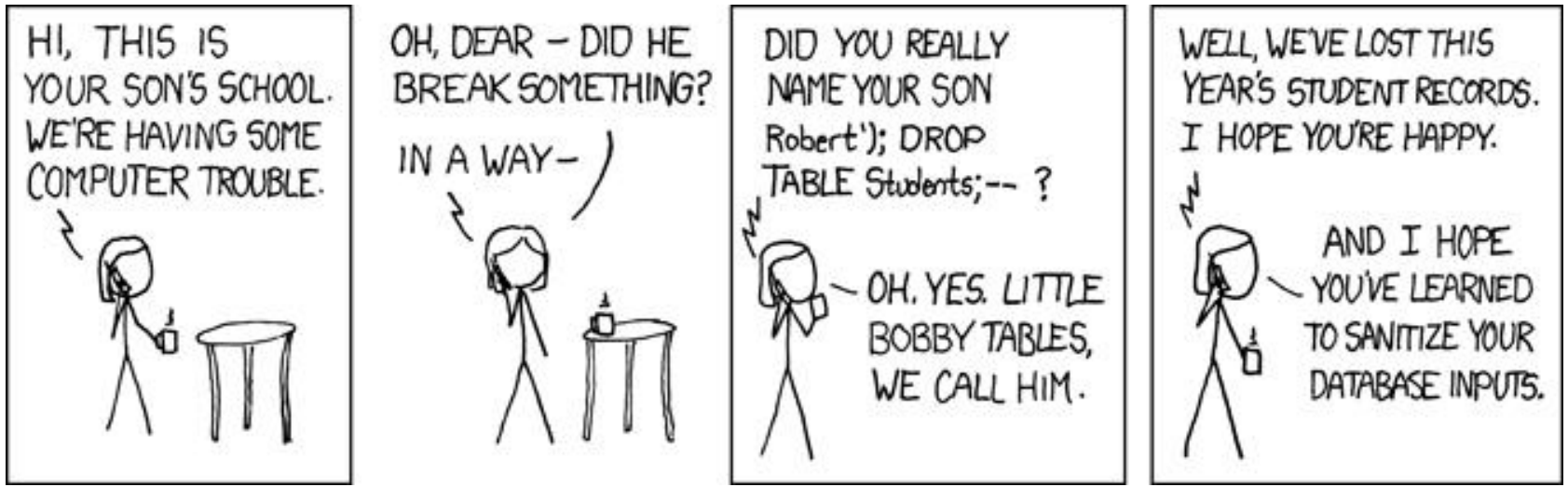
```
char stmt_buf[1024];
sprintf (stmt_buf, "SELECT * FROM mytbl WHERE name='%s'", name_val);
```

```
SELECT * FROM mytbl WHERE name='O'Malley, Brian'
```

```
mysql_real_escape_string()
```

```
SELECT * FROM mytbl WHERE name='O\'Malley, Brian'
```

Problemi con caratteri speciali



Prepared statements

- I prepared statement sono una tecnica che consente di inviare al DBMS una “query incompleta”, ossia con dei *placeholder*
  - `INSERT INTO score (event_id,student_id,score) VALUES(?,?,?)`
- Il DBMS analizza, verifica e compila tale query
- Successivamente, si possono “collegare” a tale query dei parametri
- Quando il DBMS riceve questi parametri, esegue la query completa e restituisce i risultati
  - Questa tecnica risolve i problemi di SQL injection e può migliorare le prestazioni delle applicazioni

Prepared statements

- I passi generali per utilizzare i prepared statement sono:
  1. Allocare un gestore di prepared statement
     - `mysql_stmt_init()`
  2. Inviare lo statement al server per precompilarlo
     - `mysql_stmt_prepare()`
  3. Fornire i parametri da inserire al posto dei placeholder (il numero deve essere esatto)
     - `mysql_stmt_bind_param()`
     - La struttura dati da utilizzare è `MYSQL_BIND`
  4. Inviare i parametri al server, e far eseguire lo statement
     - `mysql_stmt_execute()`
  5. Gestire gli effetti del prepared statement
     - `mysql_stmt_result_metadata()` per recuperare informazioni sul result set
     - `mysql_stmt_store_result()` per precaricare il set

Prepared statements: gestione degli errori

```
MYSQL_STMT *stmt;

if (stmt != NULL) {
      fprintf (stderr, "Error %u (%s): %s\n",
            mysql_stmt_errno (stmt),
            mysql_stmt_sqlstate(stmt),
            mysql_stmt_error (stmt));
}
```

MYSQL_BIND

```
typedef struct st_mysql_bind
{
   unsigned long *length;          /* output length pointer */
   my_bool       *is_null;         /* Pointer to null indicator */
   void          *buffer;          /* buffer to get/put data */
   /* set this if you want to track data truncations happened during fetch */
   my_bool       *error;
   enum enum_field_types buffer_type;    /* buffer type */
   /* output buffer length, must be set when fetching str/binary */
   unsigned long buffer_length;
   unsigned char *row_ptr;         /* for the current data position */
   unsigned long offset;           /* offset position for char/binary fetch */
   unsigned long length_value;     /* Used if length is 0 */
   unsigned int  param_number;     /* For null count and error messages */
   unsigned int  pack_length;      /* Internal length for packed data */
   my_bool       error_value;      /* used if error is 0 */
   my_bool       is_unsigned;      /* set if integer type is unsigned */
   my_bool       long_data_used;   /* If used with mysql_send_long_data */
   my_bool       is_null_value;    /* Used if is_null is 0 */
   ...
} MYSQL_BIND;
```

Prepared statements: tipi supportati

C Connector Type	C Type	SQL Type
MYSQL_TYPE_TINY	signed char	TINYINT field
MYSQL_TYPE_SHORT	short int	SMALLINT field
MYSQL_TYPE_LONG	int	INTEGER field
MYSQL_TYPE_INT24	int	MEDIUMINT field
MYSQL_TYPE_LONGLONG	long long int	BIGINT field
MYSQL_TYPE_NEWDECIMAL	char[]	Precision math DECIMAL or NUMERIC field
MYSQL_TYPE_FLOAT	float	FLOAT field
MYSQL_TYPE_DOUBLE	double	DOUBLE or REAL field
MYSQL_TYPE_BIT	char[]	BIT field
MYSQL_TYPE_TIMESTAMP	MYSQL_TIME	TIMESTAMP field
MYSQL_TYPE_DATE	MYSQL_TIME	DATE field
MYSQL_TYPE_TIME	MYSQL_TIME	TIME field
MYSQL_TYPE_DATETIME	MYSQL_TIME	DATETIME field
MYSQL_TYPE_YEAR	short int	YEAR field
MYSQL_TYPE_STRING	char[]	CHAR or BINARY field
MYSQL_TYPE_VAR_STRING	char[]	VARCHAR or BINARY field
MYSQL_TYPE_BLOB	char[]	BLOB or TEXT (use `max_length` to determine the maximum length)
MYSQL_TYPE_NULL		NULL-type field

Gestione del tempo

```
/*Structure which is used to represent datetime values inside MySQL.

  We assume that values in this structure are normalized, i.e. year <= 9999,
  month <= 12, day <= 31, hour <= 23, hour <= 59, hour <= 59. Many functions
  in server such as my_system_gmt_sec() or make_time() family of functions
  rely on this (actually now usage of make_*() family relies on a bit weaker
  restriction). Also functions that produce MYSQL_TIME as result ensure this.
  There is one exception to this rule though if this structure holds time
  value (time_type == MYSQL_TIMESTAMP_TIME) days and hour member can hold
  bigger values.
*/
typedef struct MYSQL_TIME {
  unsigned int year, month, day, hour, minute, second;
  unsigned long second_part; /**< microseconds */
  bool neg;
  enum enum_mysql_timestamp_type time_type;
} MYSQL_TIME;
```

Come chiamare stored procedure

- Una stored procedure può essere chiamata mediante un prepared statement del tipo `CALL(?,?,?)`
- Una stored procedure può avere più di un result set!
  - mysql_stmt_next_result() per avanzare al result set successivo
    - -1: non ci sono più result set
    - 0: c'è (almeno) un altro result set
    - >0: si è verificato un errore
- Se una procedura ha parametri OUT o INOUT, i loro valori verrano anche restituiti come un result set di una singola riga, alla fine di tutti gli altri result set
  - Il loro ordine corrisponde all'ordine di dichiarazione nella procedura
  - Si può verificare se si sta osservando questo result set controllando uno specifico bit: `conn->server_status & SERVER_PS_OUT_PARAMS`