Die MySQL C API

by Özcan Güngör
<ozcangungor(at)netscape.net>

About the author:
Ich benutze Linux seit 1997. Freiheit, Flexibilität, Open Source. Diese Eigenschaften mag ich.

Translated to English by:
Özcan Güngör
<ozcangungor(at)netscape.net>

Abstract:

In diesem Artikel werden wir lernen, wie man die C APIs ("Anwendunsprogrammierschnittstelle") für MySQL benutzt. Um diesen Artikel zu verstehen, solltest du folgendes in der Programmiersprache C kennen:

- Variablen in C
- Funktionen in C
- Zeiger in C

Einleitung

Die C APIs sind mit dem MySQL Quelltext erhältlich und sind Teil der *mysqlclient* Bibliothek. Sie werden verwendet, um eine Verbindung zu einer Datanbank herzustellen und Abfragen zu machen. Hierzu gibt es ein paar Beispiele im *clients* Verzeichnis des MySQL Quelltextes.

Typen von MySQL C Variablen

Die folgenden Typen von Variablen sind in der MySQL Bibliothek definiert. Wir benötigen diese Variablen, um die MySQL Funktionen zu benutzen. Die Variablen werden hier im Einzelnen erklärt, die Details sind allerdings nicht so wichtig, um Programme schreiben zu können.

MYSQL
Die folgende Struktur ist der Communication Handler, der benutzt wird um eine Verbindung mit der Datenbank herzustellen.

```
typedef struct st_mysql {
  NET           net;            /* Communication parameters */
```

```
  gptr          connector_fd;   /* ConnectorFd for SSL */
  char          *host,*user,*passwd,*unix_socket,
                *server_version,*host_info,*info,*db;
  unsigned int  port,client_flag,server_capabilities;
  unsigned int  protocol_version;
  unsigned int  field_count;
  unsigned int  server_status;
  unsigned long thread_id;      /* Id for connection in server */
  my_ulonglong affected_rows;
  my_ulonglong insert_id;       /* id if insert on table with NEXTNR */
  my_ulonglong extra_info;              /* Used by mysqlshow */
  unsigned long packet_length;
  enum mysql_status status;
  MYSQL_FIELD   *fields;
  MEM_ROOT      field_alloc;
  my_bool       free_me;        /* If free in mysql_close */
  my_bool       reconnect;      /* set to 1 if automatic reconnect */
  struct st_mysql_options options;
  char          scramble_buff[9];
  struct charset_info_st *charset;
  unsigned int  server_language;
} MYSQL;
```

MYSQL_RES

Diese Struktur stellt das Ergebnis einer Abfrage, die Zeilen zurückgibt, dar. Die zurückgegebenen Daten werden Ergebnis−Set genannt.

```
typedef struct st_mysql_res {
  my_ulonglong row_count;
  unsigned int  field_count, current_field;
  MYSQL_FIELD   *fields;
  MYSQL_DATA    *data;
  MYSQL_ROWS    *data_cursor;
  MEM_ROOT      field_alloc;
  MYSQL_ROW     row;            /* If unbuffered read */
  MYSQL_ROW     current_row;    /* buffer to current row */
  unsigned long *lengths;       /* column lengths of current row */
  MYSQL         *handle;        /* for unbuffered reads */
  my_bool       eof;            /* Used my mysql_fetch_row */
} MYSQL_RES;
```

MYSQL_ROW

Diese Struktur ist eine typsichere Darstellung der Daten einer Zeile. Sie kann nicht als String (Zeichenkette), der mit einem NULL−Zeichen endet, verwendet werden, da sie aus Binärdaten bestehen kann und möglicherweise NULL−Zeichen enthält.

```
typedef struct st_mysql_field {
  char *name;                   /* Name of column */
  char *table;                  /* Table of column if column was a field */
  char *def;                    /* Default value (set by mysql_list_fields) */
  enum enum_field_types type;   /* Type of field. Se mysql_com.h for types */
  unsigned int length;          /* Width of column */
  unsigned int max_length;      /* Max width of selected set */
```

```
  unsigned int flags;           /* Div flags */
  unsigned int decimals;        /* Number of decimals in field */
} MYSQL_FIELD;
```

my_ulonglong
Dies ist der Typ, der für die Anzahl der Zeilen und für mysql_affected_rows(), mysql_num_rows(), und mysql_insert_id() verwendet wird. Mit diesem Typ lässt sich ein Bereich von 0 bis 1.84e19 darstellen. Auf einigen Systemen ist es möglich, dass der Versuch, einen Wert vom Typ my_ulonglong auszugeben, missglückt. Um einen solchen Wert auszugeben, sollte man ihn in unsigned long umwandeln, und das %lu Format von printf() verwenden. Beispiel:
printf("Number of rows: %lu\n", (unsigned long) mysql_num_rows(result));

typedef unsigned long my_ulonglong;

Eine Verbindung zu MySQL herstellen und eine Abfrage machen

Ab jetzt werde ich voraussetzen, dass MySQL installiert ist und ein Benutzer und eine Tabelle angelegt sind. Sollte es dabei zu Problemen kommen, wende dich bitte an die www.mysql.com Homepage.

Wie bereits vorher erwähnt, befinden sich die MySQL Bibliotheken in der mysqlclient Bibliothek. Um ein Programm zu übersetzen, ist es also nötig, die *−lmysqlclient* Compiler Option anzugeben. Die MySQL Header−Dateien liegen unter /usr/include/mysql (Der genaue Pfad kann von deiner Linux−Distribution abhängen). Der Kopf deines Programms muss also wie folgt aussehen:

#include <mysql/mysql.h>

Diese Header−Datei enthält die MySQL Variablen, Typen und Funktionen.

Nun muss eine Variable deklariert werden, die für die Verbindung zur Datenbank verwendet wird. Das geschieht einfach durch:

MYSQL *mysql;

Bevor wir eine Verbindung zur Datenbank herstellen können, muss die Variable mysql initialisiert werden. Dies geschieht durch den fogenden Aufruf:

mysql_init(MYSQL *mysql)

Jetzt wird die Funktion

```
MYSQL * STDCALL mysql_real_connect(MYSQL *mysql,
                                   const char *host,
                                   const char *user,
                                   const char *passwd,
                                   const char *db,
                                   unsigned int port,
```

```
                                const char *unix_socket,
                                unsigned int clientflag);
```

aufgerufen, um eine Verbindung zur Datenbank herzustellen. host ist der Hostname des MySQL−Servers, user der Benutzername, mit dem die Verbindung hergestellt werden soll, password das dazugehörige Passwort und db die Datenbank, zu der wir eine Verbindung wollen. port ist der TCP/IP Port des MySQL−Servers, unix_socket ist der Verbindungstyp und clientflag das Flag (Kennzeichen), das MySQL wie ODBC laufen lässt. In diesem Artikel ist es auf 0 gesetzt. Wenn die Verbindung erfolgreich hergestellt wurde, gibt diese Funktion 0 zurück.

Nun können wir uns mit der Datenbank verbinden und eine Abfrage machen:

```
char *query;
```

Mit diesem String können wir alle SQL−Sätze bilden und eine Abfrage machen. Die Funktion, mit der die Abfrage ausgeführt wird, ist:

```
int STDCALL mysql_real_query(MYSQL *mysql,
                             const char *q,
                             unsigned int length);
```

mysql ist die Variable, die wir schon weiter oben verwendet haben, q ist der String mit der SQL−Abfrage und length ist die Länge dieses Strings. Wenn die Abfrage ohne Fehlermeldung erfolgreich ist, gibt diese Funktion 0 zurück.

Nachdem wir die Abfrage gemacht haben, brauchen wir eine Variable von Typ MYSQL_RES, um die Ergebnisse verwenden zu können. Die folgende Zeile deklariert eine solche Variable:

```
MYSQL_RES *res;
```

Dann wird die Funktion

```
mysql_use_result(MYSQL *query)
```

verwendet, um die Ergebnisse zu lesen.

Obwohl wir sehr leicht Abfragen an die Datenbank machen können, brauchen wir noch ein paar andere Funktionen, um die Ergebnisse verwenden zu können. Die erste ist:

```
MYSQL_ROW STDCALL mysql_fetch_row(MYSQL_RES *result);
```

Diese Funktion wandelt die Ergebnisse in Zeilen um. Der Rückgabewert der Funktion ist vom Typ MYSQL_ROW. Die folgende Zeile deklariert solch eine Variable:

```
MYSQL_ROW row;
```

Wie schon weiter oben erklärt wurde, ist die Variable row ein Array von Strings. Das heißt, dass row[0] die erste Spalte der ersten Zeile enthält, row[1] die zweite Spalte der ersten Zeile, usw. Wenn wir mysql_fetch_row verwenden, bekommt die Variable row die Daten der nächsten Zeile des Ergebnisses zugewiesen. Wenn das Ende des Ergebnisses erreicht ist, gibt die Funktion einen negativen Wert zurück. Am Ende müssen wir die Verbindung zur Datenbank natürlich noch beenden. Dies geschieht mit:

```
mysql_close(MYSQL *mysql)
```

Einige hilfreiche Funktionen

Die folgende Funktion gibt die Anzahl der Felder in einer Datenbank zurück:

```
unsigned int STDCALL mysql_num_fields(MYSQL *mysql);
```

Um die Anzahl der Zeilen eines Abfrage−Ergebnisses zu erfahren, verwendet man:

my_ulonglong STDCALL mysql_num_rows(MYSQL_RES *res);

my_ulonglong STDCALL mysql_affected_rows(MYSQL *mysql);

Diese Funktion gibt die Anzahl der Zeilen zurück, die von einer INSERT, DELETE oder UPDATE Abfrage betroffen sind. Beachte, dass der Rückgabetyp der Funktion my_ulonglong ist.

Zu guter Letzt noch ein Beispiel:

```
#include <mysql/mysql.h>
#include <stdio.h>

void main(){
   MYSQL *mysql;
   MYSQL_RES *res;
   MYSQL_ROW row;
   char *query;
   int t,r;

   mysql_init(mysql);
   if (!mysql_real_connect(mysql,"localhost","mysql",
        "mysql","deneme",0,NULL,0))
   {
       printf( "Error connectin ot database: %s\n",mysql_error(mysql));
   }
   else printf("Connected...\n");

   query="select * from Deneme";

   t=mysql_real_query(mysql,query,(unsigned int) strlen(query));
   if (t)
   {
      printf("Error making query: %s\n",
               mysql_error(mysql));
   }
   else printf("Query made...\n");
   res=mysql_use_result(mysql);
   for(r=0;r<=mysql_field_count(mysql);r++){
           row=mysql_fetch_row(res);
           if(row<0) break;
           for(t=0;t<mysql_num_fields(res);t++){
                   printf("%s ",row[t]);
           }
           printf("\n");
   }
   mysql_close(mysql);
}
```

Empfohlene Literatur

- Die Homepage von MySQL: www.mysql.com
- Dokumentation, die bei den MySQL Quelltexten enthalten ist (wahrscheinlich im Verzeichnis /usr/doc).

2005−01−11, generated by lfparser_pdf version 2.51