LinuxFocus article number 304
http://linuxfocus.org

L’API C de MySQL

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

L´auteur:
J’utilise Linux depuis 1997. J’apprecie particulièrement la liberté offerte, sa flexibilité et la philosophie OpenSource.

Traduit en Français par:
Guillaume Baudot
<guillaume.baudot(at)caramail.com>

Résumé:

Nous allons apprendre dans cet article comment utiliser l’API C de MySQL, autrement dit, les fonctions C d’interface vers MySQL. Pour une bonne compréhension de l’article, il est recommandé d’avoir quelques connaissances préalables du langage C, en particulier :

- les variables;
- les fonctions;
- les pointeurs.

N.d.T. : des notions en bases de données, notamment SQL, ne sont pas non plus à négliger.

Introduction

L’API C est distribuée avec les sources de MySQL, dans la bibliothèque *mysqlclient*. Il s’agit d’un jeu de fonctions utilisées pour se connecter à une base de données et éxécuter différentes requêtes. Vous pourrez trouver quelques exemples dans le répertoire *clients* inclus dans les sources MySQL.

Les variables C/MySQL

Les variables décrites ci-dessous sont définies dans la bibliothèque MySQL et seront utilisées par les fonctions MySQL. Ne vous laissez pas impressionner par la profusion de détails, dans la mesure où vous n’aurez pas à les manipuler directement dans vos programmes.

MYSQL
Cette structure définit un gestionnaire de communication, qui permettra la connexion à une base de

données.

```
typedef struct st_mysql {
  NET           net;            /* Communication parameters - parametres de communic
  gptr          connector_fd;   /* ConnectorFd for SSL - descripteur de fichier (poi
  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 - identifant de conne
  my_ulonglong affected_rows;
  my_ulonglong insert_id;       /* id if insert on table with NEXTNR - identifiant u
  my_ulonglong extra_info;      /* Used by mysqlshow - utilise par la fonction mysql
  unsigned long packet_length;
  enum mysql_status status;
  MYSQL_FIELD   *fields;
  MEM_ROOT      field_alloc;
  my_bool       free_me;        /* If free in mysql_close - (deconnexion automatique
  my_bool       reconnect;      /* set to 1 if automatic reconnect - reconnexion aut
  struct st_mysql_options options;
  char          scramble_buff[9];
  struct charset_info_st *charset;
  unsigned int  server_language;
} MYSQL;
```

MYSQL_RES

Cette structure représente le résultat d'une requête de type SELECT (SELECT, SHOW, DESCRIBE, EXPLAIN). Nous emploierons le terme result-set (collection de résultats) pour évoquer de telles données.

```
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 - pour un lecture non bufferis
  MYSQL_ROW     current_row;    /* buffer to current row - memoire tampon pour la li
  unsigned long *lengths;       /* column lengths of current row - longueurs des col
  MYSQL         *handle;        /* for unbuffered reads - pour un lecture non buffer
  my_bool       eof;            /* Used my mysql_fetch_row - utilise par la fonction
} MYSQL_RES;
```

MYSQL_ROW

C'est une représentation d'une ligne d'un result-set, qui est indépendante du type des données elles-m&eirc;me. Il est important de noter qu'en dépit de la définition même de ce type, à savoir un tableau de chaînes de caractères, on ne peut toutefois en manipuler les champs comme des chaînes classsiques : en effet, il peut s'agir de données binaires, auquel cas le caractère NULL (fin de chaîne) est susceptible d'apparaître à plusieurs reprises.

```
typedef char **MYSQL_ROW;
```

MYSQL_FIELD
Cette structure contient différentes informations relatives à un champ d'un result-set (nom, type, taille...). Ces mêmes informations permettent d'interpréter correctement le contenu du champ, en l'occurence, un élément de MYSQL_ROW.

```
typedef struct st_mysql_field {
  char *name;                 /* Name of column - nom de la colonne */
  char *table;                /* Table of column if column was a field - pointeur
  char *def;                  /* Default value (set by mysql_list_fields) - valeur
  enum enum_field_types type;   /* Type of field. Se mysql_com.h for types - type du
  unsigned int length;        /* Width of column - taille */
  unsigned int max_length;    /* Max width of selected set - taille maximale */
  unsigned int flags;         /* Div flags - divers marqueurs */
  unsigned int decimals;      /* Number of decimals in field - nombre de decimales
} MYSQL_FIELD;
```

my_ulonglong
Ce type est utilisé par les fonctions renvoyant un nombre de lignes, à savoir mysql_num_rows, mysql_insert_id et peut contenir des valeurs comprises entre 0 et 1.84e19. Sur certains systèmes, l'affichage de ce type peut provoquer une erreur, auquel cas il faut convertir le nombre au format *unsigned long* et utiliser le formatage de chaîne ad hoc.
Exemple :

```
    printf("Nombre de lignes : %lu\n", (unsigned long)mysql_num_rows(result));
```

```
typedef unsigned long my_ulonglong;
```

La connection au serveur MySQL et les requêtes

À partir d'ici, nous supposerons que nous disposons d'un serveur MySQL opérationnel, une base de données contenant une table au moins, ainsi qu'un compte utilisateur. Si cela pose problème, veuillez vous référer au site www.mysql.com et consulter la documentation.

Comme indiqué plus haut, l'API se trouve dans la bibliothèque *mysqlclient*. C'est pourquoi il nous faut l'indiquer au compilateur par le biais de l'option *-lmysqlclient*. De même, il faut inclure dans votre source le fichier d'entête MySQL (le plus souvent dans */usr/include/mysql*). Ce dernier contient les déclarations des types et fonctions et, selon la version dont vous diposez, vous pourrez éventuellement constater quelques différences d'implémentation.

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

Commençons par définir les quelques variables dont nous aurons besoin.

```
MYSQL *mysql;                                   /* pointeur vers notre gestionnaire
char *query;                                    /* chaine destinee a contenir une re
MYSQL_RES *res;                 /* pointeur vers un result-set */
MYSQL_ROW row;                          /* objet destine a accueillir une ligne d'un
```

Il nous faut ensuite initialiser notre gestionnaire de connexion.

```
MYSQL * mysql_init(MYSQL *mysql);
```

Voilà, tout est désormais en place pour établir la connexion.

```
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);
```

Comme vous pouvez le remarquer, la fonction de connexion nécessite un certain nombre de paramètres. Vous noterez en outre qu'elle retourne la valeur NULL en cas d'échec de la connexion.

- *mysql* : notre gestionnaire de connexion
- *host* : le serveur MySQL
- *user* : login du compte MySQL
- *passwd* : mot de passe propre à *user*
- *db* : la base de données à laquelle se connecter
- *port* : numéro du port TCP/IP du serveur MySQL (0 pour utiliser le port par défaut)
- *unix_socket* : type de la connexion (NULL par défaut)
- *clientflag* : permet entre autres de faire fonctionner MySQL dans différents modes, dont ODBC (mode normal : 0)

Si la connexion s'est correctement établie, nous sommes donc en mesure d'éxécuter une requête.

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

Cette fonction retourne 0, ou un code d'erreur. Et pour les paramètres :

- *mysql* : notre gestionnaire de connexion
- *q* : la requête SQL
- *length* : la longueur de la précédente chaîne

Si la requête s'est proprement déroulée, il nous faut encore en exploiter le résultat. Nous utiliserons pour cela les deux fonctions suivantes. La premiÃ¨re permet d'initialiser un result-set et la seconde de parcourir ce dernier ligne par ligne.

```
MYSQL_RES *     STDCALL mysql_use_result(MYSQL *query);
MYSQL_ROW STDCALL mysql_fetch_row(MYSQL_RES *result);
```

La fonction *mysql_fetch_row* renvoie une valeur négative lorsqu'on atteint la fin du result-set. Et pour récupérer nos données, connaissant le type MYSQL_ROW, nous trouverons le premier élément dans row[0], le second dans row[1], etc...
Il ne nous reste plus qu'à fermer la connexion pour finir proprement le programme.

```
void mysql_close(MYSQL *mysql);
```

Voici encore, en vrac quelques fonctions complémentaires dont vous trouverez certainement l'utilité dans vos programmes.

- Pour obtenir le nombre de lignes d'un result-set :

  ```
  my_ulonglong STDCALL mysql_num_rows(MYSQL_RES *res);
  ```

- Pour obtenir le nombre de colonnes d'un result-set :

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

- Pour connaître le nombre de lignes affectées par une requête de type INSERT, UPDATE, ou DELETE (d'où l'explication plus haut du type *my_ulonglong*) :

  ```
  my_ulonglong STDCALL mysql_affected_rows(MYSQL *mysql);
  ```

- Pour libérer la mémoire allouée à un result-set :

  ```
  void mysql_free_result(MYSQL_RES *result);
  ```

La mise en pratique

Forts de ces nouvelles connaissances, les doigts nous brûlent d’ affronter le compilateur... Mais pour s’assurer d’avoir tout bien saisi, mieux vaut d’abord s’appuyer sur un exemple. Ainsi, pourquoi ne pas vous inspirer de ce code ?

```
#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);
}
```

N.d.T. : Ce programme n’a pas fonctionné tel quel chez moi, probablement parce que j’utilise une autre version de MySQL que l’auteur. Je me suis donc permis de l’adapter à mes besoins. Voici donc un second exemple qui, tenant fortement du plagiat, constitue donc une alternative.

Conseils de lecture

Nous n’avons fait dans cet article qu’aborder sommairement le sujet, pour dire comme il est vaste. Je ne saurais donc trop recommander aux plus curieux de consulter le site WEB MySQL et la documentation dissitribuée avec MySQL (normalement /usr/doc/mysql/...).

Site Web maintenu par l´équipe d´édition
LinuxFocus

http://www.LinuxFocus.org

Translation information:
tr --> -- : Özcan Güngör <ozcangungor(at)netscape.net>
tr --> en: Özcan Güngör <ozcangungor(at)netscape.net>
en --> fr: Guillaume Baudot
<guillaume.baudot(at)caramail.com>

2005-01-14, generated by lfparser_pdf version 2.51