Usando MySQL na linguagem C

Autor: Ricardo Rodrigues Lucca <rlucca at gmail.com>
Data: 07/08/2003

Introdução

Este é um artigo que visa demonstrar como podemos usar a *linguagem C* para realizar as ações básicas em um *banco de dados MySQL*. As ações básicas num banco de dados MySQL são:

- inserção;
- remoção;
- alteração e;
- consulta de dados

Assim, seria recomendável possuir alguma experiência na linguagem SQL.

Inicie, criando um banco de dados chamado "teste" e crie um usuário para acessar esse BD. No caso aqui criarei um usuário chamado "guest" com a senha "guest", depois criaremos uma tabela chamada "aprendendo". Abaixo estão os comandos para você fazer isso no console ou no rxvt (se quiser usar o gráfico):

```
01 $ mysql --user=root -p
02 Enter password: <senha aqui>
03 Welcome to the MySQL monitor. Commands end with ; or \g.
04 Your MySQL connection id is 8 to server version: 3.23.51-log
05 Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
06 mysql> create database teste;
07 Query OK, 1 row affected (0.00 sec)
08 mysql> grant all on teste.* to guest identified by "guest";
09 Query OK, 0 rows affected (0.06 sec)
10 mysql> flush privileges;
11 Query OK, 0 rows affected (0.03 sec)
12 mysql> exit
13 Bye
```

Agora vamos comentar isso. Na linha 01, você pede para logar com o usuário administrador do MySQL, que no caso é chamado *root* (--user=root), mas poderia ser outro. Fora isso, temos também que dizer que para esse usuário será preciso *digitar uma senha* (-p). Se você não sabe qual é esse usuário, me desculpe, mas consulte a pessoa que instalou o MySQL na

sua máquina.

Agora, na linha 06 pedimos para criarmos o banco de dados teste e na linha 08 dizemos que o *usuário guest* (to guest) com *senha guest* (identified by "guest") terá todos os direitos no banco de dados teste (grant all on teste.*). Na linha 10, fazemos valer o novo usuário. Assim, só depois desse comando o usuário vai funcionar e *exit* é para sair (linha 12). Agora você me pergunta: "E a tabela?". Criaremos a tabela usando o usuário que possui direitos nela para irmos nos acostumando a isso! Vamos lá:

```
$ mysql --user=guest -p
Enter password: <senha aqui>
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 38 to server version: 3.23.51-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> connect teste;
Connection id:    39
Current database: teste

mysql> create table aprendendo (
   -> `ID` INT NOT NULL AUTO_INCREMENT,
   -> `nome` VARCHAR( 40 ) NOT NULL,
   -> `sexo` CHAR( 1 ) NOT NULL,
   -> unique ( `ID` )
   -> );
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye
```

Isso faz com que você reconecte com o banco de dados teste (connect teste) e depois crie uma tabela chamada aprendendo (create table aprendendo), que possui três colunas:

- ID: que receberá um inteiro incrementado automaticamente;
- nome: que será do tipo VARCHAR que não poderá ser vazio;
- sexo: que será do tipo CHAR, também não podendo ser vazio.

Por fim, definimos uma chave única para a tabela.

Um pequeno comentário quanto aos tipos VARCHAR e CHAR. VARCHAR é usado quando queremos uma economia de espaço nas tabelas. Essa economia é gerada da seguinte forma: podemos inserir no máximo 40 caracteres, sendo que se inserirmos menos, vamos ter uma string da quantidade inserida, ao contrário do tipo CHAR, pois o tipo CHAR sempre vai ter o tipo definido. Isso dá um ganho de velocidade na hora da busca. Se definirmos um CHAR de no máximo 5 e inserirmos 3 caracteres, ele ocupará espaço como se tivesse cinco caracteres.

Iniciando na programação

Vamos começar pelo começo. Para isso precisamos ensinar como inicializar a variável de conexão e depois como conectar. No header *mysql.h* temos algumas funções interessantes:

- MYSQL * mysql_init(MYSQL *mysql);
- MYSQL * mysql_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd);
- MYSQL * 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);
- void mysql_close(MYSQL *sock);

Agora, vamos analisar o que elas fazem. Na primeira função (mysql_init) temos um tipo chamado MYSQL, esse tipo traz muitos dados referentes ao que se está fazendo e seria muito baixo nível se tivéssemos que setar eles manualmente, para isso temos funções que fazem esse serviço por nós e fazem com que não nos preocupamos com esses detalhes.

A função mysql_init inicializa uma variável do tipo MYSQL para ser usada.
Exemplo de uso:

mysql_init(&conexao);

Agora, as outras duas (*connect) tem funções parecidas, ambas servem para conectar num banco de dados, mas a *mysql_real_connect* serve para conectar num banco de dados e já escolher qual banco de dados queremos usar. Isto é, o uso da função *mysql_select_db* virá desnecessário, por isso usarei essa função nos exemplos. Um exemplo de uso seria:

mysql_real_connect(&conexao, "localhost", "guest", "guest", "teste", 0, NULL, 0);

NOTA: Repare nos três últimos parâmetros, vendo os dados da função um pouco mais pra cima vamos descobrir que eles são referentes à porta usada, socket e flags do client. O uso desses parâmetros faz com que seja "automático", ou seja, o sistema operacional escolhe a melhor porta e o soquete que melhor se adapter.,

NOTA2: Quanto ao host, informamos "localhost" apenas para deixar mais claro que estamos falando com a máquina em que estamos. Se quiséssemos usar NULL daria no mesmo.

A última função é o que faz o fechamento de uma conexão. O MySQL sabe qual conexão fechar, pois nós passamos como parâmetro e a variável que guarda esses dados.

Programa exemplo:

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

void main(void)
```

```
{
  MYSQL conexao;

  mysql_init(&conexao);
  mysql_real_connect(&conexao, "localhost", "guest", "guest", "teste", 0, NULL, 0);
  printf("conectado com sucesso!\n");
  mysql_close(&conexao);
}
```

Um exemplo agora realizando testes para ver se realmente conectou!

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

void main(void)
{
    MYSQL conexao;

    mysql_init(&conexao);
    if ( mysql_real_connect(&conexao, "localhost", "guest", "guest", "teste", 0, NULL, 0) )
    {
        printf("conectado com sucesso!\n");
        mysql_close(&conexao);
     }
     else
     {
        printf("Falha de conexao\n");
        printf("Erro %d : %s\n", mysql_errno(&conexao), mysql_error(&conexao));
     }
}
```

Vamos falar sobre as duas funções novas:

- unsigned int mysql_errno(MYSQL *mysql);
- char * mysql_error(MYSQL *mysql);

A primeira retorna um número inteiro não sinalizado, isto é, retorna o código do erro que aconteceu.

Já a segunda retorna o erro por "extenso". Altere a senha do usuário para uma não válida para ter uma idéia do que ocorre.

Inserção

Geralmente as coisas que você faz no MySQL é perguntar para ele algo. Para isso criamos

"queries". Para executar uma QUERY temos a seguinte função:

- int mysql_query(MYSQL *mysql, const char *q);

Como podemos ver, essa função recebe a variável de conexão e espera também uma variável do tipo char. Aqui é onde entra a parte que você já deve conhecer do SQL. Passaremos as queries para a função e esperaremos os resultados! Infelizmente, isso não é que nem em PHP onde você fica abstraido dos tipos que o mysql lida, mas isso você já deve ter notado. Vamos por a mão na massa!

Inserção

mysql_query(&conexao, "insert into aprendendo(nome, sexo) values('Ricardo Rodrigues Lucca', 'M');

Para inserirmos dados na tabela criada precisamos uma query como a dada acima. Como vimos, na variável do tipo char que é passada como parâmetro, informamos que estamos querendo inserir algo (insert into) em algum lugar (aprendendo(nome, sexo)) com os valores X. Se quiséssemos escrever no nome "Moro na rua 24 de julho. E dane-se o mundo!", teríamos que fazer da seguinte forma a pesquisa:

"insert into aprendendo(nome, sexo) values ('Moro na rua 24 de julho. E dane-se o mundo', '!');"

A função ficaria:

mysql_query(&conexao, "insert into aprendendo(nome, sexo) values ('Moro na rua 24 de julho. E dane-se o mundo', '!');"

Sexo não poderia ser em branco, pois ao criar a tabela informamos que ele é "NOT NULL", que faz com que não aceite inserções vazias na tabela, por isso precisamos inserir algo nele. Assim sendo, ele é do tipo char e aceita qualquer caracter.

Um exemplo do que foi visto até agora:

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

void main(void)
{
   MYSQL conexao;

   mysql_init(&conexao);
   if ( mysql_real_connect(&conexao, "localhost", "guest", "guest", "teste", 0, NULL, 0) )
```

```
    {
       printf("conectado com sucesso!\n");

       mysql_query(&conexao,"INSERT INTO aprendendo(nome, sexo) values('Ricardo
Rodrigues Lucca', 'M');");

       mysql_close(&conexao);
    }
    else
    {
       printf("Falha de conexao\n");
       printf("Erro %d : %s\n", mysql_errno(&conexao), mysql_error(&conexao));
    }
}
```

O Retorno do mysql_query

Até agora falamos como inserir na tabela às "cegas". Se quisermos ter certeza que não houve erro durante a inserção, como proceder? Simples! Faça com que uma variável inteira receba o valor de retorno da *mysql_query* e depois teste a seu valor. Sempre que a variável inteira for ZERO é porque não houve erro na query. Logo, sempre que ela for diferente de zero ela estará sinalizando que há erros na função mysql_query.

Exemplo de um programa usando tudo isso:

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

void main(void)
{
    MYSQL conexao;
    int res;

    mysql_init(&conexao);
    if ( mysql_real_connect(&conexao, "localhost", "guest", "guest", "teste", 0, NULL, 0) )
    {
      printf("conectado com sucesso!\n");

      res = mysql_query(&conexao,"INSERT INTO aprendendo(nome, sexo)
values('Ricardo  Rodrigues Lucca', 'M');");

      if (!res) printf("Registros inseridos %d\n", mysql_affected_rows(&conexao));
      else printf("Erro na inserção %d : %s\n", mysql_errno(&conexao),
```

```
mysql_error(&conexao));

      mysql_close(&conexao);
   }
   else
   {
      printf("Falha de conexao\n");
      printf("Erro %d : %s\n", mysql_errno(&conexao), mysql_error(&conexao));
    }
}
```

Consulta

Para desenvolver um programa de consulta num banco de dados MySQL iremos introduzir mais quatro funções:

- unsigned int mysql_num_fields(MYSQL_RES *res);
- MYSQL_FIELD * mysql_fetch_fields(MYSQL_RES *res);
- MYSQL_ROW mysql_fetch_row(MYSQL_RES *result);
- void mysql_free_result(MYSQL_RES *result);

Vamos começar falando da primeira função, ela serve para sabermos quantos campos existem na tabela em que foi feita a "query". A estrutura que ela recebe como parâmetro é a retornada por *mysql_store_result*.

A função seguinte (*mysql_fetch_fields*) tem como retorno uma estrutura que é um array. Essa estrutura possui alguns dados muito úteis, como o nome da coluna.

A *mysql_fetch_row* retorna os registros encontrados e como cada registro tem várias colunas o valor char que é retornado é um vetor sendo que cada "posição" corresponde a uma coluna da tabela.

Por fim, a que resta ser explicada (*mysql_free_result*) serve para limpar a variável de resultado retornado por mysql_store_result.

Vejamos, um exemplo para consultar com as funções citadas.

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

#define HOST "localhost"
#define USER "guest"
#define PASS "guest"
#define DB "teste"

int main(void)
```

```
{
  MYSQL conexao;
  MYSQL_RES *resp;
  MYSQL_ROW linhas;
  MYSQL_FIELD *campos;
  char query[]="SELECT * FROM aprendendo;";
  int conta; //Contador comum

  mysql_init(&conexao);
  if (mysql_real_connect(&conexao,HOST,USER,PASS,DB,0,NULL,0))
  {
    printf("Conectado com Sucesso!\n");
    if (mysql_query(&conexao,query))
      printf("Erro: %s\n",mysql_error(&conexao));
    else
    {
      resp = mysql_store_result(&conexao);//recebe a consulta
     if (resp) //se houver consulta
     {
       //passa os dados dos campos para a variável campos
       //escreve na tela os nomes dos campos dando
       //um tab somente
       campos = mysql_fetch_fields(resp);
       for (conta=0;conta<mysql_num_fields(resp);conta++) {
         printf("%s",(campos[conta]).name);
         if (mysql_num_fields(resp)>1)
            printf("\t");
         }

         printf("\n");

         //enquanto retonrnar registros, conta até o
         //número de colunas que a tabela tem e escreve na
         //tela com um tab, depois pula a linha e tenta
         //pegar outro registro
         while ((linhas=mysql_fetch_row(resp)) != NULL)
         {
           for (conta=0;conta<mysql_num_fields(resp);conta++)
             printf("%s\t",linhas[conta]);
           printf("\n");
         }
      }
      mysql_free_result(resp);//limpa a variável do resultado: resp
     }
     mysql_close(&conexao);
  }
  else
```

```
  {
    printf("Conexao Falhou\n");
    if (mysql_errno(&conexao))
      printf("Erro %d : %s\n", mysql_errno(&conexao), mysql_error(&conexao));
  }

  return 0;
}
```

Nesse programa podemos ver um exemplo que considero básico de consulta. Nele também temos macros para facilitar mudar, por exemplo, o servidor de conexão do MySQL. Execute o programa e você deverá ver algo como:

```
$ ./a.out
Conectado com Sucesso!
ID     nome    sexo
1      Ricardo Rodrigues Lucca M
```

Remoção e alteração

Para fazer remoção e alteração num conteúdo de uma tabela é muito simples! Se baseie no programa em que usamos para inserir na tabela. Tanto para remover como para altear a tabela precisamos apenas modificar a query, isso mesmo, podemos fazer a alteração e remoção de dados na tabela só mudando a query daquele programa!

Se você não sabe como fazer isso, vá no www.google.com e pesquise sobre o assunto!

Erro de compilação

Você está tentando compilar os programas exemplos daqui e está encontrando erros esquisitos, como o encontrado a baixo?

```
$ gcc consulta.c
/tmp/ccapYor5.o: In function `main':
/tmp/ccapYor5.o(.text+0x31): undefined reference to `mysql_init'
/tmp/ccapYor5.o(.text+0x5a): undefined reference to `mysql_real_connect'
/tmp/ccapYor5.o(.text+0x8d): undefined reference to `mysql_query'
/tmp/ccapYor5.o(.text+0xa8): undefined reference to `mysql_error'
/tmp/ccapYor5.o(.text+0xdb): undefined reference to `mysql_store_result'
/tmp/ccapYor5.o(.text+0x102): undefined reference to `mysql_fetch_fields'
/tmp/ccapYor5.o(.text+0x12b): undefined reference to `mysql_num_fields'
/tmp/ccapYor5.o(.text+0x172): undefined reference to `mysql_num_fields'
/tmp/ccapYor5.o(.text+0x1b3): undefined reference to `mysql_fetch_row'
```

```
/tmp/ccapYor5.o(.text+0x1eb): undefined reference to `mysql_num_fields'
/tmp/ccapYor5.o(.text+0x250): undefined reference to `mysql_errno'
/tmp/ccapYor5.o(.text+0x268): undefined reference to `mysql_error'
/tmp/ccapYor5.o(.text+0x27d): undefined reference to `mysql_errno'
/tmp/ccapYor5.o(.text+0x2a5): undefined reference to `mysql_free_result'
/tmp/ccapYor5.o(.text+0x2b7): undefined reference to `mysql_close'
/tmp/ccapYor5.o(.text+0x2e1): undefined reference to `mysql_errno'
/tmp/ccapYor5.o(.text+0x2f9): unefined reference to `mysql_error'
/tmp/ccapYor5.o(.text+0x30e): undefined reference to `mysql_errno'
collect2: ld returned 1 exit status
```

Pois é! Essa quantidade absurda de erros é normal quando se esquecem de linkar o programa com a biblioteca *mysqlclient*! O mesmo programa compilado novamente não apresentará nenhum erro se ele for linkado corretamente, como no exemplo abaixo:

$ gcc consulta.c -lmysqlclient

Espero que isso ajude bastante, pois esquecer de linkar com essa biblioteca é um dos erros mais comuns e pode assustar bastante um cara que esteja iniciando em programação!

---

http://www.vivaolinux.com.br/artigo/Usando-MySQL-na-linguagem-C

Voltar para o site