# MySQL C API programming

*last modified September 23, 2020*

This is a C programming tutorial for the MySQL database. It covers the basics of MySQL programming with the C API. You may also consider to look at the MySQL tutorial on ZetCode.

## About MySQL database

MySQL is a leading open source database management system. It is a multi user, multithreaded database management system. MySQL is especially popular on the web. MySQL currently owned by Oracle. MySQL database is available on most important OS platforms. It runs on BSD Unix, Linux, Windows, or Mac OS.

MariaDB is a community-developed, commercially supported fork of the MySQL relational database management system

```
$ sudo apt install default-libmysqlclient-dev
```

To be able to compile C examples, we need to install the MySQL C development libraries. The above line shows how we can do it on Debian based Linux.

## C99

This tutorial uses C99. For GNU C compiler, we need to add the -std=c99 option. For Windows users, the Pelles C IDE is highly recommended. (MSVC does not support C99.)

```
MYSQL *con = mysql_init(NULL);
```

In C99, we can mix declarations with code. In older C programs, we would need to separate this line into two lines.

## MySQL C first example

In the first example, we test one MySQL function call.

**version.c**

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

int main(int argc, char **argv)
{
  printf("MySQL client version: %s\n", mysql_get_client_info());

  exit(0);
}
```

The mysql_get_client_info() shows the MySQL client version.

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

We include necessary header files.

```
printf("MySQL client version: %s\n", mysql_get_client_info());
```

This code line outputs the version of the MySQL client. For this, we use the mysql_get_client_info() function call.

```
exit(0);
```

We exit from the script.

```
$ c99 version.c -o version  `mysql_config --cflags --libs`
```

Here is how we compile the code example.

```
$ ./version
MySQL client version: 10.3.24
```

This is the output.

## MySQL C create database

The next code example will create a database. The code example can be divided into these parts:

- Initiation of a connection handle structure
- Creation of a connection
- Execution of a query
- Closing of the connection

**createdb.c**

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

int main(int argc, char **argv)
{
  MYSQL *con = mysql_init(NULL);

  if (con == NULL)
  {
      fprintf(stderr, "%s\n", mysql_error(con));
      exit(1);
  }

  if (mysql_real_connect(con, "localhost", "root", "root_passwd",
          NULL, 0, NULL, 0) == NULL)
  {
      fprintf(stderr, "%s\n", mysql_error(con));
      mysql_close(con);
      exit(1);
  }

  if (mysql_query(con, "CREATE DATABASE testdb"))
  {
      fprintf(stderr, "%s\n", mysql_error(con));
      mysql_close(con);
      exit(1);
  }

  mysql_close(con);
  exit(0);
}
```

The code example connects to the MySQL database system and creates a new database called testdb.

```
MYSQL *con = mysql_init(NULL);
```

The mysql_init() function allocates or initialises a MYSQL object suitable for mysql_real_connect() function. Remember this is C99.

```
if (con == NULL)
{
    fprintf(stderr, "%s\n", mysql_error(con));
    exit(1);
}
```

We check the return value. If the mysql_init() function fails, we print the error message and terminate the application.

```
if (mysql_real_connect(con, "localhost", "root", "root_passwd",
    NULL, 0, NULL, 0) == NULL)
{
    fprintf(stderr, "%s\n", mysql_error(con));
    mysql_close(con);
    exit(1);
}
```

The mysql_real_connect() function establishes a connection to the database. We provide connection handler, host name, user name and password parameters to the function. The other four parameters are the database name, port number, unix socket and finally the client flag. We need superuser priviliges to create a new database.

```
if (mysql_query(con, "CREATE DATABASE testdb"))
{
    fprintf(stderr, "%s\n", mysql_error(con));
    mysql_close(con);
    exit(1);
}
```

The mysql_query() executes the SQL statement. In our case, the statement creates a new database.

```
 mysql_close(con);
```

Finally, we close the database connection.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| testdb             |
+--------------------+
3 rows in set (0.00 sec)
```

This is the proof that the database was created.

# MySQL C create and populate table

Before we create a new table, we create a user that we will be using in the rest of the tutorial.

```
mysql> CREATE USER user12@localhost IDENTIFIED BY '34klq*';
```

We have created a new user user12.

```
mysql> GRANT ALL ON testdb.* to user12@localhost;
```

Here we grant all priviliges to user12 on testdb database.

The next code example creates a table and inserts some data into it.

**create_populate.c**

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

void finish_with_error(MYSQL *con)
{
  fprintf(stderr, "%s\n", mysql_error(con));
  mysql_close(con);
  exit(1);
}

int main(int argc, char **argv)
{
  MYSQL *con = mysql_init(NULL);

  if (con == NULL)
  {
      fprintf(stderr, "%s\n", mysql_error(con));
      exit(1);
  }

  if (mysql_real_connect(con, "localhost", "user12", "34klq*",
          "testdb", 0, NULL, 0) == NULL)
  {
      finish_with_error(con);
  }

  if (mysql_query(con, "DROP TABLE IF EXISTS cars")) {
      finish_with_error(con);
  }

  if (mysql_query(con, "CREATE TABLE cars(id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255),
      finish_with_error(con);
  }

  if (mysql_query(con, "INSERT INTO cars VALUES(1,'Audi',52642)")) {
      finish_with_error(con);
  }

  if (mysql_query(con, "INSERT INTO cars VALUES(2,'Mercedes',57127)")) {
      finish_with_error(con);
  }

  if (mysql_query(con, "INSERT INTO cars VALUES(3,'Skoda',9000)")) {
      finish_with_error(con);
  }

  if (mysql_query(con, "INSERT INTO cars VALUES(4,'Volvo',29000)")) {
      finish_with_error(con);
  }

  if (mysql_query(con, "INSERT INTO cars VALUES(5,'Bentley',350000)")) {
      finish_with_error(con);
  }

  if (mysql_query(con, "INSERT INTO cars VALUES(6,'Citroen',21000)")) {
      finish_with_error(con);
  }
```

```
    if (mysql_query(con, "INSERT INTO cars VALUES(7,'Hummer',41400)")) {
        finish_with_error(con);
    }

    if (mysql_query(con, "INSERT INTO cars VALUES(8,'Volkswagen',21600)")) {
        finish_with_error(con);
    }

    mysql_close(con);
    exit(0);
}
```

We don't use any new MySQL function call here. We use mysql_query() function call to both create a table and insert data into it.

```
void finish_with_error(MYSQL *con)
{
  fprintf(stderr, "%s\n", mysql_error(con));
  mysql_close(con);
  exit(1);
}
```

In order to avoid unnecessary repetition, we create a custom finish_with_error() function.

```
if (mysql_real_connect(con, "localhost", "user12", "34klq*",
        "testdb", 0, NULL, 0) == NULL)
{
    finish_with_error(con);
}
```

We connect to testdb database. The user name is user12 and password is 34klq*. The fifth parameter is the database name.

```
if (mysql_query(con, "CREATE TABLE cars(id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255), p
    finish_with_error(con);
}
```

Here we create a table named cars. It has three columns.

```
if (mysql_query(con, "INSERT INTO cars VALUES(1,'Audi',52642)")) {
    finish_with_error(con);
}
```

We insert one row into the cars table.

```
mysql> USE testdb;
mysql> SHOW TABLES;
+------------------+
| Tables_in_testdb |
+------------------+
| cars             |
+------------------+
1 row in set (0.00 sec)
```

We show tables in the database.

```
mysql> SELECT * FROM cars;
+------+------------+--------+
| Id   | Name       | Price  |
+------+------------+--------+
|    1 | Audi       |  52642 |
|    2 | Mercedes   |  57127 |
|    3 | Skoda      |   9000 |
|    4 | Volvo      |  29000 |
|    5 | Bentley    | 350000 |
|    6 | Citroen    |  21000 |
|    7 | Hummer     |  41400 |
|    8 | Volkswagen |  21600 |
+------+------------+--------+
8 rows in set (0.00 sec)
```

We select all data from the table.

## MySQL C retrieve data

In the next example, we retrieve data from a table.

We need to do the following steps:

- Create a connection
- Execute query
- Get the result set
- Fetch all available rows

- Free the result set

**retrieva_data.c**

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

void finish_with_error(MYSQL *con)
{
  fprintf(stderr, "%s\n", mysql_error(con));
  mysql_close(con);
  exit(1);
}

int main(int argc, char **argv)
{
  MYSQL *con = mysql_init(NULL);

  if (con == NULL)
  {
      fprintf(stderr, "mysql_init() failed\n");
      exit(1);
  }

  if (mysql_real_connect(con, "localhost", "user12", "34klq*",
          "testdb", 0, NULL, 0) == NULL)
  {
      finish_with_error(con);
  }

  if (mysql_query(con, "SELECT * FROM cars"))
  {
      finish_with_error(con);
  }

  MYSQL_RES *result = mysql_store_result(con);

  if (result == NULL)
  {
      finish_with_error(con);
  }

  int num_fields = mysql_num_fields(result);

  MYSQL_ROW row;

  while ((row = mysql_fetch_row(result)))
  {
      for(int i = 0; i < num_fields; i++)
      {
          printf("%s ", row[i] ? row[i] : "NULL");
      }

      printf("\n");
  }

  mysql_free_result(result);
  mysql_close(con);

  exit(0);
}
```

The example shows all rows from the cars table.

```
if (mysql_query(con, "SELECT * FROM cars"))
{
    finish_with_error(con);
}
```

We execute the query that retrieves all data from the cars table.

```
MYSQL_RES *result = mysql_store_result(con);
```

We get the result set using the mysql_store_result() function. The MYSQL_RES is a structure for holding a result set.

```
int num_fields = mysql_num_fields(result);
```

We get the number of fields (columns) in the table.

```
MYSQL_ROW row;

while ((row = mysql_fetch_row(result)))
{
    for(int i = 0; i < num_fields; i++)
    {
        printf("%s ", row[i] ? row[i] : "NULL");
    }
```

```
    printf("\n");
}
```

We fetch the rows and print them to the screen.

```
mysql_free_result(result);
mysql_close(con);
```

We free the resources.

```
$ ./retrieva_data
1 Audi 52642
2 Mercedes 57127
3 Skoda 9000
4 Volvo 29000
5 Bentley 350000
6 Citroen 21000
7 Hummer 41400
8 Volkswagen 21600
```

This is the output.

## MySQL C last inserted row id

Sometimes, we need to determine the id of the last inserted row. We can determine the last inserted row id by calling the mysql_insert_id() function. The function only works if we have defined an AUTO_INCREMENT column in the table.

**last_row_id.c**

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

void finish_with_error(MYSQL *con)
{
  fprintf(stderr, "%s\n", mysql_error(con));
  mysql_close(con);
  exit(1);
}

int main(int argc, char **argv)
{
  MYSQL *con = mysql_init(NULL);

  if (con == NULL)
  {
      fprintf(stderr, "mysql_init() failed\n");
      exit(1);
  }

  if (mysql_real_connect(con, "localhost", "user12", "34klq*",
          "testdb", 0, NULL, 0) == NULL)
  {
      finish_with_error(con);
  }

  if (mysql_query(con, "DROP TABLE IF EXISTS writers"))
  {
      finish_with_error(con);
  }

  char *sql = "CREATE TABLE writers(id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255))";

  if (mysql_query(con, sql))
  {
      finish_with_error(con);
  }

  if (mysql_query(con, "INSERT INTO writers(name) VALUES('Leo Tolstoy')"))
  {
      finish_with_error(con);
  }

  if (mysql_query(con, "INSERT INTO writers(name) VALUES('Jack London')"))
  {
      finish_with_error(con);
  }

  if (mysql_query(con, "INSERT INTO writers(name) VALUES('Honore de Balzac')"))
  {
      finish_with_error(con);
  }

  int id = mysql_insert_id(con);

  printf("The last inserted row id is: %d\n", id);

  mysql_close(con);
  exit(0);
}
```

A new table is created. Three rows are inserted into the table. We determine the last inserted row id.

```
char *sql = "CREATE TABLE writers(id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255))";
```

The id column has an AUTO_INCREMENT type.

```
int id = mysql_insert_id(con);
```

The mysql_insert_id() function returns the value generated for an AUTO_INCREMENT column by the previous INSERT or UPDATE statement.

```
$ ./last_row_id
The last inserted row id is: 3
```

This is the output.

## MySQL C column headers

In the next example, we retrieve data from the table and its column names.

**headers.c**

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

void finish_with_error(MYSQL *con)
{
  fprintf(stderr, "%s\n", mysql_error(con));
  mysql_close(con);
  exit(1);
}

int main(int argc, char **argv)
{
  MYSQL *con = mysql_init(NULL);

  if (con == NULL)
  {
      fprintf(stderr, "mysql_init() failed\n");
      exit(1);
  }

  if (mysql_real_connect(con, "localhost", "user12", "34klq*",
          "testdb", 0, NULL, 0) == NULL)
  {
      finish_with_error(con);
  }

  if (mysql_query(con, "SELECT * FROM cars LIMIT 3"))
  {
      finish_with_error(con);
  }

  MYSQL_RES *result = mysql_store_result(con);

  if (result == NULL)
  {
      finish_with_error(con);
  }

  int num_fields = mysql_num_fields(result);

  MYSQL_ROW row;
  MYSQL_FIELD *field;

  while ((row = mysql_fetch_row(result)))
  {
      for(int i = 0; i < num_fields; i++)
      {
          if (i == 0)
          {
             while(field = mysql_fetch_field(result))
             {
                printf("%s ", field->name);
             }

             printf("\n");
          }

          printf("%s  ", row[i] ? row[i] : "NULL");
      }
  }

  printf("\n");

  mysql_free_result(result);
  mysql_close(con);
```

```
  exit(0);
}
```

We print the first three rows from the cars table. We also include the column headers.

```
MYSQL_FIELD *field;
```

The MYSQL_FIELD structure contains information about a field, such as the field's name, type and size. Field values are not part of this structure; they are contained in the MYSQL_ROW structure.

```
if (i == 0)
{
    while(field = mysql_fetch_field(result))
    {
        printf("%s ", field->name);
    }

    printf("\n");
}
```

The first row contains the column headers. The mysql_fetch_field() call returns a MYSQL_FIELD structure. We get the column header names from this structure.

```
$ ./headers
id name price
1  Audi  52642
2  Mercedes  57127
3  Skoda  9000
```

This is the output of our program.

## MySQL C multiple statements

It is possible to execute multiple SQL statements in one query. We must set the CLIENT_MULTI_STATEMENTS flag in the connect method.

**multiple_statements.c**

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

void finish_with_error(MYSQL *con)
{
  fprintf(stderr, "%s\n", mysql_error(con));
  mysql_close(con);
  exit(1);
}

int main(int argc, char **argv)
{
  int status = 0;

  MYSQL *con = mysql_init(NULL);

  if (con == NULL)
  {
      fprintf(stderr, "mysql_init() failed\n");
      exit(1);
  }

  if (mysql_real_connect(con, "localhost", "user12", "34klq*",
          "testdb", 0, NULL, CLIENT_MULTI_STATEMENTS) == NULL)
  {
      finish_with_error(con);
  }

  if (mysql_query(con, "SELECT name FROM cars WHERE id=2;\
      SELECT name FROM cars WHERE id=3;SELECT name FROM cars WHERE id=6"))
  {
      finish_with_error(con);
  }

  do {
      MYSQL_RES *result = mysql_store_result(con);

      if (result == NULL)
      {
          finish_with_error(con);
      }

      MYSQL_ROW row = mysql_fetch_row(result);

      printf("%s\n", row[0]);

      mysql_free_result(result);

      status = mysql_next_result(con);

      if (status > 0) {
```

```
            finish_with_error(con);
        }

    } while(status == 0);

    mysql_close(con);
    exit(0);
}
```

In the example, we execute three SELECT statements in one query.

```
if (mysql_real_connect(con, "localhost", "user12", "34klq*",
        "testdb", 0, NULL, CLIENT_MULTI_STATEMENTS) == NULL)
{
    finish_with_error(con);
}
```

The last option of the mysql_real_connect() method is the client flag. It is used to enable certain features. The CLIENT_MULTI_STATEMENTS enables the execution of multiple statements. This is disabled by default.

```
if (mysql_query(con, "SELECT name FROM cars WHERE id=2;\
    SELECT name FROM cars WHERE id=3;SELECT name FROM cars WHERE id=6"))
{
    finish_with_error(con);
}
```

The query consists of three SELECT statements. They are separated by the semicolon ; character. The backslash character \ is used to separate the string into two lines. It has nothing to do with multiple statements.

```
do {
...
} while(status == 0);
```

The code is placed between the do/while statements. The data retrieval is to be done in multiple cycles. We retrieve data for each SELECT statement separately.

```
status = mysql_next_result(con);
```

We expect multiple result sets. Therefore, we call the mysql_next_result() function. It reads the next statement result and returns a status to indicate whether more results exist. The function returns 0 if the execution went OK and there are more results. It returns -1, when it is executed OK and there are no more results. Finally, it returns value greater than zero if an error occurred.

```
if (status > 0) {
    finish_with_error(con);
}
```

We check for error.

```
$ ./multiple_statements
Mercedes
Skoda
Citroen
```

Example output.

## MySQL C insert image

Some people prefer to put their images into the database, some prefer to keep them on the file system for their applications. Technical difficulties arise when we work with lots of images. Images are binary data. MySQL database has a special data type to store binary data called BLOB (Binary Large Object).

```
mysql> CREATE TABLE images(id INT PRIMARY KEY, data MEDIUMBLOB);
```

For our examples, we create a new Images table. The image size can be up to 16 MB. It is determined by the MEDIUMBLOB data type.

**insert_image.c**

```
#include <mysql.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void finish_with_error(MYSQL *con)
{
  fprintf(stderr, "%s\n", mysql_error(con));
  mysql_close(con);
```

```
  exit(1);
}

int main(int argc, char **argv)
{

  FILE *fp = fopen("sid.jpg", "rb");

  if (fp == NULL)
  {
      fprintf(stderr, "cannot open image file\n");
      exit(1);
  }

  fseek(fp, 0, SEEK_END);

  if (ferror(fp)) {

      fprintf(stderr, "fseek() failed\n");
      int r = fclose(fp);

      if (r == EOF) {
          fprintf(stderr, "cannot close file handler\n");
      }

      exit(1);
  }

  int flen = ftell(fp);

  if (flen == -1) {

      perror("error occurred");
      int r = fclose(fp);

      if (r == EOF) {
          fprintf(stderr, "cannot close file handler\n");
      }

      exit(1);
  }

  fseek(fp, 0, SEEK_SET);

  if (ferror(fp)) {

      fprintf(stderr, "fseek() failed\n");
      int r = fclose(fp);

      if (r == EOF) {
          fprintf(stderr, "cannot close file handler\n");
      }

      exit(1);
  }

  char data[flen+1];

  int size = fread(data, 1, flen, fp);

  if (ferror(fp)) {

      fprintf(stderr, "fread() failed\n");
      int r = fclose(fp);

      if (r == EOF) {
          fprintf(stderr, "cannot close file handler\n");
      }

      exit(1);
  }

  int r = fclose(fp);

  if (r == EOF) {
      fprintf(stderr, "cannot close file handler\n");
  }

  MYSQL *con = mysql_init(NULL);

  if (con == NULL)
  {
      fprintf(stderr, "mysql_init() failed\n");
      exit(1);
  }

  if (mysql_real_connect(con, "localhost", "user12", "34klq*",
          "testdb", 0, NULL, 0) == NULL)
  {
      finish_with_error(con);
  }

  char chunk[2*size+1];
  mysql_real_escape_string(con, chunk, data, size);

  char *st = "INSERT INTO images(id, data) VALUES(1, '%s')";
  size_t st_len = strlen(st);
```

```
  char query[st_len + 2*size+1];
  int len = snprintf(query, st_len + 2*size+1, st, chunk);

  if (mysql_real_query(con, query, len))
  {
      finish_with_error(con);
  }

  mysql_close(con);
  exit(0);
}
```

In this example, we insert one image into the images table.

```
#include <string.h>
```

This include is for the strlen() function.

```
FILE *fp = fopen("woman.jpg", "rb");

if (fp == NULL)
{
    fprintf(stderr, "cannot open image file\n");
    exit(1);
}
```

Here we open the image file. In the current working directory, we should have the sid.jpg file.

```
fseek(fp, 0, SEEK_END);

if (ferror(fp)) {

    fprintf(stderr, "fseek() failed\n");
    int r = fclose(fp);

    if (r == EOF) {
        fprintf(stderr, "cannot close file handler\n");
    }

    exit(1);
}
```

We move the file pointer to the end of the file using the fseek() function. We are going to determine the size of the image. If an error occurs, the error indicator is set. We check the indicator using the fseek() function. In case of an error, we also close the opened file handler.

```
int flen = ftell(fp);

if (flen == -1) {

    perror("error occurred");
    int r = fclose(fp);

    if (r == EOF) {
        fprintf(stderr, "cannot close file handler\n");
    }

    exit(1);
}
```

For binary streams, the ftell() function returns the number of bytes from the beginning of the file, e.g. the size of the image file. In case of an error, the function returns -1 and the errno is set. The perrro() function interprets the value of errno as an error message, and prints it to the standard error output stream.

```
char data[flen+1];
```

In this array, we are going to store the image data.

```
int size = fread(data, 1, flen, fp);
```

We read the data from the file pointer and store it in the data array. The total number of elements successfully read is returned.

```
int r = fclose(fp);

if (r == EOF) {
    fprintf(stderr, "cannot close file handler\n");
}
```

After the data is read, we can close the file handler.

```
char chunk[2*size+1];
mysql_real_escape_string(con, chunk, data, size);
```

The mysql_real_escape_string() function adds an escape character, the backslash, \, before certain potentially dangerous characters in a string passed in to the function. This can help prevent SQL injection attacks. The new buffer must be at least 2*size+1 long.

```
char *st = "INSERT INTO images(id, data) VALUES(1, '%s')";
size_t st_len = strlen(st);
```

Here we start building the SQL statement. We determine the size of the SQL string using the strlen() function.

```
char query[st_len + 2*size+1];
int len = snprintf(query, st_len + 2*size+1, st, chunk);
```

The query must take be long enough to contain the size of the SQL string statement and the size of the image file. Using the snprintf() function, we write the formatted output to query buffer.

```
if (mysql_real_query(con, query, len))
{
    finish_with_error(con);
};
```

We execute the query using the mysql_real_query() function. The mysql_query() cannot be used for statements that contain binary data; we must use the mysql_real_query() instead.

## MySQL C select image

In the previous example, we have inserted an image into the database. In the following example, we select the inserted image from the database.

**select_image.c**

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

void finish_with_error(MYSQL *con)
{
  fprintf(stderr, "%s\n", mysql_error(con));
  mysql_close(con);
  exit(1);
}

int main(int argc, char **argv)
{
  FILE *fp = fopen("sid2.jpg", "wb");

  if (fp == NULL)
  {
      fprintf(stderr, "cannot open image file\n");
      exit(1);
  }

  MYSQL *con = mysql_init(NULL);

  if (con == NULL)
  {
      fprintf(stderr, "mysql_init() failed\n");
      exit(1);
  }

  if (mysql_real_connect(con, "localhost", "user12", "34klq*",
          "testdb", 0, NULL, 0) == NULL)
  {
      finish_with_error(con);
  }

  if (mysql_query(con, "SELECT data FROM images WHERE id=1"))
  {
      finish_with_error(con);
  }

  MYSQL_RES *result = mysql_store_result(con);

  if (result == NULL)
  {
      finish_with_error(con);
  }

  MYSQL_ROW row = mysql_fetch_row(result);
  unsigned long *lengths = mysql_fetch_lengths(result);

  if (lengths == NULL) {
      finish_with_error(con);
  }
```

```
    fwrite(row[0], lengths[0], 1, fp);

    if (ferror(fp))
    {
        fprintf(stderr, "fwrite() failed\n");
        mysql_free_result(result);
        mysql_close(con);

        exit(1);
    }

    int r = fclose(fp);

    if (r == EOF) {
        fprintf(stderr, "cannot close file handler\n");
    }

    mysql_free_result(result);
    mysql_close(con);

    exit(0);
}
```

In this example, we create an image file from the database.

```
FILE *fp = fopen("sid2.jpg", "wb");

if (fp == NULL)
{
    fprintf(stderr, "cannot open image file\n");
    exit(1);
}
```

We open a new file handler for writing.

```
if (mysql_query(con, "SELECT data FROM images WHERE id=1"))
{
    finish_with_error(con);
}
```

We select the data column from the images table with id 1.

```
MYSQL_ROW row = mysql_fetch_row(result);
```

The row contains raw data.

```
unsigned long *lengths = mysql_fetch_lengths(result);
```

We get the length of the image.

```
fwrite(row[0], lengths[0], 1, fp);

if (ferror(fp))
{
    fprintf(stderr, "fwrite() failed\n");
    mysql_free_result(result);
    mysql_close(con);

    exit(1);
}
```

We write the retrieved data to the disk using the fwrite() function call. We check for the error indicator with the ferror() function.

```
int r = fclose(fp);

if (r == EOF) {
    fprintf(stderr, "cannot close file handler\n");
}
```

After we have written the image data, we close the file handler using the fclose() function.

This was MySQL C API tutorial. You may be also interested in PyMySQL tutorial, MySQL Visual Basic tutorial, or PHP mysqli tutorial, PostgreSQL C tutorial, or SQLite C tutorial on ZetCode.