## MySQL数据库API库

访问MySQL服务器，这需要使用mysqlclient库，MySQL的大多数客户端API（除Java和.NET）都是通过这个库来和MySQL服务器通讯的，而这个库正是使用C语言编写的。

可使用mysql -V 命令查看当前系统内所使用的mysql数据库版本信息。数据库版本为5.6.20版。因此，我们可从帮助手册refman-5.6-en.a4.pdf入手，了解学习MySQL C API使用的一般信息。

从API手册23.8中可获取信息，MySQL客户端使用 libmysqlclient 库内部的函数访问MySQL服务器。因此我们在编程过程中，如若使用到库内的函数，必须链接函数库，对应的要找到头文件所在目录位置、函数库路径。以便我们在使用gcc编译工具时可以填充参数-I、-L、-l。

从手册中可获知，函数库名为mysqlclient。

因此我们使用命令find / -name libmysqlclient\* 查找该库的路径。得到 /usr/lib64/mysql/libmysqlclient.a。

nm /usr/lib64/mysql/libmysqlclient.a命令可查看库内包含的函数。

### 编写hello应用链接函数库

编写一个hello.c应用程序，链接使用该库。

用到头文件 <mysql.h> 可使用locate mysql.h查看其目录位置/usr/include/mysql/mysql.h。

编译引用了库的应用程序。

gcc hello.c -o hello -I/usr/include/mysql/ -L/usr/lib64/mysql/ -lmysqlclient

参见帮助手册refman-5.6-en.a4.pdf：23.8.4.3小节。

## MySQL API常用函数

### 总体印象

使用MySQL库API函数的一般步骤：

a. **初始化.**  MYSQL \*mysql\_init(MYSQL \*mysql)

b. **错误处理** unsigned int mysql\_errno(MYSQL \*mysql)

char \*mysql\_error(MYSQL \*mysql);

c. **建立连接.** 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 long client\_flag);

d. **执行SQL语句** int mysql\_query(MYSQL \*mysql, const char \*stmt\_str)

e. **获取结果** MYSQL\_RES \*mysql\_store\_result(MYSQL \*mysql)

MYSQL\_ROW mysql\_fetch\_row(MYSQL\_RES \*result)

f. **释放内存** void mysql\_free\_result(MYSQL\_RES \*result)

g. **关闭连接** void mysql\_close(MYSQL \*mysql)

### 初始化

编写程序测试 初始化函数MYSQL \*mysql\_init(MYSQL \*mysql)。

其中有一种新数据类型MYSQL。可在头文件mysql.h → 263. typedef struct st\_mysql {...} MYSQL;找到其定义。是一个结构体。

处理错误码的函数：unsigned int mysql\_errno(MYSQL \*mysql)

#include <stdio.h>

#include "mysql.h"

int main(void)

{

int i, ret = 0, num = 0;

MYSQL \*mysql = mysql\_init(NULL);

if (mysql == NULL) {

ret = mysql\_errno(mysql);

printf("mysql\_init err:%d\n", ret);

return ret;

}

printf("init ok...\n");

return 0;

}

编译出错，原因是64位Linux环境下，动态库配置不完整。需手动指定编译所用的动态库。根据错误提示分析需要加入如下函数库：

1. \_\_gxx\_personality\_v0 --> -lstdc++ 使用g++相关的环境

2. dlclose/dlopen/dlsym --> -ldl 完成用一个程序加载其他动态库的作用。

3. pthread\_\* --> -lpthread 线程库

4. `my\_getsystime'/`clock\_gettime' --> -lrt librt.so是glibc中对real-time的支持库

使用ldd命令可以查看该可执行文件运行所依赖的库文件。

### Makefile 管理

src = $(wildcard \*.c)

target = $(patsubst %.c, %, $(src))

inc\_path = /usr/include/mysql/

lib\_path = /usr/lib64/mysql/

all: $(target)

%:%.c

gcc $< -o $@ -I$(inc\_path) -L$(lib\_path) -lmysqlclient -lstdc++ -lpthread -ldl -lrt

clean:

-rm -rf $(target)

.PHONY: all clean

注意：在测试makefile时，应先使用-n参数，检查无误再执行。

### 连接数据库关闭连接

依据proc猜想应该是一个类似于connect的函数，查看API文档发现：mysql\_connect();但该函数已经过时，应该使用手册中推荐的mysql\_real\_connect函数取而代之。

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 long client\_flag)

根据手册中的描述，我们可以使用基础的链接方式与MySQL数据库建立连接。

mysql = mysql\_real\_connect(mysql, "localhost", "root", "123456", "mydb61", 0, NULL, 0);

连接数据库成功。对表中数据进行访问，访问结束需调用void mysql\_close(MYSQL \*mysql) 函数关闭连接。该函数在断开连接的同时，还可以解除分配由mysql指向的连接句柄。

mysql\_close(mysql);

### 读取数据

#### 查询表数据

mysql\_query函数不单单能完成查询sql的功能，还能完成非select语句在c程序中的执行。是一个十分万能的c程序中执行SQL语句的函数。并且该函数本身直接支持静态SQL。查询以\0结尾的字符串。如果语句中包含二进制数据，则需要调用mysql\_real\_query来执行查询语句。

函数原型：int mysql\_query(MYSQL \*mysql, const char \*query); 成功返回0，失败返回非0

char \*psql = "select \* from emp";

ret = mysql\_query(mysql, psql);

若执行的是UPDATE, DELETE或INSERT语句，则可通过mysql\_affected\_rows()获知受影响的记录数。

若执行的是SELECT语句，查询结束后，查询结果被保存在mysql句柄中。需要使用获取结果集的API函数将结果集获取出来。有两种方式可以获取结果集。

注意: mysql\_query执行的SQL语句不应为语句添加终结分号（‘;’）或“\g”。

#### 获取结果集

一种方式是通过mysql\_store\_result()将整个结果集全部取回来。另一种方式则是调用mysql\_use\_result()初始化获取操作，但暂时不取回任何记录。视结果集的条目数选择获取结果集的函数。两种方法均通过mysql\_fetch\_row()来访问每一条记录。

MYSQL\_RES \*mysql\_store\_result(MYSQL \*mysql) 成功返回MYSQL\_RES结果集指针，失败返回NULL。

MYSQL\_RES是一个结构体类型，可以从mysql.h头文件中找到该结构体的定义：

mysql.h → 308. typedef struct st\_mysql\_res {...} MYSQL\_RES;

整体获取的结果集，保存在 MYSQL\_RES 结构体指针中，通过检查mysql\_store\_result()是否返回NULL，可检测函数执行是否成功：

MYSQL\_RES \*result = mysql\_store\_result(mysql);

if (result == NULL) {

ret = mysql\_errno(mysql);

printf("mysql\_store\_result error: %s\n", mysql\_error(mysql));

return ret;

}

该函数调用成功，则SQL查询的结果被保存在result中，但我们不清楚有多少条数据。所以应使用游标的方式将结果集中的数据逐条取出。

#### 解析结果集

通过游标一行一行fetch结果集中的数据。根据游标使用的一般特性，应使用循环结构，到达结尾或者出错，返回NULL。

函数原型：MYSQL\_ROW mysql\_fetch\_row(MYSQL\_RES \*result) 成功返回下一行的MYSQL\_ROW结构。如果没有更多要检索的行或出现了错误，返回NULL。-----MYSQL\_ROW定义在118行

select \* from emp 可以看到emp表一共有8列数据。可以循环将每行上每一列的数据显示到屏幕。

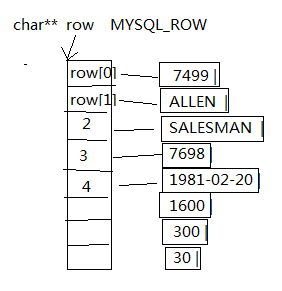
MYSQL\_ROW row = NULL; //typedef char \*\*MYSQL\_ROW;

while ((row = mysql\_fetch\_row(result))) {

printf("%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n", row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7]);

}

MYSQL\_ROW的本质是 typedef char \*\* MYSQL\_ROW; 数据信息存储的形式如下图所示：



从mysql.h头文件可查看MYSQL\_ROW定义: 118. typedef char \*\*MYSQL\_ROW; /\*return data as array of string\*/

从上图分析MYSQL\_ROW为什么被定义为char\*\*类型呢？推测mysq\_fetch\_row()的函数实现大致思想如下：

char \*\*mysql\_fetch\_row(){

char \*\*tmp = (char \*\*) malloc(sizeof(char \*) \* 8);

for (i = 0; i < 8; i++) {

tmp[i] = (char \*)malloc(50);

}

strcpy(tmp[0], "7369");

strcpy(tmp[1], "SMITH");

strcpy(tmp[2], "CLERK");

...

return tmp;

}

#### 释放结果集

结果集处理完成，应调用对应的函数释放所占用的内存。

void mysql\_free\_result(MYSQL\_RES \*result); 成功释放参数传递的结果集。没有失败情况。

mysql\_free\_result(result);

思考：上述实现是直接在MySQL工具中数出列数。找寻能获取列数的API函数、获取表头的API函数。

#### 获取列数

查看帮助手册可以看到，有两个函数具备获取列数的功能：

unsigned int mysql\_field\_count(MYSQL \*mysql) 从mysql句柄中获取有多少列。

unsigned int mysql\_num\_fields(MYSQL\_RES \*result) 从返回的结果集中获取有多少列。

选择任意一种方式均可以完成该功能。

int num = mysql\_field\_count(connect);

while (row = mysql\_fetch\_row(result)) {

for (i = 0; i < num; i++) {

printf("%s\t", row[i]);

}

printf("\n");

//printf("%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n", row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7]);

}

#### 获取表头

获取表头的API函数同样有两个：

MYSQL\_FIELD \*mysql\_fetch\_fields(MYSQL\_RES \*result) 全部获取

MYSQL\_FIELD \*mysql\_fetch\_field(MYSQL\_RES \*result) 获取单个

MYSQL\_FIELD也是一个结构体类型，其内部保存了选择列表项的信息，其中的name成员变量就保存着列名。可从头文件mysql.h中94-116行找到其定义。

MYSQL\_FIELD \*fields = NULL;

fields = mysql\_fetch\_fields(result); //得到表头的结构体数组

for (i = 0; i < num; i++) { //已通过 mysql\_field\_count 获取了总列数

printf("%s\t", fields[i].name); //每一列的列名保存在name成员中

}

printf("\n");

### 示例程序

#include <stdio.h>

#include "mysql.h"

int main(void)

{

int i, ret = 0, num = 0;

char \*psql = "select \* from emp";

MYSQL\_RES \*result = NULL;

MYSQL\_FIELD \*fields = NULL;

MYSQL\_ROW row = NULL;

MYSQL \*mysql = mysql\_init(NULL);

if (mysql == NULL) {

ret = mysql\_errno(mysql);

printf("mysql\_init err:%d\n", ret);

return ret;

}

printf("init ok...\n");

mysql = mysql\_real\_connect(mysql, "localhost", "root", "123456", "mydb61", 0, NULL, 0);

if (mysql == NULL) {

ret = mysql\_errno(mysql);

printf("mysql\_init err:%d\n", ret);

return ret;

}

printf("connect ok...\n");

ret = mysql\_query(mysql, psql);

if (ret != 0) {

printf("mysql\_query error: %s\n", mysql\_error(mysql));

return ret;

}

num = mysql\_field\_count(mysql);

result = mysql\_store\_result(mysql);

if (result == NULL) {

ret = mysql\_errno(mysql);

printf("mysql\_store\_result error: %s\n", mysql\_error(mysql));

return ret;

}

fields = mysql\_fetch\_fields(result);

for (i = 0; i < num; i++) {

printf("%10s\t", fields[i].name);

}

printf("\n");

while ((row = mysql\_fetch\_row(result))) {

for (i = 0; i < num; i++) {

printf("%10s\t", row[i]);

}

printf("\n");

}

mysql\_free\_result(result);

mysql\_close(mysql);

return 0;

}

## MySQL tools实现

依托我们所学习的MySQL基础类API函数，可以编写程序实现简单的sqlplus/mysql 工具的功能。

### 思路分析

1. 仿照mysql工具，应在连接数据库成功之后，在一个while循环中不断的接受用户输入的SQL语句。定义char sqlbuf[1024] 存储用户输入的SQL语句。初始化该buf，并提示用户输入SQL语句。使用gets函数在循环中动态接收用户输入。

while (1) {

memset(sqlbuf, 0, sizeof(sqlbuf));

printf("\nYourSQL> ");

fgets(sqlbuf, sizeof(sqlbuf), stdin);

}

1. 在mysql\_query(connect, sqlbuf)之前，如果用户输入了“exit”那么程序直接结束。
2. 在执行完 mysql\_query(connect, sqlbuf)之后，应该判别用户输入的是否为select语句。如不是select语句不需要查询结果集、处理结果集等繁复操作。
3. 如用户输入的是有结果集的SQL语句，将获取列数、获取结果集、获取表头、解析结果集、释放结果集等相关代码一起并入if (strncmp(sqlbuf, "select", 6))中。

测试注意：执行SQL语句时不要在结尾加“;”

### 程序实现

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "mysql.h"

int main(void)

{

int i, ret = 0, num = 0;

//char \*psql = "select \* from emp";

char sqlbuf[1024];

MYSQL \*mysql = mysql\_init(NULL);

if (mysql == NULL) {

ret = mysql\_errno(mysql);

printf("mysql\_init err:%d\n", ret);

return ret;

}

printf("init ok...\n");

mysql = mysql\_real\_connect(mysql, "localhost", "root", "123456", "mydb61", 0, NULL, 0);

if (mysql == NULL) {

ret = mysql\_errno(mysql);

printf("mysql\_init err:%d\n", ret);

return ret;

}

printf("connect ok...\n");

ret = mysql\_query(mysql, "set names utf8");

if (ret != 0) {

printf("mysql\_query error: %s\n", mysql\_error(mysql));

return ret;

}

while (1) { memset(sqlbuf, 0, sizeof(sqlbuf));

printf("YourSQL> ");

fgets(sqlbuf, sizeof(sqlbuf), stdin);

if (strncmp(sqlbuf, "exit", 4) == 0 || strncmp(sqlbuf, "quit", 4) == 0) {

break;

}

ret = mysql\_query(mysql, sqlbuf);

if (ret != 0) {

printf("mysql\_query error: %s\n", mysql\_error(mysql));

return ret;

}

if (strncmp(sqlbuf, "select", 6) == 0 || strncmp(sqlbuf, "SELECT", 6) == 0) {

num = mysql\_field\_count(mysql);

MYSQL\_RES \*result = NULL;

result = mysql\_store\_result(mysql);

if (result == NULL) {

ret = mysql\_errno(mysql);

printf("mysql\_store\_result error: %s\n", mysql\_error(mysql));

return ret;

}

//打印表头

MYSQL\_FIELD \*fields = NULL;

fields = mysql\_fetch\_fields(result);

for (i = 0; i < num; i++) {

printf("%10s\t", fields[i].name);

}

printf("\n");

//解析结果集

MYSQL\_ROW row = NULL; //typedef char \*\*MYSQL\_ROW;

while ((row = mysql\_fetch\_row(result))) {

for (i = 0; i < num; i++) {

printf("%10s\t", row[i]);

}

printf("\n");

}

mysql\_free\_result(result);

} else {

printf("\n--- not select sql---\n");

printf("affected: %ld\n", (long)mysql\_affected\_rows(mysql));

}

}

mysql\_close(mysql);

return 0;

}

### 中文问题：

修改mysql\_real\_connect()参数，连接到表中有中文数据的数据库，如mydb2，执行程序，测试显示中文出现乱码。我们可以使用mysql\_query函数来解决该问题。

在 while (1) 之前使用 ret = mysql\_query(mysql, "set names utf8"); 来设置查询属性(也可以加到while中)。表示在查询的时候使用utf8的形式进行查询。

或者mysql\_set\_character\_set(mysql, "utf8");

获取当前使用的字符集: const char \*mysql\_character\_set\_name(MYSQL \*mysql)

## 预处理类API函数：

该类函数解决问题：处理带有占位符的SQL语句。insert into table111(col1, col2, col3) values(?, ?, ?);

这种SQL语句由两部分组成，一部分是SQL语句体模型部分，另一部分是？所匹配的值。

性能、调优是数据库编程永恒不变的主题！如果能把SQL语句框架预先处理好，当真正要执行SQL语句时只需要发送对应的参数到对应的SQL框架中，就能提高客户端访问服务器的速度，且数据量小，可以减少网络通信量，提高数据传输效率高。

元数据（Metadata）：又称中介数据、中继数据，为描述数据的数据，主要是描述数据属性的信息，用来支持如指示存储位置、历史数据、资源查找、文件记录等功能。

根据API提供的案例学习该部分内容。主要有 4 个函数：

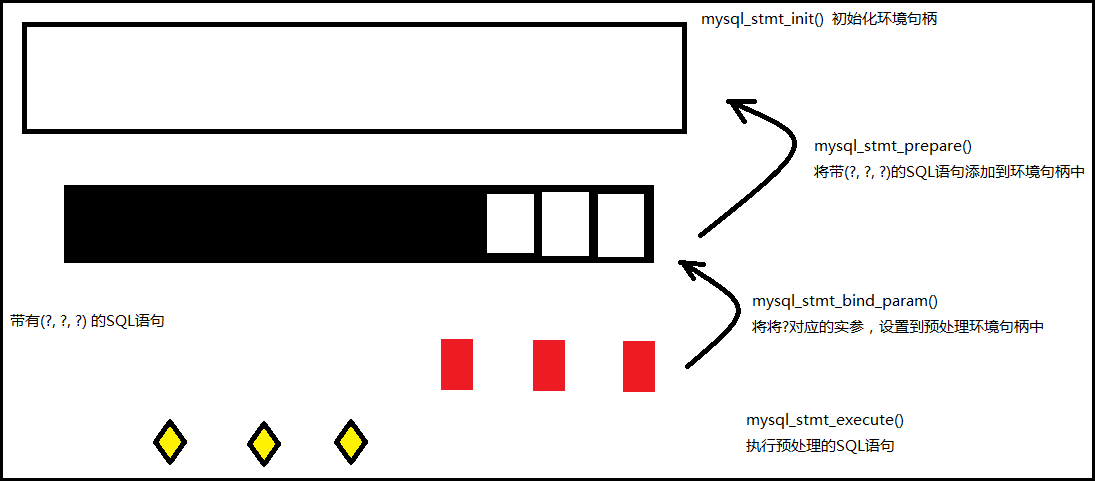
mysql\_stmt\_init() 初始化预处理环境句柄。 返回一个结构体指针 MYSQL\_STMT \*stmt

mysql\_stmt\_prepare() 向上面句柄中添加SQL语句，带有 (?,?,?) 占位符

mysql\_stmt\_param\_count() 求绑定变量的个数(辅助函数)， 有多少个'?'就返回多少

mysql\_stmt\_bind\_param() 将?对应的实参，设置到预处理环境句柄中

mysql\_stmt\_execute() 执行预处理的SQL语句



在不熟悉这套API函数的情况下，如何能快速的找到一个完整的案例，使用这套函数呢？分析：在以上4个过程中，哪个最重要呢？找到它，去查看API文档！发现有对应的demo程序。将该demo导入到我们的程序中，运行，观察它的作用。

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "mysql.h"

#define STRING\_SIZE 50

#define DROP\_SAMPLE\_TABLE "DROP TABLE IF EXISTS test\_table"

#define CREATE\_SAMPLE\_TABLE "CREATE TABLE test\_table(col1 INT,\

col2 VARCHAR(40),\

col3 SMALLINT,\

col4 TIMESTAMP)"

#define INSERT\_SAMPLE "INSERT INTO test\_table(col1,col2,col3) VALUES(?,?,?)"

int main(void)

{

int ret = 0;

MYSQL \*mysql = mysql\_init(NULL);

if (mysql == NULL) {

//unsigned int mysql\_errno(MYSQL \*mysql)

ret = mysql\_errno(mysql);

printf("mysql\_init err:%d\n", ret);

return ret;

}

printf("init ok...\n");

mysql = mysql\_real\_connect(mysql, "localhost", "root", "123456", "mydb61", 0, NULL, 0);

if (mysql == NULL) {

ret = mysql\_errno(mysql);

printf("mysql\_init err:%d\n", ret);

return ret;

}

printf("connect ok...\n");

////////////////////////以下为demo源码////////////////////////////////

MYSQL\_STMT \*stmt;

MYSQL\_BIND bind[3];

my\_ulonglong affected\_rows;

int param\_count;

short small\_data;

int int\_data;

char str\_data[STRING\_SIZE];

unsigned long str\_length;

my\_bool is\_null;

if (mysql\_query(mysql, DROP\_SAMPLE\_TABLE))

{

fprintf(stderr, " DROP TABLE failed\n");

fprintf(stderr, " %s\n", mysql\_error(mysql));

exit(0);

}

if (mysql\_query(mysql, CREATE\_SAMPLE\_TABLE))

{

fprintf(stderr, " CREATE TABLE failed\n");

fprintf(stderr, " %s\n", mysql\_error(mysql));

exit(0);

}

/\* Prepare an INSERT query with 3 parameters \*/

/\* (the TIMESTAMP column is not named; the server \*/

/\* sets it to the current date and time) \*/

stmt = mysql\_stmt\_init(mysql);

if (!stmt)

{

fprintf(stderr, " mysql\_stmt\_init(), out of memory\n");

exit(0);

}

if (mysql\_stmt\_prepare(stmt, INSERT\_SAMPLE, strlen(INSERT\_SAMPLE)))

{

fprintf(stderr, " mysql\_stmt\_prepare(), INSERT failed\n");

fprintf(stderr, " %s\n", mysql\_stmt\_error(stmt));

exit(0);

}

fprintf(stdout, " prepare, INSERT successful\n");

/\* Get the parameter count from the statement \*/

param\_count= mysql\_stmt\_param\_count(stmt);

fprintf(stdout, " total parameters in INSERT: %d\n", param\_count);

if (param\_count != 3) /\* validate parameter count \*/

{

fprintf(stderr, " invalid parameter count returned by MySQL\n");

exit(0);

}

/\* Bind the data for all 3 parameters \*/

memset(bind, 0, sizeof(bind));

/\* INTEGER PARAM \*/

/\* This is a number type, so there is no need to specify buffer\_length \*/

bind[0].buffer\_type= MYSQL\_TYPE\_LONG;

bind[0].buffer= (char \*)&int\_data;

bind[0].is\_null= 0;

bind[0].length= 0;

/\* STRING PARAM \*/

bind[1].buffer\_type= MYSQL\_TYPE\_STRING;

bind[1].buffer= (char \*)str\_data;

bind[1].buffer\_length= STRING\_SIZE;

bind[1].is\_null= 0;

bind[1].length= &str\_length;

/\* SMALLINT PARAM \*/

bind[2].buffer\_type= MYSQL\_TYPE\_SHORT;

bind[2].buffer= (char \*)&small\_data;

bind[2].is\_null= &is\_null;

bind[2].length= 0;

/\* Bind the buffers \*/

if (mysql\_stmt\_bind\_param(stmt, bind))

{

fprintf(stderr, " mysql\_stmt\_bind\_param() failed\n");

fprintf(stderr, " %s\n", mysql\_stmt\_error(stmt));

exit(0);

}

/\* Specify the data values for the first row \*/

int\_data= 10; /\* integer \*/

strncpy(str\_data, "MySQL", STRING\_SIZE); /\* string \*/

str\_length= strlen(str\_data);

/\* INSERT SMALLINT data as NULL \*/

is\_null= 1;

/\* Execute the INSERT statement - 1\*/

if (mysql\_stmt\_execute(stmt))

{

fprintf(stderr, " mysql\_stmt\_execute(), 1 failed\n");

fprintf(stderr, " %s\n", mysql\_stmt\_error(stmt));

exit(0);

}

/\* Get the total number of affected rows \*/

affected\_rows= mysql\_stmt\_affected\_rows(stmt);

fprintf(stdout, " total affected rows(insert 1): %lu\n",

(unsigned long) affected\_rows);

if (affected\_rows != 1) /\* validate affected rows \*/

{

fprintf(stderr, " invalid affected rows by MySQL\n");

exit(0);

}

/\* Specify data values for second row, then re-execute the statement \*/

int\_data= 1000;

strncpy(str\_data, "The most popular Open Source database", STRING\_SIZE);

str\_length= strlen(str\_data);

small\_data= 1000; /\* smallint \*/

is\_null= 0; /\* reset \*/

/\* Execute the INSERT statement - 2\*/

if (mysql\_stmt\_execute(stmt))

{

fprintf(stderr, " mysql\_stmt\_execute, 2 failed\n");

fprintf(stderr, " %s\n", mysql\_stmt\_error(stmt));

exit(0);

}

/\* Get the total rows affected \*/

affected\_rows= mysql\_stmt\_affected\_rows(stmt);

fprintf(stdout, " total affected rows(insert 2): %lu\n",

(unsigned long) affected\_rows);

if (affected\_rows != 1) /\* validate affected rows \*/

{

fprintf(stderr, " invalid affected rows by MySQL\n");

exit(0);

}

/\* Close the statement \*/

if (mysql\_stmt\_close(stmt))

{

fprintf(stderr, " failed while closing the statement\n");

fprintf(stderr, " %s\n", mysql\_stmt\_error(stmt));

exit(0);

}

mysql\_close(mysql);

printf("hello mysql...\n");

return 0;

}

注意：bind --> mysql.h下 MYSQL\_BIND结构体 bind[3]; 是一个结构体数组。有3个‘?’占位符，所以用三个结构体(数组)来对应保存信息。0->第一列；1->第二列；2->第三列。

mysql.h中查找 MYSQL\_BIND 结构体原型。对比：select \* from teat\_table; 和 desc test\_table; 的查询结果。

帮助理解bind的小程序框架：

typedef struct xxx{

char \*p1;

char \*p2;

char \*p3;

} MYSQL\_BIND;

void saveXXXInfo(MYSQL\_BIND \*bind, int num)

{

insert into test\_table valudes(bind[0].p2, bind[1].p3, bind[2].p1);

}

int main(void)

{

MYSQL\_BIND bind[3];

bind[0].p1 = "类型1"; //第一列

bind[0].p2 = "10";

bind[0].p3 = "其他数据";

bind[1].p1 = "类型"; //第二列

bind[1].p2 = "数据";

bind[1].p3 = "描述";

bind[2].p1 = "p1p1p1"; //第三列

//第四列是时间戳，不需要用户使用?来指定，直接使用了系统时间。

bind[2].p2 = "数据";

bind[2].p3 = "其他限定条件";

saveXXXInfo(bind, 3);

}

## 日期时间类API函数

练习：熟悉上述预处理类工作模式，模拟精简一个将时间插入数据库的程序。将时间存入数据库有两种方式： 1. 使用SQL语句方式

2. 预处理环境句柄变量方式存入

提示：

MYSQL\_TIME ts; 浏览头文件 mysql\_time.h 熟悉MYSQL\_TIME结构体。

MYSQL\_BIND bind[3];

MYSQL\_STMT \*stmt;

可直接使用SQL语句提前创建表test\_table2，也可以使用mysql\_query函数来创建。

create table test\_table2 (date\_field date, time\_field time, timestamp\_field timestamp);

char query[1024] = "INSERT INTO test\_table2(date\_field, time\_field, timestamp\_field) VALUES(?,?,?)";

stmt = mysql\_stmt\_init(mysql);

MYSQL\_TIME 是一个结构体，使用typedef定义。位于mysql\_time.h文件中。

API参考：refman-5.6-en.a4.pdf手册25.2.10. 日期和时间值的C API处理

## 多查询执行的C API函数

一次性执行多条SQL语句，包括select、drop、update、create等。 如：

mysql\_query(mysql,"DROP TABLE IF EXISTS test\_table;\

CREATE TABLE test\_table(id INT);\

INSERT INTO test\_table VALUES(10);\

UPDATE test\_table SET id=20 WHERE id=10;\

SELECT \* FROM test\_table;\

DROP TABLE test\_table");

文档：25.2.9. 多查询执行的C API处理。中文文档只有demo框架。查阅对应英文文档refman-5.6-en.a4.pdf。关键字Multiple 23.8.17

注意：打桩函数——函数接口

if (mysql\_real\_connect (mysql, host\_name, user\_name, password,

db\_name, port\_num, socket\_name, CLIENT\_MULTI\_STATEMENTS) == NULL)

CLIENT\_MULTI\_STATEMENTS：客户端通知Server，将要发送多个SQL语句。

mysql\_field\_count(mysql)：影响的行数。 如：

当select \* from dept; 执行结束，提示：“5 rows in set” 表示影响了4行。

当Create一张表， 执行结束，提示：“Query OK, 0 rows affected (0.01 sec)”

当delete一行， 执行结束，提示：“Query OK, 1 row affected (0.00 sec)”

mysql\_field\_count函数调用后会将影响的行数保存到句柄 mysql 中。

将帮助文档中的demo导入程序，分析与我们之前掌握的API函数间的区别与联系：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "mysql.h"

void process\_result\_set(MYSQL \*mysql, MYSQL\_RES \*result)

{

int i, num;

num = mysql\_field\_count(mysql);

MYSQL\_FIELD \*fields = NULL;

fields = mysql\_fetch\_fields(result);

for (i = 0; i < num; i++) {

printf("%10s\t", fields[i].name);

}

printf("\n");

MYSQL\_ROW row = NULL;

while ((row = mysql\_fetch\_row(result))) {

for (i = 0; i < num; i++) {

printf("%10s\t", row[i]);

}

printf("\n");

}

}

int main(void)

{

int ret = 0, status = 0;

MYSQL\_RES \*result = NULL;

MYSQL \*mysql = mysql\_init(NULL);

if (mysql == NULL) {

//unsigned int mysql\_errno(MYSQL \*mysql)

ret = mysql\_errno(mysql);

printf("mysql\_init err:%d\n", ret);

return ret;

}

mysql = mysql\_real\_connect(mysql, "localhost", "root", "123456", "mydb61", 0, NULL, CLIENT\_MULTI\_STATEMENTS);

if (mysql == NULL) {

ret = mysql\_errno(mysql);

printf("mysql\_init err:%d\n", ret);

return ret;

}

/////////////////////////////以下为demo源码//////////////////////////////

/\* execute multiple statements \*/

status = mysql\_query(mysql,"DROP TABLE IF EXISTS test\_table;\

CREATE TABLE test\_table(id INT);\

INSERT INTO test\_table VALUES(10);\

UPDATE test\_table SET id=20 WHERE id=10;\

SELECT \* FROM test\_table;");

DROP TABLE test\_table

if (status)

{

printf("Could not execute statement(s)");

mysql\_close(mysql);

exit(0);

}

/\* process each statement result \*/

do {

/\* did current statement return data? \*/

result = mysql\_store\_result(mysql);

if (result)

{

/\* yes; process rows and free the result set \*/

process\_result\_set(mysql, result);

mysql\_free\_result(result);

}

else /\* no result set or error \*/

{

if (mysql\_field\_count(mysql) == 0)

{

printf("%lld rows affected\n",

mysql\_affected\_rows(mysql));

}

else /\* some error occurred \*/

{

printf("Could not retrieve result set\n");

break;

} }

/\* more results? -1 = no, >0 = error, 0 = yes (keep looping) \*/

if ((status = mysql\_next\_result(mysql)) > 0)

printf("Could not execute statement\n");

printf("------------status: %d\n", status);

} while (status == 0);

mysql\_close(mysql);

return 0;

}

process\_result\_set函数是文档中给我们预留的打桩函数，需要我们在使用的过程中，自己实现它。

函数实现就是借助mysql和result两个参数打印一条sql语句查询到的结果集到屏幕。

可以直接使用mysq\_tool.c中if (strncmp(sqlbuf, "select", 6) == 0 || strncmp(sqlbuf, "SELECT", 6) == 0)内的代码。“获取结果集”片段可以删除。“释放结果集”片段可以删除。API示例中含有该部分内容。

常见错误：在process\_result\_set函数实现中，不要使用mysql\_store\_result（mysql）再次获取结果集， 该result已经在API函数接口传入，直接使用参数result即可。否则会出现【段错误】。

## MySQL中的事务

测试MySQL中事务的特性。

MySQL的事务的默认自动提交的，每执行一个sql语句都自动commit

Oracle的事务是自动打开的(以你执行的一条DML语句为标志)，但每次执行需要手动commit

在程序中设置autocommit修改MySQL事务的属性。

set autocommit = 0 禁止自动提交

set autocommit = 1 开启自动提交MySQL中InnoDB引擎才支持事务默认自动提交机制。MYISAM引擎不支持。

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "mysql.h"

#define SET\_TRAN "SET AUTOCOMMIT=0" //手动commit

#define UNSET\_TRAN "SET AUTOCOMMIT=1" //自动commit

//设置事务为手动提交

int mysql\_OperationTran(MYSQL \*mysql)

{

//--开启事务

int ret = mysql\_query(mysql, "start transaction");

if (ret != 0) {

printf("mysql\_OperationTran query start err: %s\n", mysql\_error(mysql));

return ret;

}

//--设置事务为手动提交

ret = mysql\_query(mysql, SET\_TRAN);

if (ret != 0) {

printf("mysql\_OperationTran query set err: %s\n", mysql\_error(mysql));

return ret;

}

return ret;

}

//设置事务为自动提交

int mysql\_AutoTran(MYSQL \*mysql)

{

//--开启事务

int ret = mysql\_query(mysql, "start transaction");

if (ret != 0) {

printf("mysql\_AutoTran query start err: %s\n", mysql\_error(mysql));

return ret;

}

//--设置事务为自动提交

ret = mysql\_query(mysql, UNSET\_TRAN);

if (ret != 0) {

printf("mysql\_AutoTran query set err: %s\n", mysql\_error(mysql));

return ret;

}

return ret;

}

//执行commit，手动提交事务

int mysql\_Commit(MYSQL \*mysql)

{

int ret = mysql\_query(mysql, "COMMIT");

if (ret != 0) {

printf("commit err: %s\n", mysql\_error(mysql));

return ret;

}

return ret;

}

//执行rollback，回滚事务

int mysql\_Rollback(MYSQL \*mysql)

{

int ret = mysql\_query(mysql, "ROLLBACK");

if (ret != 0) {

printf("rollback err: %s\n", mysql\_error(mysql));

return ret;

}

return ret;

}

#define DROP\_SAMPLE\_TABLE "DROP TABLE IF EXISTS test\_table"

#define CREATE\_SAMPLE\_TABLE "CREATE TABLE test\_table(col1 INT,\

col2 VARCHAR(10),\

col3 VARCHAR(10))"

#define sql01 "INSERT INTO test\_table(col1,col2,col3) VALUES(10, 'AAA', 'A1')"

#define sql02 "INSERT INTO test\_table(col1,col2,col3) VALUES(20, 'BBB', 'B2')"

#define sql03 "INSERT INTO test\_table(col1,col2,col3) VALUES(30, 'CCC', 'C3')"

#define sql04 "INSERT INTO test\_table(col1,col2,col3) VALUES(40, 'DDD', 'D4')"

int main(void)

{

int ret = 0;

MYSQL \*mysql = mysql\_init(NULL);

mysql = mysql\_real\_connect(mysql, "localhost", "root", "123456", "mydb2", 0, NULL, 0);

if (mysql == NULL) {

ret = mysql\_errno(mysql);

printf("func mysql\_real\_connect() err：%d\n", ret);

return ret;

}

printf(" --- connect ok......\n");

if (mysql\_query(mysql, DROP\_SAMPLE\_TABLE)) {

fprintf(stderr, " DROP TABLE failed\n");

fprintf(stderr, " %s\n", mysql\_error(mysql));

exit(0);

}

if (mysql\_query(mysql, CREATE\_SAMPLE\_TABLE)) {

fprintf(stderr, " CREATE TABLE failed\n");

fprintf(stderr, " %s\n", mysql\_error(mysql));

exit(0);

}

ret = mysql\_OperationTran(mysql); //开启事务，并修改事务属性为手动commit

if (ret != 0) {

printf("mysql\_OperationTran() err:%d\n", ret);

return ret;

}

ret = mysql\_query(mysql, sql01); //向表中插入第一行数据 ‘AAA’

if (ret != 0) {

printf("mysql\_query() err:%d\n", ret); return ret;

}

ret = mysql\_query(mysql, sql02); //向表中插入第二行数据 ‘BBB’

if (ret != 0) {

printf("mysql\_query() err:%d\n", ret);

return ret;

}

ret = mysql\_Commit(mysql); //手动提交事务

if (ret != 0) {

printf("mysql\_Commit() err:%d\n", ret);

return ret;

}

ret = mysql\_AutoTran(mysql); // =再次= 修改事务属性为【自动】commit

if (ret != 0) {

printf("mysql\_OperationTran() err:%d\n", ret);

return ret;

}

ret = mysql\_OperationTran(mysql); // =再次= 修改事务属性为【手动】commit

if (ret != 0) {

printf("mysql\_OperationTran() err:%d\n", ret);

return ret;

}

ret = mysql\_query(mysql, sql03); //向表中插入第三行数据 ‘CCC’

if (ret != 0) {

printf("mysql\_query() err:%d\n", ret);

return ret;

}

ret = mysql\_query(mysql, sql04); //向表中插入第四行数据 ‘DDD’

if (ret != 0) {

printf("mysql\_query() err:%d\n", ret);

return ret;

}

ret = mysql\_Rollback(mysql); //直接rollback操作

if (ret != 0) {

printf("mysql\_Rollback() err:%d\n", ret);

return ret;

}

//rollback操作是否能回退掉CCC、DDD的值，取决于事务属性。

mysql\_close(mysql);

return 0;

}

对应参考API手册。中文：25.2.3.2. 英文：23.8.7.2