# 嵌入式SQL

配置环境

MySql5.6 + VisualStudio2010

1. 找到MySql的lib文件夹: C:\Program Files\MySQL\MySQL Server 5.6\include
2. 找到libmysql.lib和libmysql.dll文件: C:\Program Files\MySQL\MySQL Server 5.6\lib
3. 新建VisualStudio2010 工程
4. 在菜单栏中依次选择: **项目->项目属性->配置属性->C/C++->附加包含目录** 中黏贴lib文件路径(还有一种方法是将C:\Program Files\MySQL\MySQL Server 5.6\include中所有的文件全部复制粘贴到visualStudio2010安装目录下VC/include文件夹中没试过,不确定可不可用)
5. 然后点选 **链接器->输入->附加依赖项** 添加libmysql.lib
6. 将libmysql.lib 和 libmysql.dll 分别放置到VisualStudio2010 安装目录下的VC/bin 和VC/lib文件夹中
7. 若程序在编译时报错 LINK : fatal error LNK1123: 转换到 COFF 期间失败: 文件无效或损坏 ---**项目->项目属性->配置属性->链接器->启用增量链接设置为 否**

建立数据表

create table User

(

Uno char(10) primary key,

Uname char(40) ,

Upswd char(20) ,

Uage smallint

);

insert into User values('0000000001' ,'赵一' ,'123456789' ,20);

insert into User values('0000000002' ,'钱二' ,'234567891' ,22);

insert into User values('0000000003' ,'孙三' ,'345678912' ,21);

insert into User values('0000000004' ,'李四' ,'456789123' ,18);

insert into User values('0000000005' ,'王五' ,'567891234' ,17);

create table Time

(

Uno char(10) ,

Ttime char(20) ,

foreign key (Uno) references User(Uno)

);

insert into Time values('0000000001' ,'周六下午');

insert into Time values('0000000003' ,'周日');

insert into Time values('0000000004' ,'周五下午');

insert into Time values('0000000002' ,'周六');

insert into Time values('0000000005' ,'周日');

insert into Time values('0000000001' ,'周一晚上');

create table Site

(

Uno char(10),

Ssite char(40) ,

foreign key (Uno) references User(Uno)

);

insert into Site values('0000000001' ,'操场');

insert into Site values('0000000003' ,'KTV');

insert into Site values('0000000004' ,'饭店');

insert into Site values('0000000002' ,NULL);

insert into Site values('0000000005' ,'KTV');

insert into Site values('0000000001' ,'篮球场');

将MySql语句嵌入到C++程序中:

**连接数据库**

MYSQL \* STDCALL mysql\_real\_connect(

建立的数据库对象 MYSQL \*mysql,

要连接的主机 const char \*host,

登录主机的用户名 const char \*user,

登录密码 const char \*passwd,

数据库名称 const char \*db,

连接端口 unsigned int port,

Socket对象 const char \*unix\_socket,

? unsigned long clientflag

);

**进行查询**

int STDCALL mysql\_query(MYSQL \*mysql,

查询语句串 const char \*q);

**使用查询结果**

MYSQL\_RES \* STDCALL mysql\_store\_result(MYSQL \*mysql)

**取结果中一行数据**

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

取出的结果都是字符串型的变量,需要直接使用还需要转换;

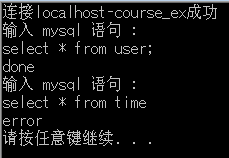
string tmptest = sql\_row[i];

cout << tmptest << endl;

**清除结果变量中保存的数据**

mysql\_free\_result(dynamic\_query);

**遇到的问题**

****

Mysql\_query成功一次后无法仅需进行操作.

原因

没有清除msql\_query返回的结果保存的数据(缓冲区中的数据)

方法:

将请求返回结果保存到MYSQL\_RES类型的变量,并使用mysql\_free\_result()清除释放该变量占用空间.

**相关的数据结构**

typedef struct **st\_mysql\_res**

{

my\_ulonglong row\_count;                               // 结果集的行数

unsigned int field\_count, current\_field;            // 结果集的列数，当前列

**MYSQL\_FIELD** \*fields;                                    // 结果集的列信息

**MYSQL\_DATA** \*data;                                      // 结果集的数据

**MYSQL\_ROWS** \*data\_cursor;                        // 结果集的光标

**MEM\_ROOT** field\_alloc;                                   // 内存结构

**MYSQL\_ROW** row;                                         // 非缓冲的时候用到

**MYSQL\_ROW** current\_row;                           //mysql\_store\_result时会用到。

   unsigned long \*lengths;                                //每列的长度

**MYSQL** \*handle;                                        // mysql\_use\_result会用。

   my\_bool eof;                                                 //是否为行尾

} **MYSQL\_RES;**

typedef char \*\* **MYSQL\_ROW**; /\* 返回的每一行的值，全部用字符串来表示\*/

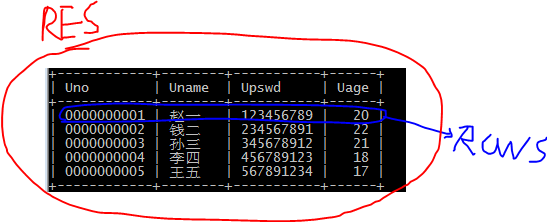
typedef struct st\_mysql\_rows

{

   struct st\_mysql\_rows \*next; /\* list of rows \*/

   MYSQL\_ROW data;

} **MYSQL\_ROWS**;       //mysql的数据的链表节点。可见mysql的结果集是链表结构



typedef struct st\_mysql\_data

{

   my\_ulonglong rows;

   unsigned int fields;

  MYSQL\_ROWS \*data;

   MEM\_ROOT alloc;

} **MYSQL\_DATA**; // 数据集的结构

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**;  //列信息的结构

typedef struct st\_used\_mem

/\* struct for once\_alloc \*/

{

   struct st\_used\_mem \*next; /\* Next block in use \*/

   unsigned int left; /\* memory left in block  \*/

   unsigned int size; /\* Size of block \*/

} **USED\_MEM**; //内存结构

typedef struct st\_mem\_root

{

   USED\_MEM \*free;

   USED\_MEM \*used;

   USED\_MEM \*pre\_alloc;

   unsigned int min\_malloc;

   unsigned int block\_size;

   void (\*error\_handler)(void);

} **MEM\_ROOT**;  //内存结构

之所以列出这些结构，是想让大家对mysql的结果集有更清晰的认识。

以下是取得数据的相关API:

1. **. mysql\_store\_result()**

MYSQL\_RES \* mysql\_store\_result(MYSQL \*mysql)

**描述**

对于成功检索了数据的每个查询（SELECT、SHOW、DESCRIBE、EXPLAIN、CHECK TABLE等），必须调用mysql\_store\_result() 或mysql\_use\_result() 。

对于其他查询，不需要调用mysql\_store\_result()或mysql\_use\_result()，但是如果在任何情况下均调用了mysql\_store\_result()，它也不会导致任何伤害或性能降低。通过检查mysql\_store\_result()是否返回0，可检测查询是否没有结果集（以后会更多）。

如果希望了解查询是否应返回结果集，可使用mysql\_field\_count()进行检查。 mysql\_store\_result()将查询的全部结果读取到客户端，分配1个MYSQL\_RES结构，并将结果置于该结构中。

如果查询未返回结果集，mysql\_store\_result()将返回Null指针（例如，如果查询是INSERT语句）。

如果读取结果集失败，mysql\_store\_result()还会返回Null指针。通过检查mysql\_error()是否返回非空字符串，mysql\_errno()是否返回非0值，或mysql\_field\_count()是否返回0，可以检查是否出现了错误。

如果未返回行，将返回空的结果集。（空结果集设置不同于作为返回值的空指针）。

一旦调用了mysql\_store\_result()并获得了不是Null指针的结果，可调用mysql\_num\_rows()来找出结果集中的行数。

可以调用mysql\_fetch\_row()来获取结果集中的行，或调用mysql\_row\_seek()和mysql\_row\_tell()来获取或设置结果集中的当前行位置。

**一旦完成了对结果集的操作，必须调用mysql\_free\_result()。**

2.**mysql\_use\_result()**

MYSQL\_RES \* mysql\_use\_result(MYSQL \*mysql)

**描述**

对于成功检索数据的每个查询（SELECT、SHOW、DESCRIBE、EXPLAIN），必须用mysql\_store\_result()或mysql\_use\_result()。

mysql\_use\_result()将初始化结果集检索，但并不像mysql\_store\_result()那样将结果集实际读取到客户端。**它必须通过对mysql\_fetch\_row()的调用，对每一行分别进行检索。这将直接从服务器读取结果，而不会将其保存在临时表或本地缓冲区内，与mysql\_store\_result()相比，速度更快而且使用的内存也更少。**客户端仅为当前行和通信缓冲区分配内存，分配的内存可增加到max\_allowed\_packet字节。

另一方面，如果你正在客户端一侧为各行进行大量的处理操作，或者将输出发送到了用户可能会键入“^S”（停止滚动）的屏幕，就不应使用mysql\_use\_result()。这会绑定服务器，并阻止其他线程更新任何表（数据从这类表获得）。

使用mysql\_use\_result()时，必须执行mysql\_fetch\_row()，直至返回NULL值，否则，**未获取的行将作为下一个检索的一部分返回**。C API给出命令不同步错误，如果忘记了执行该操作，将不能运行该命令。

不能对结果集用mysql\_data\_seek()、mysql\_row\_seek()、mysql\_row\_tell()、mysql\_num\_rows()或mysql\_affected\_rows()，也不应发出其他查询，直至mysql\_use\_result()完成为止。（但是，提取了所有行后，mysql\_num\_rows()将准确返回提取的行数）。

**一旦完成了对结果集的操作，必须调用mysql\_free\_result()。**

**返回值**

MYSQL\_RES结果结构。如果出现错误，返回NULL。

3.取得一行的数据：

**mysql\_fetch\_row()**

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

**描述**

检索结果集的下一行。在mysql\_store\_result()之后使用时，如果没有要检索的行，mysql\_fetch\_row()返回NULL。

在mysql\_use\_result()之后使用时，如果没有要检索的行或出现了错误，mysql\_fetch\_row()返回NULL。

结果集的列数目由mysql\_num\_fields(result)给出。如果行中保存了调用mysql\_fetch\_row()返回的值，将按照row[0]到row[mysql\_num\_fields(result)-1]，访问这些值的指针。行中的NULL值由NULL指针指明。

可以通过调用mysql\_fetch\_lengths()来获得行中字段值的长度。对于空字段以及包含NULL的字段，长度为0。通过检查字段值的指针，能够区分它们。如果指针为NULL，字段为NULL，否则字段为空。

**返回值**

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。

示例代码：

4. 取得列的信息。

**mysql\_fetch\_field()**

MYSQL\_FIELD \*mysql\_fetch\_field(MYSQL\_RES \*result)

**描述**

返回采用MYSQL\_FIELD结构的结果集的列。重复调用该函数，以检索关于结果集中所有列的信息。未剩余字段时，mysql\_fetch\_field()返回NULL。

每次执行新的SELECT查询时，将复位mysql\_fetch\_field()，以返回关于第1个字段的信息。调用mysql\_field\_seek()也会影响mysql\_fetch\_field()返回的字段。

如果调用了mysql\_query()以在表上执行SELECT，但未调用mysql\_store\_result()，如果调用了mysql\_fetch\_field()以请求BLOB字段的长度，MySQL将返回默认的Blob长度（8KB）。之所以选择8KB是因为MySQL不知道BLOB的最大长度。应在日后使其成为可配置的。一旦检索了结果集，field->max\_length将包含特定查询中该列的最大值的长度。

**返回值**

当前列的MYSQL\_FIELD结构。如果未剩余任何列，返回NULL。

**错误**

无。

**示例：**

5. 同4.

#### mysql\_fetch\_fields()

MYSQL\_FIELD \*mysql\_fetch\_fields(MYSQL\_RES \*result)

**描述**

对于结果集，返回所有MYSQL\_FIELD结构的数组。每个结构提供了结果集中1列的字段定义。

**返回值**

关于结果集所有列的MYSQL\_FIELD结构的数组。

**示例：**

6. **mysql\_free\_result()**

void mysql\_free\_result(MYSQL\_RES \*result)

**描述**

**释放由mysql\_store\_result()、mysql\_use\_result()、mysql\_list\_dbs()等为结果集分配的内存。完成对结果集的操作后，必须调用mysql\_free\_result()释放结果集使用的内存。**

**释放完成后，不要尝试访问结果集。**

**返回值**

无。

**记得要释放结果集！！！**