SQL中的字符类型

Tinyint 占1字节 http://blog.csdn.net/lyd518/article/details/20703095

Smallint 占2字节

Mediumint 占3字节

Int 占4字节

Bigint 占8字节

Varchar 变长的字符

tinyint(1) 和 tinyint(4) 中的1和4并不表示存储长度，只有字段指定zerofill是有用，  
如tinyint(4)，如果实际值是2，如果列指定了zerofill，查询结果就是0002，左边用0来填充

# 若字段类型定义为int(4),插入的值为12345，值得宽度5超过了定义的显示宽度4，只要定义的类型能容纳的下，数据库不会进行四舍五入的，还是会显示12345,

句柄

句柄（handle），有多种意义，其中第一种是指程序设计，第二种是指Windows编程。现在大部分都是指程序设计/程序开发这类。

第一种解释：句柄是一种特殊的[智能指针](https://baike.baidu.com/item/%E6%99%BA%E8%83%BD%E6%8C%87%E9%92%88) 。当一个应用程序要引用其他系统（如数据库、操作系统）所管理的内存块或对象时，就要使用句柄。

第二种解释：整个Windows编程的基础。一个句柄是指使用的一个唯一的整数值，即一个4字节(64位程序中为8字节)长的数值，来**标识**[**应用程序**](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F)**中的不同**[**对象**](https://baike.baidu.com/item/%E5%AF%B9%E8%B1%A1/17158)**和同类中的不同的**[**实例**](https://baike.baidu.com/item/%E5%AE%9E%E4%BE%8B)**，**诸如，一个窗口，按钮，图标，滚动条，输出设备，控件或者文件等。应用程序能够通过句柄访问相应的对象的信息，但是句柄不是[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88)，程序不能利用句柄来直接阅读文件中的信息。如果句柄不在[I/O](https://baike.baidu.com/item/I%2FO/84718)文件中，它是毫无用处的。 句柄是Windows用来标志应用程序中建立的或是使用的唯一整数，[Windows](https://baike.baidu.com/item/Windows)大量使用了句柄来[标识](https://baike.baidu.com/item/%E6%A0%87%E8%AF%86)对象。

句柄类似指向[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88)的指针，仅仅是类似，通过句柄可以找到对应的数据

句柄与普通指针的区别在于，指针包含的是引用对象的内存地址，而句柄则是由系统所管理的引用标识，该标识可以被系统重新定位到一个内存地址上。这种间接访问对象的模式增强了系统对引用对象的控制。

SQL常用C语言API

数据库C API数据类型

MYSQL

该结构代表1个数据库连接的句柄。几乎所有的MySQL函数均使用它。不应尝试拷贝MYSQL结构。不保证这类拷贝结果会有用。

MYSQL\_RES

该结构代表返回行的查询结果（SELECT, SHOW, DESCRIBE, EXPLAIN）。在本节的剩余部分，将查询返回的信息称为“结果集”。

MYSQL\_ROW

这是1行数据的“类型安全”表示。它目前是按照计数字节字符串的数组实施的。（如果字段值可能包含二进制数据，不能将其当作由Null终结的字符串对待，这是因为这类值可能会包含Null字节）。行是通过调用mysql\_fetch\_row()获得的。

MYSQL\_FIELD

该结构包含关于字段的信息，如字段名、类型和大小。这里详细介绍了其成员。通过重复调用mysql\_fetch\_field()，可为每个字段获得MYSQL\_FIELD结构。字段值不是该结构的组成部份，它们包含在MYSQL\_ROW结构中。

MYSQL\_FIELD\_OFFSET

这是MySQL字段列表偏移量的“类型安全”表示（由mysql\_field\_seek()使用）。偏移量是行内的字段编号，从0开始。

http://blog.csdn.net/lzjsqn/article/details/54385424

|  |  |
| --- | --- |
| **mysql\_affected\_rows()** | 返回被最新的UPDATE, DELETE或INSERT查询影响的行数。 |
| **mysql\_close()** | 关闭一个服务器连接。 |
| **mysql\_connect()** | 连接一个**MySQL**服务器。该函数不推荐；使用mysql\_real\_connect()代替。 |
| **mysql\_change\_user()** | 改变在一个打开的连接上的用户和数据库。 |
| **mysql\_create\_db()** | 创建一个数据库。该函数不推荐；而使用SQL命令CREATE DATABASE。 |
| **mysql\_data\_seek()** | 在一个查询结果集合中搜寻一任意行。 |
| **mysql\_debug()** | 用给定字符串做一个DBUG\_PUSH。 |
| **mysql\_drop\_db()** | 抛弃一个数据库。该函数不推荐；而使用SQL命令DROP DATABASE。 |
| **mysql\_dump\_debug\_info()** | 让服务器将调试信息写入日志文件。 |
| **mysql\_eof()** | 确定是否已经读到一个结果集合的最后一行。这功能被反对; mysql\_errno()或mysql\_error()可以相反被使用。 |
| **mysql\_errno()** | 返回最近被调用的**MySQL**函数的出错编号。 |
| **mysql\_error()** | 返回最近被调用的**MySQL**函数的出错消息。 |
| **mysql\_escape\_string()** | 用在SQL语句中的字符串的转义特殊字符。 |
| **mysql\_fetch\_field()** | 返回下一个表字段的类型。 |
| **mysql\_fetch\_field\_direct ()** | 返回一个表字段的类型，给出一个字段编号。 |
| **mysql\_fetch\_fields()** | 返回一个所有字段结构的数组。 |
| **mysql\_fetch\_lengths()** | 返回当前行中所有列的长度。 |
| **mysql\_fetch\_row()** | 从结果集合中取得下一行。 |
| **mysql\_field\_seek()** | 把列光标放在一个指定的列上。 |
| **mysql\_field\_count()** | 返回最近查询的结果列的数量。 |
| **mysql\_field\_tell()** | 返回用于最后一个mysql\_fetch\_field()的字段光标的位置。 |
| **mysql\_free\_result()** | 释放一个结果集合使用的内存。 |
| **mysql\_get\_client\_info()** | 返回客户版本信息。 |
| **mysql\_get\_host\_info()** | 返回一个描述连接的字符串。 |
| **mysql\_get\_proto\_info()** | 返回连接使用的协议版本。 |
| **mysql\_get\_server\_info()** | 返回服务器版本号。 |
| **mysql\_info()** | 返回关于最近执行得查询的信息。 |
| **mysql\_init()** | 获得或初始化一个MYSQL结构。 |
| **mysql\_insert\_id()** | 返回有前一个查询为一个AUTO\_INCREMENT列生成的ID。 |
| **mysql\_kill()** | 杀死一个给定的线程。 |
| **mysql\_list\_dbs()** | 返回匹配一个简单的正则表达式的数据库名。 |
| **mysql\_list\_fields()** | 返回匹配一个简单的正则表达式的列名。 |
| **mysql\_list\_processes()** | 返回当前服务器线程的一张表。 |
| **mysql\_list\_tables()** | 返回匹配一个简单的正则表达式的表名。 |
| **mysql\_num\_fields()** | 返回一个结果集合重的列的数量。 |
| **mysql\_num\_rows()** | 返回一个结果集合中的行的数量。 |
| **mysql\_options()** | 设置对mysql\_connect()的连接选项。 |
| **mysql\_ping()** | 检查对服务器的连接是否正在工作，必要时重新连接。 |
| **mysql\_query()** | 执行指定为一个空结尾的字符串的SQL查询。 |
| **mysql\_real\_connect()** | 连接一个**MySQL**服务器。 |
| **mysql\_real\_query()** | 执行指定为带计数的字符串的SQL查询。 |
| **mysql\_reload()** | 告诉服务器重装授权表。 |
| **mysql\_row\_seek()** | 搜索在结果集合中的行，使用从mysql\_row\_tell()返回的值。 |
| **mysql\_row\_tell()** | 返回行光标位置。 |
| **mysql\_select\_db()** | 连接一个数据库。 |
| **mysql\_shutdown()** | 关掉数据库服务器。 |
| **mysql\_stat()** | 返回作为字符串的服务器状态。 |
| **mysql\_store\_result()** | 检索一个完整的结果集合给客户。 |
| **mysql\_thread\_id()** | 返回当前线程的ID。 |
| **mysql\_use\_result()** | 初始化一个一行一行地结果集合的检索。 |

|  |  |
| --- | --- |
| **mysql\_affected\_rows()** | 返回被最新的UPDATE, DELETE或INSERT查询影响的行数。 |
| **mysql\_close()** | 关闭一个服务器连接。 |
| **mysql\_connect()** | 连接一个**MySQL**服务器。该函数不推荐；使用mysql\_real\_connect()代替。 |
| **mysql\_change\_user()** | 改变在一个打开的连接上的用户和数据库。 |
| **mysql\_create\_db()** | 创建一个数据库。该函数不推荐；而使用SQL命令CREATE DATABASE。 |
| **mysql\_data\_seek()** | 在一个查询结果集合中搜寻一任意行。 |
| **mysql\_debug()** | 用给定字符串做一个DBUG\_PUSH。 |
| **mysql\_drop\_db()** | 抛弃一个数据库。该函数不推荐；而使用SQL命令DROP DATABASE。 |
| **mysql\_dump\_debug\_info()** | 让服务器将调试信息写入日志文件。 |
| **mysql\_eof()** | 确定是否已经读到一个结果集合的最后一行。这功能被反对; mysql\_errno()或mysql\_error()可以相反被使用。 |
| **mysql\_errno()** | 返回最近被调用的**MySQL**函数的出错编号。 |
| **mysql\_error()** | 返回最近被调用的**MySQL**函数的出错消息。 |
| **mysql\_escape\_string()** | 用在SQL语句中的字符串的转义特殊字符。 |
| **mysql\_fetch\_field()** | 返回下一个表字段的类型。 |
| **mysql\_fetch\_field\_direct ()** | 返回一个表字段的类型，给出一个字段编号。 |
| **mysql\_fetch\_fields()** | 返回一个所有字段结构的数组。 |
| **mysql\_fetch\_lengths()** | 返回当前行中所有列的长度。 |
| **mysql\_fetch\_row()** | 从结果集合中取得下一行。 |
| **mysql\_field\_seek()** | 把列光标放在一个指定的列上。 |
| **mysql\_field\_count()** | 返回最近查询的结果列的数量。 |
| **mysql\_field\_tell()** | 返回用于最后一个mysql\_fetch\_field()的字段光标的位置。 |
| **mysql\_free\_result()** | 释放一个结果集合使用的内存。 |
| **mysql\_get\_client\_info()** | 返回客户版本信息。 |
| **mysql\_get\_host\_info()** | 返回一个描述连接的字符串。 |
| **mysql\_get\_proto\_info()** | 返回连接使用的协议版本。 |
| **mysql\_get\_server\_info()** | 返回服务器版本号。 |
| **mysql\_info()** | 返回关于最近执行得查询的信息。 |
| **mysql\_init()** | 获得或初始化一个MYSQL结构。 |
| **mysql\_insert\_id()** | 返回有前一个查询为一个AUTO\_INCREMENT列生成的ID。 |
| **mysql\_kill()** | 杀死一个给定的线程。 |
| **mysql\_list\_dbs()** | 返回匹配一个简单的正则表达式的数据库名。 |
| **mysql\_list\_fields()** | 返回匹配一个简单的正则表达式的列名。 |
| **mysql\_list\_processes()** | 返回当前服务器线程的一张表。 |
| **mysql\_list\_tables()** | 返回匹配一个简单的正则表达式的表名。 |
| **mysql\_num\_fields()** | 返回一个结果集合重的列的数量。 |
| **mysql\_num\_rows()** | 返回一个结果集合中的行的数量。 |
| **mysql\_options()** | 设置对mysql\_connect()的连接选项。 |
| **mysql\_ping()** | 检查对服务器的连接是否正在工作，必要时重新连接。 |
| **mysql\_query()** | 执行指定为一个空结尾的字符串的SQL查询。 |
| **mysql\_real\_connect()** | 连接一个**MySQL**服务器。 |
| **mysql\_real\_query()** | 执行指定为带计数的字符串的SQL查询。 |
| **mysql\_reload()** | 告诉服务器重装授权表。 |
| **mysql\_row\_seek()** | 搜索在结果集合中的行，使用从mysql\_row\_tell()返回的值。 |
| **mysql\_row\_tell()** | 返回行光标位置。 |
| **mysql\_select\_db()** | 连接一个数据库。 |
| **mysql\_shutdown()** | 关掉数据库服务器。 |
| **mysql\_stat()** | 返回作为字符串的服务器状态。 |
| **mysql\_store\_result()** | 检索一个完整的结果集合给客户。 |
| **mysql\_thread\_id()** | 返回当前线程的ID。 |
| **mysql\_use\_result()** | 初始化一个一行一行地结果集合的检索。 |

int [mysql](https://www.baidu.com/s?wd=mysql&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3mHb1P16sm1uhuWcLujcd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjR3rHT3P1f1)\_query([MYSQL](https://www.baidu.com/s?wd=MYSQL&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3mHb1P16sm1uhuWcLujcd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjR3rHT3P1f1) \*mysql, const char \*query)   
描述  
执行由“Null终结的字符串”查询指向的SQL查询。正常情况下，字符串必须包含1条[SQL语句](https://www.baidu.com/s?wd=SQL%E8%AF%AD%E5%8F%A5&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3mHb1P16sm1uhuWcLujcd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjR3rHT3P1f1)，而且不应为语句添加终结分号（‘;’）或“\g”。如果允许多语句执行，字符串可包含多条由分号隔开的语句。  
mysql\_query()不能用于包含二进制数据的查询，应使用mysql\_real\_query()取而代之（二进制数据可能包含字符‘\0’，mysql\_query()会将该字符解释为查询字符串结束）。  
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。  
返回值  
如果查询成功，返回0。如果出现错误，返回非0值。

[mysql事务处理用法与实例详解](http://www.cnblogs.com/ymy124/p/3718439.html)

 MySQL的事务支持不是绑定在MySQL服务器本身，而是与存储引擎相关1.MyISAM：不支持事务，用于只读程序提高性能 2.InnoDB：支持ACID事务、行级锁、并发 3.Berkeley DB：支持事务

**一个事务是一个连续的一组**[**数据库**](http://www.111cn.net/database/database.html)**操作，就好像它是一个单一的工作单元进行。换言之，永远不会是完整的事务，除非该组内的每个单独的操作是成功的。如果在事务的任何操作失败，则整个事务将失败。**

实际上，会俱乐部许多SQL查询到一个组中，将执行所有的人都一起作为事务的一部分。

事务的特性：   
事务有以下四个标准属性的缩写ACID，通常被称为：

原子性: 确保工作单元内的所有操作都成功完成，否则事务将被中止在故障点，和以前的操作将回滚到以前的状态。

一致性: 确保[数据库](http://www.111cn.net/list-55/)正确地改变状态后，成功提交的事务。

隔离性: 使事务操作彼此独立的和透明的。

持久性: 确保提交的事务的结果或效果的系统出现故障的情况下仍然存在。

在MySQL中，事务开始使用COMMIT或ROLLBACK语句开始工作和结束。开始和结束语句的SQL命令之间形成了大量的事务。

COMMIT & ROLLBACK:   
这两个关键字提交和回滚主要用于MySQL的事务。

当一个成功的事务完成后，发出COMMIT命令应使所有参与表的更改才会生效。

如果发生故障时，应发出一个ROLLBACK命令返回的事务中引用的每一个表到以前的状态。

可以控制的事务行为称为AUTOCOMMIT设置会话变量。如果AUTOCOMMIT设置为1（默认值），然后每一个SQL语句（在事务与否）被认为是一个完整的事务，并承诺在默认情况下，当它完成。 AUTOCOMMIT设置为0时，发出SET AUTOCOMMIT =0命令，在随后的一系列语句的作用就像一个事务，直到一个明确的COMMIT语句时，没有活动的提交。

MYSQL的事务处理主要有两种方法

举例见http://www.cnblogs.com/ymy124/p/3718439.html  
1.用begin,rollback,commit来实现  
    begin开始一个事务  
    rollback事务回滚  
    commit 事务确认  
2.直接用set来改变mysql的自动提交模式  
    mysql默认是自动提交的，也就是你提交一个query，就直接执行！可以通过  
    set autocommit = 0 禁止自动提交  
    set autocommit = 1 开启自动提交

mysql\_real\_connect（）函数

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)  
上面描述了五个参数的主要取值，

MYSQL \*为mysql\_init函数返回的指针，

host为null或 localhost时链接的是本地的计算机，

当mysql默认安装在unix（或类unix）系统中，root账户是没有密码的，因此用户名使用root，密码为null，

当db为空的时候，函数链接到默认数据库，在进行 mysql安装时会存在默认的test数据库，因此此处可以使用test数据库名称，

port端口为0，

使用 unix连接方式，unix\_socket为null时，表明不使用socket或管道机制，最后一个参数经常设置为0  
mysql\_real\_connect()尝试与运行在主机上的MySQL数据库引擎建立连接。在你能够执行需要有效MySQL连接句柄结构的任何其他API函数之前，mysql\_real\_connect()必须成功完成。

**返回值**

如果连接成功，返回MYSQL\*连接句柄。如果连接失败，返回NULL。对于成功的连接，返回值与第1个参数的值相同。

mysql\_query() 函数

作用：执行一条 MySQL 查询。

**语法**

mysql\_query(query,connection)

|  |  |
| --- | --- |
| **参数** | **描述** |
| query | 必需。规定要发送的 SQL 查询。注释：查询字符串不应以分号结束。 |
| connection | 可选。规定 SQL 连接标识符。如果未规定，则使用上一个打开的连接。 |

**说明**

如果没有打开的连接，本函数会尝试无参数调用 mysql\_connect() 函数来建立一个连接并使用之。

**返回值**

mysql\_query() 仅对 SELECT，SHOW，EXPLAIN 或 DESCRIBE 语句返回一个资源标识符，如果查询执行不正确则返回 FALSE。

对于其它类型的 SQL 语句，mysql\_query() 在执行成功时返回 TRUE，出错时返回 FALSE。

非 FALSE 的返回值意味着查询是合法的并能够被服务器执行。这并不说明任何有关影响到的或返回的行数。很有可能一条查询执行成功了但并未影响到或并未返回任何行。

# mysql\_use\_result

对于成功检索数据的每个查询（SELECT、SHOW、DESCRIBE、EXPLAIN），必须调用mysql\_store\_result()或mysql\_use\_result()。mysql\_use\_result()将初始化[结果集](https://baike.baidu.com/item/%E7%BB%93%E6%9E%9C%E9%9B%86)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](https://baike.baidu.com/item/%E4%B8%B4%E6%97%B6%E8%A1%A8)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](https://baike.baidu.com/item/%E5%90%8C%E6%AD%A5%E9%94%99%E8%AF%AF)，如果忘记了执行该操作，将不能运行该命令。不应与从mysql\_use\_result()返回的结果一起使用[mysql\_data\_seek](https://baike.baidu.com/item/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](https://baike.baidu.com/item/%E7%BB%93%E6%9E%9C%E9%9B%86)的操作，必须调用mysql\_free\_result()。

使用libmysqld[嵌入式](https://baike.baidu.com/item/%E5%B5%8C%E5%85%A5%E5%BC%8F)服务器时，由于在调用mysql\_free\_result()之前，内存使用将随着每个检索的行增加，内存效益将基本丧失。

系统软件

计算机的软件分为系统软件和应用软件。系统软件是为了计算机能正常、高效工件所配备的各种管理、监控和维护系统的程序及其有关资料。系统软件主要包括如下几个方面：   
（1） [操作系统](https://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmhwhPhf4mHuBuj64uymY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHmknWbzP1fY)软件，这是软件的核心   
（2） 各种语言的[解释程序](https://www.baidu.com/s?wd=%E8%A7%A3%E9%87%8A%E7%A8%8B%E5%BA%8F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmhwhPhf4mHuBuj64uymY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHmknWbzP1fY)和编译程序（如[BASIC语言](https://www.baidu.com/s?wd=BASIC%E8%AF%AD%E8%A8%80&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmhwhPhf4mHuBuj64uymY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHmknWbzP1fY)[解释程序](https://www.baidu.com/s?wd=%E8%A7%A3%E9%87%8A%E7%A8%8B%E5%BA%8F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmhwhPhf4mHuBuj64uymY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHmknWbzP1fY)等）   
（3） 各种服务性程序（如机器的调试、故障检查和诊断程序等）   
（4） 各种数据库管理系统（FoxPro等）

**一、什么是主键、外键：**

关系型数据库中的一条记录中有若干个属性，若其中某一个属性组(注意是组)能唯一标识一条记录，该属性组就可以成为一个主键

比如

学生表(学号，姓名，性别，班级)

其中每个学生的学号是唯一的，学号就是一个主键

课程表(课程编号,课程名,学分)

其中课程编号是唯一的,课程编号就是一个主键

成绩表(学号,课程号,成绩)

成绩表中单一一个属性无法唯一标识一条记录，学号和课程号的组合才可以唯一标识一条记录，所以 学号和课程号的属性组是一个主键

成绩表中的学号不是成绩表的主键，但它和学生表中的学号相对应，并且学生表中的学号是学生表的主键，则称成绩表中的学号是学生表的外键

同理 成绩表中的课程号是课程表的外键

定义主键和外键主要是为了维护关系数据库的完整性，总结一下：

主键是能确定一条记录的唯一标识，比如，一条记录包括身份证号，姓名，年龄。身份证号是唯一能确定你这个人的，其他都可能有重复，所以，身份证号是主键。

外键用于与另一张表的关联。是能确定另一张表记录的字段，用于保持数据的一致性。比如，A表中的一个字段，是B表的主键，那他就可以是A表的外键。**二、  主键、外键和索引的区别**[收藏](http://blog.csdn.net/u011995790/article/details/11894885)

主键、外键和索引的区别？

|  |  |  |  |
| --- | --- | --- | --- |
|  | 主键 | 外键 | 索引 |
| 定义： | 唯一标识一条记录，不能有重复的，不允许为空 | 表的外键是另一表的主键, 外键可以有重复的, 可以是空值 | 该字段没有重复值，但可以有一个空值 |
| 作用： | 用来保证数据完整性 | 用来和其他表建立联系用的 | 是提高查询排序的速度 |
| 个数： | 主键只能有一个 | 一个表可以有多个外键 | 一个表可以有多个惟一索引 |

主键与外键

主键：

关系数据库依赖于主键---它是数据库物理模式的基石。主键在物理层面上只有两个用途：

      1. 惟一地标识一行。

      2. 作为一个可以被外键有效引用的对象。

基于以上这两个用途，下面给出了我在设计物理层面的主键时所遵循的一些原则：

      1. 主键应当是对用户没有意义的。如果用户看到了一个表示多对多关系的连接表中的数据，并抱怨它没有什么用处，那就证明它的主键设计地很好。

      2. 主键应该是单列的，以便提高连接和筛选操作的效率。

     注：使用复合键的人通常有两个理由为自己开脱，而这两个理由都是错误的。其一是主键应当具有实际意义，然而，让主键具有意义只不过是给人为地破坏数据库提供了方便。其二是利用这种方法可以在描述多对多关系的连接表中使用两个外部键来作为主键，我也反对这种做法，理由是：复合主键常常导致不良的外键，即当连接表成为另一个从表的主表，而依据上面的第二种方法成为这个表主键的一部分，然，这个表又有可能再成为其它从表的主表，其主键又有可能成了其它从表主键的一部分，如此传递下去，越靠后的从表，其主键将会包含越多的列了。

      3. 永远也不要更新主键。实际上，因为主键除了惟一地标识一行之外，再没有其他的用途了，所以也就没有理由去对它更新。如果主键需要更新，则说明主键应对用户无意义的原则被违反了。

     注：这项原则对于那些经常需要在数据转换或多数据库合并时进行数据整理的数据并不适用。

      4. 主键不应包含动态变化的数据，如时间戳、创建时间列、修改时间列等。

      5. 主键应当有计算机自动生成。如果由人来对主键的创建进行干预，就会使它带有除了惟一标识一行以外的意义。一旦越过这个界限，就可能产生认为修改主键的动机，这样，这种系统用来链接记录行、管理记录行的关键手段就会落入不了解数据库设计的人的手中。

外键

外键是数据库一级的一个完整性约束，就是数据库基础理论书中所说的“参照完整性”的数据库实现方式。

外键属性当然是可以去掉的，如果你不想再用这种约束，对编程当然不会有什么影响，但相应的录入数据的时候就不对录入的数据进行“参照完整性”检查了。

例如有两个表

A(a,b) :a为主键，b为外键（来自于B.b)

B(b,c,d) :b为主键

如果我把字段b的外键属性去掉，对编程没什么影响。

如上面，A中的b要么为空，要么是在B的b中存在的值，有外键的时候，数据库会自动帮你检查A的b是否在B的b中存在。

1、外建表达的是参照完整性：这是数据固有的，与程序无关。因此，应该交给DBMS来做。

2、使用外建，简单直观，可以直接在数据模型中体现，无论是设计、维护等回有很大的好处，特别是对于分析现有的数据库的好处时非常明显的--前不久我分析了一个企业现有的数据库，里面的参照完整性约束有的是外键描述，有的是用触发器实现，感觉很明显。当然，文档里可能有，但是也可能不全，但是外键就非常明显和直观。

3、既然我们可以用触发器或程序完成的这个工作（指参照完整性约束），DBMS已经提供了手段，为什么我们要自己去做？而且我们做的应该说没有RDBMS做得好。实际上，早期的RDBMS并没有外键，现在都有了，我认为数据库厂商增加这个功能是有道理的。从这个角度来说，外键更方便。

4、关于方便，根据我带项目的情况来看，程序员确实有反映，主要是在调试时输入数据麻烦：如果数据可以违反参照完整性，那么就是说参照完整性本身就不对名誉业务冲突，此时也不应该用触发期货程序实现；否则，说明数据是错误的，根本就不应该进入数据库！而且，这也应该是测试系统的一个内容：阻止非法数据。实际上，前台程序应该对这种提交失败做出处理。数据是企业的而非程序的，储程序要尽量与数据分离，反之亦然。

最后说一下,建键几个原则:

1、 为关联字段创建外键。

2、 所有的键都必须唯一。

3、避免使用复合键。

4、外键总是关联唯一的键字段。

一个查询机箱的例子

#include <mysql/mysql.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(){

MYSQL \*mysql;

MYSQL\_RES \*res;

MYSQL\_ROW row;

char \*query;

int t,q,r;

mysql=mysql\_init(NULL); //初始化数据库

if (!mysql\_real\_connect(mysql,"localhost","wcdj","123456","MemorySourceDb",0,NULL,0))

{

printf( "Error connecting to database: %s/n",mysql\_error(mysql));

}

else printf("Connected.../n");

query="select \* from case\_info; ";

t=mysql\_real\_query(mysql,query,(unsigned int) strlen(query)); //查询成功返回0

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++) //这个貌似也行，返回刚才查询的列数

//for(r=0;r<=mysql\_num\_rows(mysql);r++) //遍历查询的行数

{

row=mysql\_fetch\_row(res);

if(row<0) break;

for(q=0;q<mysql\_num\_fields(res);q++) //返回查询字段的列数

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

}

} ;

\*/此代码可以写成下面的形式

while(row=mysql\_fetch\_rows(res)) //一行一行的取

{ for(q=0;q<mysql\_num\_fields(res);q++)

{ printf(“%s\t”,row[q]);

} printf(“\n”);

}

mysql\_free\_result(res);

mysql\_close(mysql);

return 0;

}

Linux下编译 连接mysql服务器(库) mysql的库文件的路径

gcc -o dbtest dbtest.c -lmysqlclient -L/usr/lib64/mysql

数据库创建失败解决办法

创建数据库出现以下问题：

MariaDB [(none)]> create database w1;

ERROR 1044 (42000): Access denied for user 'wcdj'@'localhost' to database 'w1'

Solution：[root@localhost mysql]# cd /etc

加入代码（红色部分）

[root@localhost etc]# vi my.cnf

[mysqld]

datadir=/var/lib/mysql

socket=/var/lib/mysql/mysql.sock

skip-external-locking

skip-name-resolve

skip-grant-tables

# Disabling symbolic-links is recommended to prevent assorted security risks

symbolic-links=0

# Settings user and group are ignored when systemd is used.

# If you need to run mysqld under a different user or group,

# customize your systemd unit file for mariadb according to the

# instructions in http://fedoraproject.org/wiki/Systemd

[mysqld\_safe]

log-error=/var/log/mariadb/mariadb.log

pid-file=/var/run/mariadb/mariadb.pid

然后重启数据库[root@localhost etc]# systemctl restart mariadb.service

information\_schema数据库

information\_schema数据库是MySQL自带的，它提供了访问数据库元数据的方式。什么是元数据呢？元数据是关于数据的数据，如数据库名或表名，列的数据类型，或访问权限等。有些时候用于表述该信息的其他术语包括“数据词典”和“系统目录”。  
在MySQL中，把 information\_schema 看作是一个数据库，确切说是信息数据库。其中保存着关于MySQL服务器所维护的所有其他数据库的信息。如数据库名，数据库的表，表栏的数据类型与访问权 限等。在INFORMATION\_SCHEMA中，有数个只读表。它们实际上是视图，而不是基本表，因此，你将无法看到与之相关的任何文件。、

MySQL中的存储引擎

在MySQL中的存储引擎有很多种，可以通过“SHOW ENGINES”语句来查看。下面重点关注InnoDB、MyISAM、MEMORY这三种。

**一.InnoDB存储引擎**

 InnoDB给MySQL的表提供了**事务处理**、**回滚**、**崩溃修复能力**和**多版本并发控制**的事务安全。在MySQL从3.23.34a开始包含InnnoDB。它是MySQL上第一个提供外键约束的表引擎。而且InnoDB对事务处理的能力，也是其他存储引擎不能比拟的。靠后版本的MySQL的默认存储引擎就是InnoDB。

InnoDB存储引擎总支持**AUTO\_INCREMENT**。自动增长列的值不能为空，并且值必须唯一。MySQL中规定自增列必须为主键。在插入值的时候，如果自动增长列不输入值，则插入的值为自动增长后的值；如果输入的值为0或空（NULL），则插入的值也是自动增长后的值；如果插入某个确定的值，且该值在前面没有出现过，就可以直接插入。

InnoDB还支持**外键（FOREIGN KEY）**。外键所在的表叫做子表，外键所依赖（REFERENCES）的表叫做父表。父表中被字表外键关联的字段必须为主键。当删除、更新父表中的某条信息时，子表也必须有相应的改变，这是数据库的**参照完整性规则**。

InnoDB中，创建的表的表结构存储在*.frm*文件中（我觉得是frame的缩写吧）。数据和索引存储在innodb\_data\_home\_dir和innodb\_data\_file\_path定义的表空间中。

 InnoDB的优势在于提供了良好的事务处理、崩溃修复能力和并发控制。缺点是读写效率较差，占用的数据空间相对较大。

**二.MyISAM存储引擎**

MyISAM是MySQL中常见的存储引擎，曾经是MySQL的默认存储引擎。MyISAM是基于ISAM引擎发展起来的，增加了许多有用的扩展。

 MyISAM的表存储成3个文件。文件的名字与表名相同。拓展名为*frm*、*MYD*、*MYI*。其实，frm文件存储表的结构；MYD文件存储数据，是MYData的缩写；MYI文件存储索引，是MYIndex的缩写。

 基于MyISAM存储引擎的表支持3种不同的存储格式。包括静态型、动态型和压缩型。其中，静态型是MyISAM的默认存储格式，它的字段是固定长度的；动态型包含变长字段，记录的长度不是固定的；压缩型需要用到myisampack工具，占用的磁盘空间较小。

MyISAM的优势在于占用空间小，处理速度快。缺点是不支持事务的完整性和并发性。

**三.MEMORY存储引擎**

 MEMORY是MySQL中一类特殊的存储引擎。它使用存储在内存中的内容来创建表，而且**数据全部放在内存中**。这些特性与前面的两个很不同。

 每个基于MEMORY存储引擎的表实际对应一个磁盘文件。该文件的文件名与表名相同，类型为*frm*类型。该文件中只存储表的结构。而其数据文件，都是存储在内存中，这样有利于数据的快速处理，提高整个表的效率。值得注意的是，服务器需要有足够的内存来维持MEMORY存储引擎的表的使用。如果不需要了，可以释放内存，甚至删除不需要的表。

 MEMORY默认使用哈希索引。速度比使用B型树索引快。当然如果你想用B型树索引，可以在创建索引时指定。

注意，MEMORY用到的很少，因为它是把数据存到内存中，如果内存出现异常就会影响数据。如果重启或者关机，所有数据都会消失。因此，基于MEMORY的表的生命周期很短，一般是一次性的。

**四.怎样选择存储引擎**

 在实际工作中，选择一个合适的存储引擎是一个比较复杂的问题。每种存储引擎都有自己的优缺点，不能笼统地说谁比谁好。



总结如下：

**InnoDB**：支持事务处理，支持外键，支持崩溃修复能力和并发控制。如果需要**对事务的完整性要求比较高**（比如银行），**要求实现并发控制**（比如售票），那选择InnoDB有很大的优势。如果需要**频繁的更新、删除**操作的数据库，也可以选择InnoDB，因为支持事务的提交（commit）和回滚（rollback）。

**MyISAM**：插入数据快，空间和内存使用比较低。如果表主要是**用于插入新记录和读出记录**，那么选择MyISAM能实现处理高效率。如果应用的完整性、并发性要求比 较低，也可以使用。

**MEMORY**：所有的数据都在内存中，数据的处理速度快，但是安全性不高。如果需要**很快的读写速度**，对数据的安全性要求较低，可以选择MEMOEY。它对表的大小有要求，不能建立太大的表。所以，这类数据库只使用在相对较小的数据库表。

注意，同一个数据库也可以使用多种存储引擎的表。如果一个表要求比较高的事务处理，可以选择InnoDB。这个数据库中可以将查询要求比较高的表选择MyISAM存储。如果该数据库需要一个用于查询的临时表，可以选择MEMORY存储引擎。

linux下mysql数据的导出和导入

**导出整个**[**数据库**](https://www.2cto.com/database/)**中的所有数据**

1、在linux命令行下输入：

|  |  |
| --- | --- |
|  | mysqldump -u userName -p  dabaseName  > fileName.sql |

fileName.sql最好加上路径名

**导出数据库中的某个表的数据**

|  |  |
| --- | --- |
|  | mysqldump -u userName -p  dabaseName tableName > fileName.sql |

**导出整个数据库中的所有的表结构**

在linux命令行下输入：

|  |  |
| --- | --- |
|  | mysqldump -u userName -p -d dabaseName  > fileName.sql |

注意：是加了-d

**导出整个数据库中某个表的表结构**

在linux命令行下输入：

|  |  |
| --- | --- |
|  | mysqldump -u userName -p -d dabaseName tableName > fileName.sql |

注意：是加了-d

**导入mysql方法1**

进入linux命令命令行下：

|  |  |
| --- | --- |
| 1  2  3 | mysql -uroot -p 回车  输入密码  source fileName.sql  注意fileName.sql要有路径名，例如：source /home/user/data/fileName.sql |

**导入mysql方法2**

进入linux命令命令行下：

|  |  |
| --- | --- |
| 1  2 | mysql -uroot -p database < fileName.sql  注意fileName.sql要有路径名 |

MYSQL数据库的常用指令

测试环境：mysql 5.0.45 【注：可以在mysql中通过mysql> SELECT VERSION();来查看数据库版本】 整理：leo

1. **连接MYSQL。**

格式： mysql -h主机地址 -u用户名 －p用户密码

1、连接到本机上的MYSQL。 首先打开DOS窗口，然后进入目录mysql/bin，再键入命令mysql -u root -p，回车后提示你输密码.注意用户名前可以有空格也可以没有空格，但是密码前必须没有空格，否则让你重新输入密码. 如果刚安装好MYSQL，超级用户root是没有密码的，故直接回车即可进入到MYSQL中了，MYSQL的提示符是： mysql> 2、连接到远程主机上的MYSQL。假设远程主机的IP为：110.110.110.110，用户名为root,密码为abcd123。则键入以下命令： mysql -h110.110.110.110 -u root -p 123; （注:u与root之间可以不用加空格，其它也一样） 3、退出MYSQL命令： exit （回车）

**二、修改密码。**

格式：mysqladmin -u用户名 -p旧密码 password 新密码 1、给root加个密码ab12。首先在DOS下进入目录mysql/bin，然后键入以下命令 mysqladmin -u root -password ab12 注：因为开始时root没有密码，所以-p旧密码一项就可以省略了。 2、再将root的密码改为djg345。 mysqladmin -u root -p ab12 password djg345

1. **增加新用户。**

（注意：和上面不同，下面的因为是MYSQL环境中的命令，所以后面都带一个分号作为命令结束符） 格式：grant select on 数据库.\* to 用户名@登录主机 identified by “密码” 1、增加一个用户test1密码为abc，让他可以在任何主机上登录，并对所有数据库有查询、插入、修改、删除的权限。首先用root用户连入MYSQL，然后键入以下命令： grant select,insert,update,delete on \*.\* to [email=test1@”%]test1@”%[/email]” Identified by “abc”; 但增加的用户是十分危险的，你想如某个人知道test1的密码，那么他就可以在internet上的任何一台电脑上登录你的mysql数据库并对你的数据可以为所欲为了，解决办法见2。

2、增加一个用户test2密码为abc,让他只可以在localhost上登录，并可以对数据库mydb进行查询、插入、修改、删除的操作（localhost指本地主机，即MYSQL数据库所在的那台主机）， 这样用户即使用知道test2的密码，他也无法从internet上直接访问数据库，只能通过MYSQL主机上的web页来访问了。 grant select,insert,update,delete on mydb.\* to [email=test2@localhost]test2@localhost[/email] identified by “abc”; 如果你不想test2有密码，可以再打一个命令将密码消掉。 grant select,insert,update,delete on mydb.\* to [email=test2@localhost]test2@localhost[/email] identified by “”; 下篇我是MYSQL中有关数据库方面的操作。注意：你必须首先登录到MYSQL中，以下操作都是在MYSQL的提示符下进行的，而且每个命令以分号结束。

**四、操作技巧**

1、如果你打命令时，回车后发现忘记加分号，你无须重打一遍命令，只要打个分号回车就可以了。 也就是说你可以把一个完整的命令分成几行来打，完后用分号作结束标志就OK。 2、你可以使用光标上下键调出以前的命令。

**五、显示命令**

1、显示当前数据库服务器中的数据库列表： mysql> SHOW DATABASES; 注意：mysql库里面有MYSQL的系统信息，我们改密码和新增用户，实际上就是用这个库进行操作。

2、显示数据库中的数据表： mysql> USE 库名； mysql> SHOW TABLES;

3、显示数据表的结构： mysql> DESCRIBE 表名;

4、建立数据库： mysql> CREATE DATABASE 库名;

5、建立数据表： mysql> USE 库名; mysql> CREATE TABLE 表名 (字段名 VARCHAR(20), 字段名 CHAR(1));

6、删除数据库： mysql> DROP DATABASE 库名;

7、删除数据表： mysql> DROP TABLE 表名；

8、将表中记录清空： mysql> DELETE FROM 表名;

9、显示表中的记录： mysql> SELECT \* FROM 表名;

10、往表中插入记录： mysql> INSERT INTO 表名 VALUES (”hyq”,”M”);

11、更新表中数据： mysql-> UPDATE 表名 SET 字段名1=’a',字段名2=’b’ WHERE 字段名3=’c';

12、用文本方式将数据装入数据表中： mysql> LOAD DATA LOCAL INFILE “D:/mysql.txt” INTO TABLE 表名;

13、导入.sql文件命令： mysql> USE 数据库名; mysql> SOURCE d:/mysql.sql;

14、命令行修改root密码： mysql> UPDATE mysql.user SET password=PASSWORD(’新密码’) WHERE User=’root’; mysql> FLUSH PRIVILEGES;

15、显示use的数据库名： mysql> SELECT DATABASE();

16、显示当前的user： mysql> SELECT USER();

**六、一个建库和建表以及插入数据的实例**

drop database if exists school; //如果存在SCHOOL则删除 create database school;           //建立库SCHOOL use school;   //打开库SCHOOL create table teacher //建立表TEACHER ( id int(3) auto\_increment not null primary key, name char(10) not null, address varchar(50) default ‘深圳’, year date ); //建表结束

//以下为插入字段 insert into teacher values(”,’allen’,'大连一中’,'1976-10-10′); insert into teacher values(”,’jack’,'大连二中’,'1975-12-23′);

**如果你在mysql提示符键入上面的命令也可以，但不方便调试。** （1）你可以将以上命令原样写入一个文本文件中，假设为school.sql，然后复制到c://下，并在DOS状态进入目录[url=file:////mysql//bin]//mysql//bin[/url]，然后键入以下命令： mysql -uroot -p密码 < c://school.sql 如果成功，空出一行无任何显示；如有错误，会有提示。（以上命令已经调试，你只要将//的注释去掉即可使用）。 （2）或者进入命令行后使用 mysql> source c://school.sql; 也可以将school.sql文件导入数据库中。

**七、将文本数据转到数据库中**

1、文本数据应符合的格式：字段数据之间用tab键隔开，null值用[url=file:////n]//n[/url]来代替.例： 3 rose 大连二中 1976-10-10 4 mike 大连一中 1975-12-23 假设你把这两组数据存为school.txt文件，放在c盘根目录下。 2、数据传入命令 load data local infile “c://school.txt” into table 表名; 注意：你最好将文件复制到[url=file:////mysql//bin]//mysql//bin[/url]目录下，并且要先用use命令打表所在的库。

**八、备份数据库**

1.导出整个数据库 导出文件默认是存在mysql/bin目录下 mysqldump -u 用户名 -p 数据库名 > 导出的文件名 mysqldump -u user\_name -p123456 database\_name > outfile\_name.sql

2.导出一个表 mysqldump -u 用户名 -p 数据库名 表名> 导出的文件名 mysqldump -u user\_name -p database\_name table\_name > outfile\_name.sql

3.导出一个数据库结构 mysqldump -u user\_name -p -d –add-drop-table database\_name > outfile\_name.sql -d 没有数据 –add-drop-table 在每个create语句之前增加一个drop table

4.带语言参数导出 mysqldump -uroot -p –default-character-set=latin1 –set-charset=gbk –skip-opt database\_name > outfile\_name.sql

**九、导入数据库**

例如：数据库名为 dbTest

首先，进入mysql

mysql -uroot -p123456

然后，创建数据库

create dbTest

exit

最后，导入数据库

mysql -uroot -p123456 dbTest < dbTest\_bk.sql

**十、清空表中的数据**

TRUNCATE [TABLE] *tbl\_name*

**数据库及表导出导入示例：**

导出数据库 mysqldump -u root -p123456 gameTop > gameTop\_db.sql  
导出数据库的表 mysqldump -u -p123456 root gameTop gametop800 > gameTop\_table.sql  
导出数据库的特定表 mysqldump -u root -p123456  gameTop --table gametop800  > gameTop\_table.sql  
  
导出数据库（只导结构，不要数据） mysqldump -u root -p123456 --opt -d gameTop > gameTop\_db.sql

**或者**

mysqldump -u root -p123456  -d gameTop > gameTop\_db.sql 导出数据库（只导数据，不要结构） mysqldump -u root -p123456  -t gameTop > gameTop\_db.sql  
**注：**不加 -d 和 -t 则既导出结构，也导出数据

**mysqldump导出抛出异常：**

mysqldump: Got error: 1044: Access denied for user 'username'@'%' to database 'dbname' when using LOCK TABLES

解决办法，添加参数 **--skip-lock-tables**：

mysqldump **--skip-lock-tables** -h172.88.12.102 -username-pAnJnVs3C2tYXyTwV dbname> dbname\_bk.sql

-------------------------- **导入数据库：**  
登录MySQL:      mysql -uroot -p123456 创建数据库：    create database gameTop;

导入数据库：

mysql -uroot -p123456 gameTop < gameTop\_db.sql

导入数据库表：

mysql -uroot -p123456 gameTop  gametop800 <  gameTop\_table.sql

导入数据库表： mysql -uroot -p123456 gameTop  < gameTop\_table.sql   （不指定表名）

**MemorySourceDb数据库中表的建立**

**Cd /home/yeguanwen/mysql/create\_memorydb.sql**

/\*create database\*//\*此代码为数据库的表结构的创建代码\*/

create database MemorySourceDb;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\*create shelf\_info table\*/

use MemorySourceDb;/\***创建数据库时只有not null的字段才能设置为主键,not null的字段必须插入值**

MEMORY存储引擎的表的生命周期很短，一般是一次性的，重启或关机数据会消失\*/

create table shelf\_info( shelf\_id SMALLINT unsigned not null PRIMARY KEY, area\_id SMALLINT, state\_code tinyint)engine=Memory;

create table case\_info( case\_id char(16) not null, shelf\_id SMALLINT unsigned, mch\_addr char(16) not null, state\_code enum('NORMAL','ALARM') not null, vol TINYINT unsigned, temp TINYINT unsigned, primary key (case\_id))engine=Memory;

create table board\_state( case\_id char(16) not null references shelf\_info(case\_id), amc\_slot TINYINT unsigned not null, brd\_type enum('PPU','BBU','CPRI','NONE') not null, brd\_id char(18) , brd\_state enum('OUTAGE','SLEEP','NORMAL','ALARM'), alarm\_code TINYINT unsigned, primary key(case\_id,amc\_slot))engine=Memory;

create table ppu\_board( brd\_id char(18) not null PRIMARY KEY, control\_addr char(16) not null, state\_code enum('OUTAGE','SLEEP','NORMAL','ALARM') not null, case\_id char(16) not null references shelf\_info(case\_id), amc\_slot TINYINT unsigned not null, brd\_vol TINYINT unsigned not null, cpu\_temp TINYINT unsigned not null, cpu\_usg TINYINT unsigned not null, mem\_usg TINYINT unsigned not null, alarm\_code TINYINT unsigned, brd\_load TINYINT unsigned not null, ker\_usg TINYINT unsigned not null, remain\_ker TINYINT unsigned not null)engine=Memory;

create table bbu\_board( brd\_id char(18) not null PRIMARY KEY, dsp1\_ctrl\_addr char(18), dsp2\_ctrl\_addr char(18), dsp1\_data\_addr TINYINT, dsp2\_data\_addr TINYINT, v6\_ctrl\_addr char(18), s6\_ctrl\_addr char(18), dsp1\_rapidio char(17), dsp2\_rapidio char(17), v6\_data\_addr TINYINT, s6\_data\_addr TINYINT, state\_code enum('OUTAGE','SLEEP','NORMAL','ALARM'), alarm\_code TINYINT unsigned, case\_id char(16) references shelf\_info(case\_id), amc\_slot TINYINT unsigned, brd\_vol TINYINT unsigned, brd\_temp TINYINT unsigned, cpu\_temp TINYINT unsigned, dsp\_usg TINYINT unsigned)engine=Memory;create table cpri\_board( brd\_id char(18) not null primary key, control\_addr char(16), alarm\_code TINYINT unsigned, state\_code enum('OUTAGE','SLEEP','NORMAL','ALARM'), case\_id char(16) references shelf\_info(case\_id) , amc\_slot TINYINT unsigned, brd\_vol TINYINT unsigned, brd\_temp TINYINT unsigned, cpu\_temp TINYINT unsigned, fre\_0 SMALLINT unsigned, fre\_1 SMALLINT unsigned, fre\_2 SMALLINT unsigned, fre\_3 SMALLINT unsigned)engine=Memory;

create table vbs\_info( vbs\_id SMALLINT unsigned not null , ppu\_board char(18) references ppu\_board(brd\_id), ppu\_mask TINYINT unsigned, ppu\_num TINYINT unsigned, bbu\_board char(18) references bbu\_board(brd\_id), bbu\_mask TINYINT(8) unsigned, state enum('WAIT','WORK') not null, primary key(vbs\_id,state))engine=Memory;

create table cell\_info( vbs\_id SMALLINT unsigned not null PRIMARY KEY, cell\_id SMALLINT unsigned not null, rru\_id SMALLINT unsigned not null, vbs\_mode TINYINT, vbs\_user\_request INT UNSIGNED, vbs\_bandwidth TINYINT, ppu\_case\_id char(16), ppu\_amc\_slot TINYINT, ppu\_board\_id char(18) references ppu\_board(brd\_id), ppu\_cpu\_mask TINYINT unsigned, ppu\_cpu\_num TINYINT unsigned, ppu\_board\_ip char(16), vbs\_ip char(16), ppu\_pid INT unsigned, ppu\_memory\_size INT, ppu\_memory\_addr INT UNSIGNED, bbu\_case\_id char(16), bbu\_amc\_slot TINYINT, bbu\_board\_id char(18) references bbu\_board(brd\_id), s6\_ctrl\_addr char(18), bbu\_dsp\_mask TINYINT unsigned, access\_num INT UNSIGNED, active\_num INT UNSIGNED, packet\_lost\_rate INT UNSIGNED, throughput INT unsigned)engine=Memory;

DELIMITER $/\*DELIMITER是分号的意思，mysql遇到”；”自动就执行，当前的意思是把分号换成$,程序遇到$才会运行\*/

create trigger CS\_infoShelf\_Idafter insert on case\_info for each rowbegindeclare c int;select count(\*) into cfrom shelf\_infowhere shelf\_id = new.shelf\_id;/\*创建触发器CS\_infoShelf\_Id 在对case\_info表插入数据后，shelf.id=new.shelf.id的意思为修改后的机架ID与修改前相同\*/

if c=0 thendelete from case\_info where shelf\_id = new.shelf\_id;end if;end$

create trigger BC\_infoState\_Idafter update on board\_state for each rowbegindeclare c int;if new.brd\_state = 'ALARM' thenupdate case\_info set state\_code = 'ALARM'where case\_id = old.case\_id;end if;

select count(\*) into cfrom case\_infowhere case\_id = new.case\_id;if c=0 thendelete from board\_state where case\_id = new.case\_id;end if;

end$

create trigger PB\_infoBrd\_Stateafter update on ppu\_board for each rowbeginupdate board\_stateset brd\_state = new.state\_codewhere brd\_id = old.brd\_id;end$

create trigger BB\_infoBrd\_Stateafter update on bbu\_board for each rowbeginupdate board\_stateset brd\_state = new.state\_codewhere brd\_id = old.brd\_id;end$

create trigger CB\_infoBrd\_Stateafter update on cpri\_board for each rowbeginupdate board\_stateset brd\_state = new.state\_codewhere brd\_id = old.brd\_id;end$

DELIMITER ;

启动sbs.sh进行数据库初始化

use MemorySourceDb

insert into shelf\_info(shelf\_id,area\_id,state\_code)values(001,100,1);

insert into case\_info(shelf\_id,case\_id,mch\_addr,state\_code,vol,temp)

values(001,"1000000","10.21.150.2","NORMAL",12,40);

insert into case\_info(shelf\_id,case\_id,mch\_addr,state\_code,vol,temp)

values(001,"1000001","10.21.150.1","NORMAL",12,40);

insert into board\_state(case\_id,amc\_slot,brd\_type)

values("1000000",1,"NONE");

insert into board\_state(case\_id,amc\_slot,brd\_type)

values("1000000",2,"NONE");

insert into board\_state(case\_id,amc\_slot,brd\_type)

values("1000000",3,"NONE");

insert into board\_state(case\_id,amc\_slot,brd\_type)

values("1000000",4,"NONE");

insert into board\_state(case\_id,amc\_slot,brd\_type)

values("1000000",5,"NONE");

insert into board\_state(case\_id,amc\_slot,brd\_type)

values("1000000",6,"NONE");

insert into board\_state(case\_id,amc\_slot,brd\_type)

values("1000000",7,"NONE");

insert into board\_state(case\_id,amc\_slot,brd\_type)

values("1000000",8,"NONE");

insert into board\_state(case\_id,amc\_slot,brd\_type)

values("1000000",9,"NONE");

insert into board\_state(case\_id,amc\_slot,brd\_type)

values("1000000",10,"NONE");

insert into ppu\_board(brd\_id,control\_addr,state\_code,case\_id,amc\_slot,brd\_vol,cpu\_temp,cpu\_usg,mem\_usg,brd\_load,ker\_usg,remain\_ker)values("00:80:82:1C:88:9E","10.21.100.11","NORMAL","1000000",1,12,42,24,3,120,0,10);

insert into ppu\_board(brd\_id,control\_addr,state\_code,case\_id,amc\_slot,brd\_vol,cpu\_temp,cpu\_usg,mem\_usg,brd\_load,ker\_usg,remain\_ker)values("00:80:82:1C:B6:25","10.21.100.60","NORMAL","1000000",2,12,42,24,3,120,0,10);

insert into ppu\_board(brd\_id,control\_addr,state\_code,case\_id,amc\_slot,brd\_vol,cpu\_temp,cpu\_usg,mem\_usg,brd\_load,ker\_usg,remain\_ker) values("00:80:82:1C:88:B7","10.21.100.30","NORMAL","1000000",3,12,42,24,3,120,0,10);

insert into ppu\_board(brd\_id,control\_addr,state\_code,case\_id,amc\_slot,brd\_vol,cpu\_temp,cpu\_usg,mem\_usg,brd\_load,ker\_usg,remain\_ker)values("00:80:82:1C:F2:6A","10.21.100.90","NORMAL","1000000",4,12,42,24,3,120,0,10);

insert into ppu\_board(brd\_id,control\_addr,state\_code,case\_id,amc\_slot,brd\_vol,cpu\_temp,cpu\_usg,mem\_usg,brd\_load,ker\_usg,remain\_ker)values("00:80:82:99:A9:5B","10.21.100.14","NORMAL","1000000",5,12,42,24,3,120,0,10);

insert into ppu\_board(brd\_id,control\_addr,state\_code,case\_id,amc\_slot,brd\_vol,cpu\_temp,cpu\_usg,mem\_usg,brd\_load,ker\_usg,remain\_ker)values("00:80:82:99:A8:CB","10.21.100.13","NORMAL","1000000",6,12,42,24,3,120,0,10);

insert into ppu\_board(brd\_id,control\_addr,state\_code,case\_id,amc\_slot,brd\_vol,cpu\_temp,cpu\_usg,mem\_usg,brd\_load,ker\_usg,remain\_ker)values("00:80:82:4B:DE:00","10.21.100.80","NORMAL","1000000",7,12,42,24,3,120,0,10);

insert into ppu\_board(brd\_id,control\_addr,state\_code,case\_id,amc\_slot,brd\_vol,cpu\_temp,cpu\_usg,mem\_usg,brd\_load,ker\_usg,remain\_ker)values("00:80:82:1C:F2:4F","10.21.100.110","NORMAL","1000000",8,12,42,24,3,120,0,10);

insert into ppu\_board(brd\_id,control\_addr,state\_code,case\_id,amc\_slot,brd\_vol,cpu\_temp,cpu\_usg,mem\_usg,brd\_load,ker\_usg,remain\_ker)values("00:80:82:1C:F2:10","10.21.100.21","NORMAL","1000000",9,12,42,24,3,120,0,10);

insert into ppu\_board(brd\_id,control\_addr,state\_code,case\_id,amc\_slot,brd\_vol,cpu\_temp,cpu\_usg,mem\_usg,brd\_load,ker\_usg,remain\_ker)values("00:80:82:99:A9:52","10.21.100.70","NORMAL","1000000",10,12,42,24,3,120,0,10);

insert into board\_state(case\_id,amc\_slot,brd\_type,brd\_id,brd\_state)

values("1000001",3,"BBU","1111111111111000","NORMAL");

insert into board\_state(case\_id,amc\_slot,brd\_type,brd\_id,brd\_state)

values("1000001",4,"BBU","1111111111111001","NORMAL");

insert into board\_state(case\_id,amc\_slot,brd\_type,brd\_id,brd\_state)

values("1000001",5,"BBU","1111111111111002","NORMAL");

insert into board\_state(case\_id,amc\_slot,brd\_type,brd\_id,brd\_state)

values("1000001",6,"BBU","1111111111111003","NORMAL");

insert into board\_state(case\_id,amc\_slot,brd\_type,brd\_id,brd\_state)

values("1000001",7,"BBU","1111111111111004","NORMAL");

insert into board\_state(case\_id,amc\_slot,brd\_type,brd\_id,brd\_state)

values("1000001",8,"BBU","1111111111111005","NORMAL");

insert into bbu\_board(brd\_id,dsp1\_ctrl\_addr,dsp2\_ctrl\_addr,v6\_ctrl\_addr,s6\_ctrl\_addr,dsp1\_rapidio,dsp2\_rapidio,state\_code,case\_id,amc\_slot,brd\_vol,brd\_temp,cpu\_temp,dsp\_usg)values("1111111111111000","FE:FF:FF:FF:FF:F1","FE:FF:FF:FF:FF:E1","FE:FF:FF:FF:FF:D1","FE:FF:FF:FF:FF:C1","1111111111111111","2111111111111111","NORMAL","1000001",3,12,0,0,0);

insert into bbu\_board(brd\_id,dsp1\_ctrl\_addr,dsp2\_ctrl\_addr,v6\_ctrl\_addr,s6\_ctrl\_addr,dsp1\_rapidio,dsp2\_rapidio,state\_code,case\_id,amc\_slot,brd\_vol,brd\_temp,cpu\_temp,dsp\_usg)values("1111111111111001","FE:FF:FF:FF:FF:F2","FE:FF:FF:FF:FF:E2","FE:FF:FF:FF:FF:D2","FE:FF:FF:FF:FF:C2","1111111111111112","2111111111111112","NORMAL","1000001",4,12,0,0,0);

insert into bbu\_board(brd\_id,dsp1\_ctrl\_addr,dsp2\_ctrl\_addr,v6\_ctrl\_addr,s6\_ctrl\_addr,dsp1\_rapidio,dsp2\_rapidio,state\_code,case\_id,amc\_slot,brd\_vol,brd\_temp,cpu\_temp,dsp\_usg)values("1111111111111002","FE:FF:FF:FF:FF:F3","FE:FF:FF:FF:FF:E3","FE:FF:FF:FF:FF:D3","FE:FF:FF:FF:FF:C3","1111111111111113","2111111111111113","NORMAL","1000001",5,12,0,0,0);

insert into bbu\_board(brd\_id,dsp1\_ctrl\_addr,dsp2\_ctrl\_addr,v6\_ctrl\_addr,s6\_ctrl\_addr,dsp1\_rapidio,dsp2\_rapidio,state\_code,case\_id,amc\_slot,brd\_vol,brd\_temp,cpu\_temp,dsp\_usg)values("1111111111111003","FE:FF:FF:FF:FF:F4","FE:FF:FF:FF:FF:E4","FE:FF:FF:FF:FF:D4","FE:FF:FF:FF:FF:C4","1111111111111114","2111111111111114","NORMAL","1000001",6,12,0,0,0);

insert into bbu\_board(brd\_id,dsp1\_ctrl\_addr,dsp2\_ctrl\_addr,v6\_ctrl\_addr,s6\_ctrl\_addr,dsp1\_rapidio,dsp2\_rapidio,state\_code,case\_id,amc\_slot,brd\_vol,brd\_temp,cpu\_temp,dsp\_usg)values("1111111111111004","FE:FF:FF:FF:FF:F5","FE:FF:FF:FF:FF:E5","FE:FF:FF:FF:FF:D5","FE:FF:FF:FF:FF:C5","1111111111111115","2111111111111115","NORMAL","1000001",7,12,0,0,0);

insert into bbu\_board(brd\_id,dsp1\_ctrl\_addr,dsp2\_ctrl\_addr,v6\_ctrl\_addr,s6\_ctrl\_addr,dsp1\_rapidio,dsp2\_rapidio,state\_code,case\_id,amc\_slot,brd\_vol,brd\_temp,cpu\_temp,dsp\_usg)values("1111111111111005","FE:FF:FF:FF:FF:F6","FE:FF:FF:FF:FF:E6","FE:FF:FF:FF:FF:D6","FE:FF:FF:FF:FF:C6","1111111111111116","2111111111111116","NORMAL","1000001",8,12,0,0,0);

**mysql\_num\_fields和mysql\_field\_count区别**

mysql\_num\_fields和mysql\_field\_count都是获取结果中列的个数，但是这两

个函数在使用的时候是存在区别的。

mysql\_field\_count()是返回刚才查询的栏数

先看一段代码（上篇文章提供的数据库）：

**[cpp]** [view plain](http://blog.csdn.net/bwmwm/article/details/6080160) [copy](http://blog.csdn.net/bwmwm/article/details/6080160)

1. #include <mysql/mysql.h>
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <string.h>
6. int main()
7. {
8. char out[100];
9. MYSQL mysql;
10. MYSQL\_RES\* res;
12. int num\_fields;
13. int count\_fields;
14. int num\_rows;
16. mysql\_init(&mysql);
17. mysql\_real\_connect(&mysql,NULL,"root","123","ta",0,NULL,0);
19. fprintf(stdout,"--insert------------------------/n");
21. mysql\_real\_query(&mysql,"INSERT INTO t1 VALUES(6,'n6')",29);
23. count\_fields = mysql\_field\_count(&mysql);
24. sprintf(out,"insert----count\_fields is %d/n",count\_fields);
25. fprintf(stdout,out);
27. res = mysql\_store\_result(&mysql);
28. if(res == NULL)
29. {
30. fprintf(stdout,"insert---store return null/n");
31. }
32. else
33. {
34. num\_fields = mysql\_num\_fields(res);
35. sprintf(out,"insert---num\_fields is %d/n",num\_fields);
36. fprintf(stdout,out);
37. }
39. fprintf(stdout,"---select-----------/n");
41. mysql\_real\_query(&mysql,"SELECT \* FROM t1",16);
43. count\_fields = mysql\_field\_count(&mysql);
44. sprintf(out,"select---count\_fields is %d/n",count\_fields);
45. fprintf(stdout,out);
47. res = mysql\_store\_result(&mysql);
48. if(res == NULL)
49. {
50. fprintf(stdout,"select---store return null/n");
51. }
52. else
53. {
54. num\_fields = mysql\_num\_fields(res);
55. sprintf(out,"select---num\_fields is %d/n",num\_fields);
56. fprintf(stdout,out);
58. num\_rows = mysql\_num\_rows(res);
59. sprintf(out,"select---num\_rows is %d/n",num\_rows);
60. fprintf(stdout,out);
61. }
63. fprintf(stdout,"---select 0------/n");
65. mysql\_real\_query(&mysql,"SELECT \* FROM t1 where id = 10",30);
67. count\_fields = mysql\_field\_count(&mysql);
68. sprintf(out,"select 0---count\_fields is %d/n",count\_fields);
69. fprintf(stdout,out);
71. res = mysql\_store\_result(&mysql);
72. if(res == NULL)
73. {
74. fprintf(stdout,"select 0---store return null/n");
75. }
76. else
77. {
78. num\_fields = mysql\_num\_fields(res);
79. sprintf(out,"select 0---num\_fields is %d/n",num\_fields);
80. fprintf(stdout,out);
82. num\_rows = mysql\_num\_rows(res);
83. sprintf(out,"select 0---num\_rows is %d/n",num\_rows);
84. fprintf(stdout,out);
85. }
87. mysql\_free\_result(res);
88. mysql\_close(&mysql);
89. exit(0);
90. }

执行结果是：  
--insert------------------------  
insert----count\_fields is 0  
insert---store return null  
---select-----------  
select---count\_fields is 2  
select---num\_fields is 2  
select---num\_rows is 7  
---select 0------  
select 0---count\_fields is 2  
select 0---num\_fields is 2  
select 0---num\_rows is 0

显然，当执行insert语句的时候，是没有结果返回的，因此列的个数为0，且mysql\_store\_result返回NULL。因此可以通过mysql\_field\_count()是否返回0来判断是否有结果返回，而不需要执行mysql\_store\_result来判断是否返回了NULL。我想，mysql\_field\_count()的效率肯定要比mysql\_store\_result()高。

在这种情况下，由于没有返回结果，因此mysql\_store\_result()返回NULL，也就是得不到res指针，于是mysql\_num\_fields()函数就无法执行，缺少必要的参数。

当执行第一条select语句的时候，返回了结果，因此mysql\_field\_count()和mysql\_num\_fields()都返回了正确的列的个数2，mysql\_num\_rows()返回了记录的条数7.

当执行第二条select语句，由于表中没有 id = 0 的记录，因此mysql\_num\_rows返回了0表示记录数为0，但是，我们发现mysql\_store\_result()并没有返回NULL，mysql\_num\_fields()和mysql\_field\_count()还是返回了2.

因此我们可以得出这样的结论：

执行结果有三种情况，第一是执行insert、update和delete这样的语句的时候，是不会有任何内容返回，因此mysql\_store\_result()会返回一个NULL。第二，执行select或show这样的语句时，一定会有内容返回，可以取得列信息，但是记录可以为0，也可以不为0。这就像一个表，表头一定存在，但是表中可以没有数据。

另外：

mysql\_affected\_rows()函数的作用是，当执行update  insert  delete 时，返回影响的行数。