Con Clase, http://www.conclase.net

Curso del API C de **MySQL** por Salvador Pozo Coronado

Prólogo

__MySQL__ es una marca registrada por __MySQL AB__. Parte del material que se expone aquí, concretamente las referencias de funciones del API de __MySQL__ y de la sintaxis de SQL, son traducciones del manual original de __MySQL__ que se puede encontrar en inglés en www.mysql.com.

Este sitio está desarrollado exclusivamente por los componentes de __Con Clase__, cualquier error en la presente documentación es sólo culpa nuestra.

Introducción

El presente manual trata sobre la integración de bases de datos **MySQL** dentro de aplicaciones C y C++. Para ello nos limitaremos a explicar algunas de las funciones y estructuras del API de **MySQL** y al modo en que estas se combinan para crear programas que trabajen con bases de datos.

Se supone que el lector de este documento ya tiene conocimientos sobre diseño de bases de datos y sobre el lenguaje de consulta SQL. De todos modos, en **Con Clase** también se incluye un curso sobre estos temas, al que se puede acceder desde MySQL con Clase.

También supondremos que trabajamos en un sistema que tiene instalado el servidor **MySQL**. Se puede consultar la documentación en la página de **MySQL** para obtener detalles sobre el modo de instalar este servidor en distintos sistemas operativos.

Instalación de librerías para Dev-C++

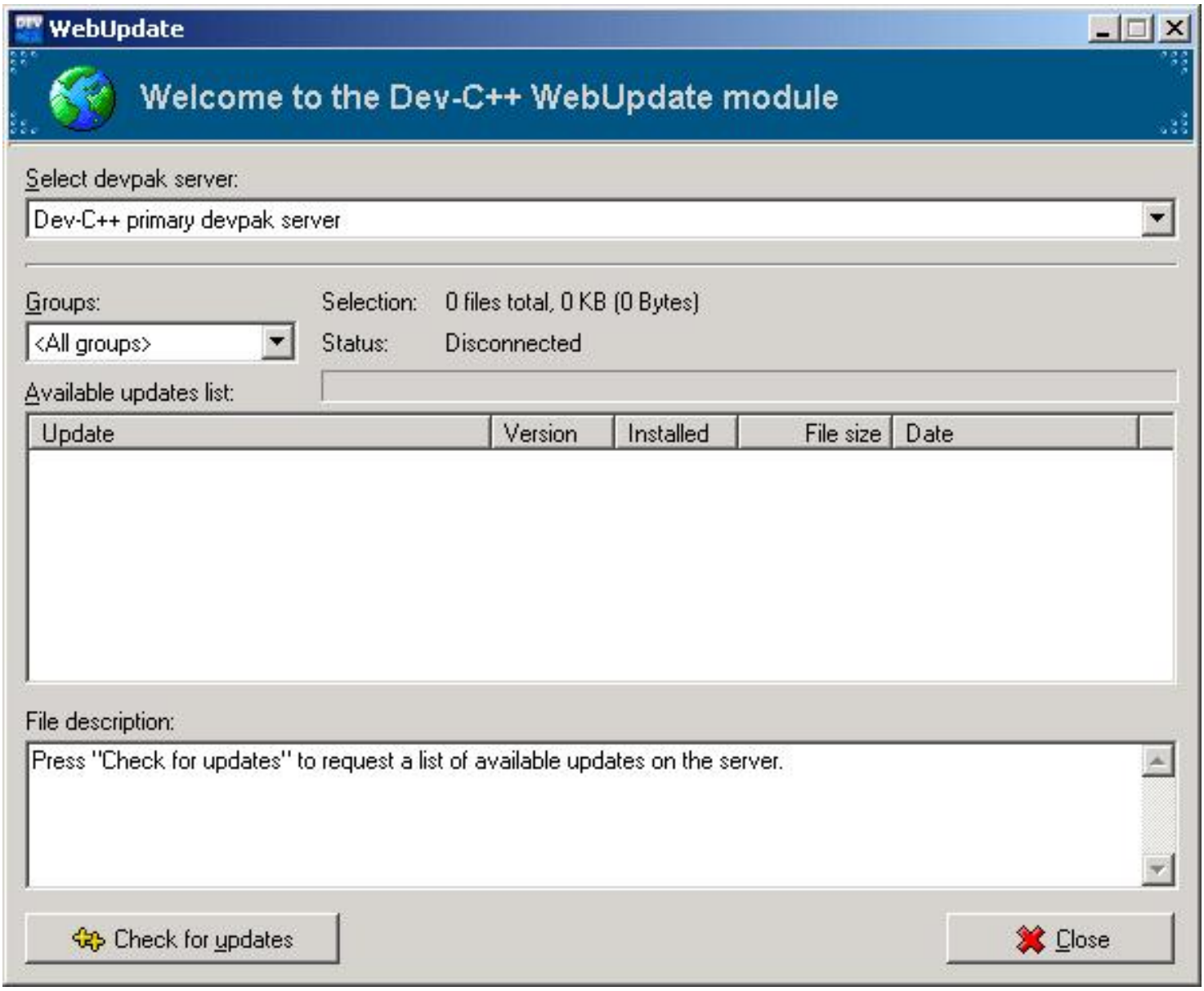
Lo que sí necesitamos es, por supuesto, un compilador de C/C++. Los ejemplos que aparecen en este curso están escritos en C++, pero usaremos el API C, de modo que las estructuras de los programas deben ser fácilmente adaptables a código C.

> **Nota:** Aunque existe un API C++, que encapsula el API C de **MySQL**, de momento no se incluye documentación sobre él en este curso.

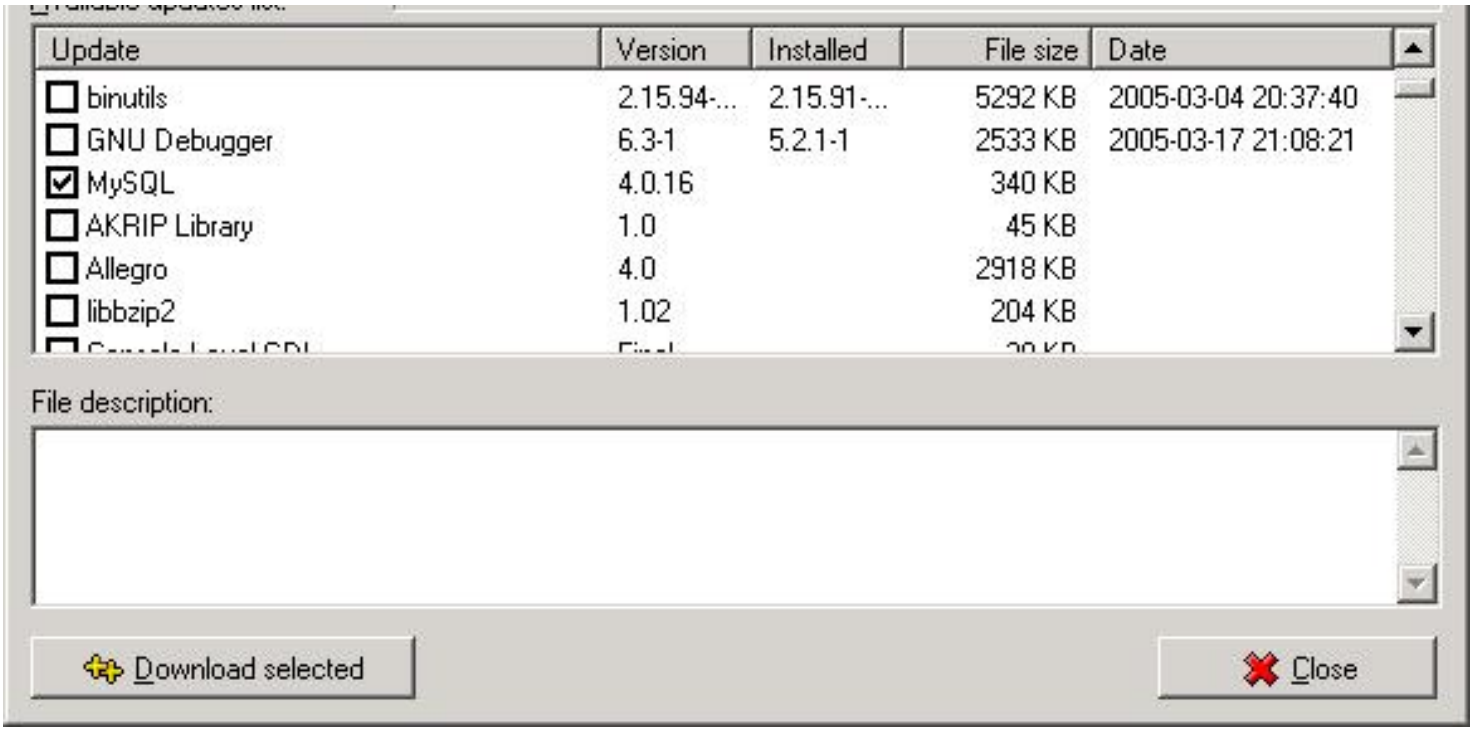
Siguiendo lo que ya es una tradición en **Con Clase** usaremos el compilador **Mingw** y el entorno de desarrollo **Dev-C++** para crear nuestros programas de ejemplo. En este entorno se puede descargar un paquete para usar el API de **MySQL**. Veamos cómo hacerlo paso a paso:

1. Usar la opción Ayuda->Sobre..., se mostrará este cuadro de diálogo:

2. Pulsar el botón "Buscar actualizaciones", se mostrará el cuadro de diálogo de actualizaciones:



3. Pulsar el botón "Check for updates". Se leerá una lista de los paquetes disponibles.
4. Seleccionar el paquete **MySQL** y pulsar el botón "Download selected":



5. Se descargará el paquete desde Internet, y aparecerá este mensaje: "The updates you selected have been downloaded. Now they will be installed.", es decir, que se han descargado las actualizaciones y se procederá a instalarlas. Pulsamos "Ok".
6. Se abrirá otra ventana, con el "Installation Wizard", pulsamos "Install>":

7. Cuando termine la instalación pulsamos "Finish", y se cerrará el "Installation Wizard".

Ahora tendremos un subdirectorio *MySQL* bajo el directorio *include* de *Dev-C++*, que contendrá los ficheros de cabecera ".h". En el directorio *lib* de **Dev-C++** se habrán copiado los ficheros "libmySQL.a" y "libmySQL.def". Y en el directorio de *examples* de **Dev-C++** habrá una carpeta con un proyecto de ejemplo *MySQLClientTest*, (que no funcionará :-), aunque esto no debe preocuparnos, ya que tampoco lo vamos a necesitar).

Usar ficheros incluidos con MySQL

Pero en realidad, aunque esto funcione, lo cierto es que junto con la instalación del servidor **MySQL** ya existen los ficheros de cabecera y las librerías necesarias para usar **MySQL** en nuestros programas C/C++.

El método que vamos a explicar tiene la ventaja de que usaremos las librerías adecuadas para la versión de **MySQL** que tenemos instalada. Si usamos el método del paquete de **Dev-C++** probablemente estaremos usando librerías de una versión anterior.

Los ficheros de cabecera están en el directorio *include* dentro del directorio de instalación de **MySQL**, que dependerá de dónde lo hayas instalado. Esos ficheros pueden ser copiados al directorio a un directorio *mysql* dentro del directorio *include* de **Dev-C++**.

Además, en el directorio *lib\opt* de **MySQL** existen varios ficheros de librería, y una librería de enlace dinámico *libmysql.dll*. Copiaremos esta librería a un directorio donde sea accesible, por ejemplo al directorio *system32* de **Windows**.

De los ficheros de librerías estáticas sólo nos interesa uno: *libmysql.lib*.

Pero **Mingw** no puede usar esa librería directamente. Este compilador usa otro formato de librerías estáticas, que además de tener la extensión ".a", tienen un formato interno diferente. Deberemos, por lo tanto, convertir la librería.

Para poder hacerlo necesitamos los siguientes ficheros:

- El fichero de librería dinámica "dll": *libmysql.dll*.
- El fichero de librería estática "lib": *libmysql.lib*.
- El fichero de definición de librería "def": *libmysql.def*.

Y estas tres utilidades:

- Utilidad *reimp.exe*.
- Utilidad *dlltool.exe*.
- Utilidad *as.exe*.

Los dos primeros ficheros ya los tenemos, los encontraremos en el directorio *lib\opt* de modo que para trabajar más fácilmente, los copiaremos a un directorio temporal de trabajo, por ejemplo, *C:\mysqltmp*.

Para obtener el tercero necesitamos una utilidad de **Mingw** llamada *reimp*. Esta utilidad sirve para extraer ficheros de definición de librerías "lib".

Para conseguir la herramienta *reimp* hay que descargar el fichero Utilidades mingw, que contiene algunas utilidades, pero del que sólo nos interesa el fichero *reimp.exe*.

Copiaremos el fichero *reimp.exe* al directorio de trabajo temporal, y ejecutamos esta sentencia desde la línea de comandos:

```
C:\mysqltmp> reimp -d libmysql.lib
```

Ahora ya tenemos el fichero *libmysql.def*. El siguiente paso es generar el fichero de librería estática *libmysql.a*. Para ello necesitamos las otras dos utilidades, *dlltool.exe* y *as.exe* que están incluidan con el compilador *Mingw*, y que podemos encontrar en el subdirectorio *mingw32\bin* del directorio donde esté instalado **Dev-C++**. Para comodidad, copiaremos estas utilidades al directorio temporal de trabajo, y ejecutaremos la siguiente sentencia desde la línea de comandos:

```
C:\mysqltmp>dlltool -d libmysql.def -D libmysql.dll -k -l libmysql.a
```

La utilidad *as.exe* se invoca automáticamente desde *dlltool*. Ahora ya tenemos nuestro fichero de librería estática *libmysql.a*, y lo copiaremos al directorio *lib* de **Dev-C++**.

Por último, podemos borrar el directorio de trabajo temporal. Y ya podemos usar bases de datos desde programas C/C++.

Este proceso es válido para convertir cualquier librería en formato "lib" a su equivalente en formato "a".

Usar librerías desde otros compiladores

No lo he verificado, pero es de suponer que otros compiladores usarán, o bien las librerías estáticas con extensión "a" si se basan en **Mingw**, o las que tienen extensión "lib", como los compiladores de **Borland** o **Microsoft**.

En cualquier caso, si quieres compartir tus experiencias en estos compiladores con nosotros, podremos ampliar este apartado.

1 Conectar y desconectar

Siempre, cada vez que queramos usar **MySQL** en una de nuestras aplicaciones, deberemos realizar algunas tareas antes y después de acceder al servidor. Esto es bastante corriente cuando se trabaja con "motores", ya sean gráficos, de bases de datos, o de lo que sea.

Primero hay que verificar que el motor está presente y en ejecución. Después hay que establecer una conexión o canal que sirva para mantener un diálogo con el servidor. A partir de ahí podremos entablar comunicación con ese servidor, y cuando hayamos terminado, cerrar la conexión.

Iniciar y conectar con el servidor MySQL

Lo primero es iniciar un objeto del tipo MYSQL. Este objeto se usa por el API para mantener las variables de la conexión con el motor de **MySQL**.

Para iniciar uno de estos objetos usamos la función mysql_init(). Esta función sólo necesita un parámetro, que es un puntero a un objeto de tipo MYSQL. Si ya disponemos de un objeto de ese tipo podemos pasar su dirección como parámetro, y la función lo iniciará. Si no disponemos de tal objeto, pasaremos un puntero nulo y la función lo creará dinámicamente y nos devolverá un puntero al objeto.

Por ejemplo, usando un objeto dinámico:

```
MYSQL *myData;

// Intentar iniciar MySQL:
if(!(myData = mysql_init(0))) {
   // Imposible crear el objeto myData
   return 1;
}
```

Y usando un objeto estático:

```
MYSQL myData;

// Iniciar MySQL:
mysql_init((MYSQL*)&myData));
```

Generalmente trabajaremos con un objeto MYSQL dinámico, al menos en los ejemplos de este curso.

Establecer una conexión

Una vez hemos inicializado el objeto MYSQL, intentaremos establecer una conexión con el servidor. Para ello usaremos la función mysql_real_connect().

Esta función necesita muchos parámetros, concretamente ocho. Veamos qué significa cada uno de ellos:

1. Un puntero a un objeto MYSQL, que previamente tendremos que haber iniciado con la función mysql_init().
2. El nombre del ordenador donde se está ejecutando el servidor. Puede ser también una dirección IP. Si el servidor está ejecutándose en la misma máquina que la aplicación, este nombre puede ser "localhost", o simplemente, un puntero nulo.
3. El nombre del usuario. En sistemas **Unix** puede ser un puntero nulo, para indicar el usuario actual. En **Windows ODBC** debe especificarse.
4. La contraseña del usuario seleccionado.
5. Base de datos por defecto. Puede ser NULL si no queremos usar una base de datos determinada.
6. Número de puerto. Generalmente usaremos la constante **MYSQL_PORT**.
7. Socket **Unix**. Usaremos NULL para conexiones locales.
8. Opciones de cliente, normalmente 0, pero se puede usar una combinación de ciertos valores para activar algunas opciones. Ver sintaxis de mysql_real_connect().

Si no se puede establecer una conexión, mysql_real_connect() devuelve el valor NULL.

Al menos al principio usaremos conexiones locales para acceder a las bases de datos, aunque más tarde veremos que no hay problema para hacer conexiones a través de una red local o a través de Internet.

para nuestros experimentos, que sólo tenga acceso total a la base de datos "prueba":

```
mysql> GRANT ALL ON prueba.* TO curso IDENTIFIED BY 'clave';
Query OK, 0 rows affected (0.16 sec)
```

Esto hace que los parámetros a usar sean más sencillos. Una llamada típica para acceso local puede ser:

```
   if(!mysql_real_connect(myData, NULL, "curso", "clave", "prueba",
MYSQL_PORT, NULL, 0)) {
```

```
    // No se puede conectar con el servidor en el puerto especificado.
    cout << "Imposible conectar con servidor mysql en el puerto "
         << MYSQL_PORT << endl;
    mysql_close(myData);
    return 1;
}
```

Cerrar

Cuando hayamos terminado de trabajar con las bases de datos cerramos la conexión con el motor de bases de datos, para eso usamos la función mysql_close().

```
// Cerrar la conexión
mysql_close(myData);
```

Cerrar la conexión tiene un doble propósito. Por una parte, se cierra la conexión abierta con el servidor mediante la función mysql_real_connect(). Por otra, se libera la memoria correspondiente al objeto MYSQL, si la función mysql_init() fue invocada con un puntero nulo.

Ahora ya estamos en condiciones de empezar a trabajar con bases de datos.

Reconexiones

Un problema frecuente cuando se trabaja con **MySQL** es que las conexiones abiertas que permanecen mucho tiempo inactivas pueden cerrarse de forma automática. Esto no es un defecto, sino algo habitual con conexiones de red. Las conexiones "débiles" (con poco tráfico) se cierran de forma automática pasado un tiempo determinado, para ahorrar recursos.

Esto pasa también con la consola de **MySQL**, pero al intentar nuevas consultas se realiza una reconexión automática. Esto no será así con nuestras aplicaciones, ya que si la conexión se cierra recibiremos un mensaje de error, y la conexión no se restablecerá.

En el API existe una función para verificar si la conexión sigue abierta y reconectar en caso necesario.

La función es mysql_ping(), y devuelve un valor cero si la conexión está activa.

```
    if(mysql_ping(&myData)) {
        cout << "Error: conexión imposible" << endl;
```

```
        mysql_close(&myData);
    }
```

Es recomendable hacer uso de esta función si existe la posibilidad de que la conexión haya estado inactiva durante mucho tiempo, por ejemplo, en aplicaciones **Windows** que pueden estar en segundo plano durante un tiempo indeterminado.

2 Consultas

Ahora que ya sabemos cómo establecer una conexión con el servidor **MySQL** y cómo cerrar esa conexión, veamos cómo podemos hacer consultas SQL desde nuestros programas C/C++.

Seleccionar una base de datos

Para acceder a la información almacenada en una base de datos, debemos hacer referencia a la base de datos mediante su nombre. En el cliente **MySQL** usamos la orden USE para seleccionar una base de datos por defecto.

Usando el API hemos visto que podemos seleccionar la base de datos en la llamada a la función mysql_real_connect(). Pero si queremos cambiar la base de datos por defecto durante la ejecución no será necesario cerrar la conexión y abrir una nueva, podemos usar en su lugar la función mysql_select_db():

```
   if(mysql_select_db(myData, "prueba")) {
      // Error al seleccionar base de datos.
      cout << "ERROR: " <<  mysql_error(myData) << endl;
      mysql_close(myData);
      rewind(stdin);
      getchar();
      return 2;
   }
```

La función mysql_select_db() hace que la base de datos especificada como segundo parámetro sea considerada la base de datos por defecto en las siguientes consultas. En rigor esto no es imprescindible, ya que podemos acceder a una tabla mediante el especificador de base de datos y el nombre de la tabla, pero si todas las consultas se van a hacer sobre la misma base de datos, esto nos ahorrará mucho trabajo (de escritura).

Si el valor de retorno de la función es distinto de cero, indica que se ha producido un error. En este ejemplo hemos usardo la función mysql_error para mostrar un mensaje de error que indique el motivo.

Esta función puede fallar por varios motivos, pero de momento sólo nos preocupa si el error es provocado porque la base de datos no existe, en ese caso obtendremos el error ER_BAD_DB_ERROR.

Seleccionar datos de una tabla

La primera operación que intentaremos sobre la base de datos es leer algunos registros de una tabla. Empezaremos por una consulta sencilla, leyendo todos los registros de la tabla 'gente'.

Nota: supondremos que tal tabla ya existe y que contiene datos, pero si no es así en tu caso, puedes crearla y añadir datos desde la consola siguiendo los pasos del curso de **MySQL**: , , .

La consulta que vamos a hacer es:

```
SELECT * FROM gente;
```

Necesitamos usar una sentencia SQL, en este caso una sentencia SELECT.

Para incluir consultas dentro de programas C/C++ se usan dos funciones del API C de **MySQL**, mysql_query o mysql_real_query. En ambos casos, primer parámetro es, como siempre, un manipulador de un objeto MYSQL iniciado; el segundo contiene la cadena con la consulta o instrucción SQL a ejecutar.

La función mysql_real_query usa un tercer parámetro, que el la longitud de la cadena que contiene la consulta. Esta función permite usar consultas binarias, es decir, que la cadena de la consulta puede contener caracteres nulos.

```
mysql_query(myData, "SELECT * FROM gente");
```

Veamos un ejemplo de cómo lanzar esta consulta:

```
   // Hacer una consulta con el comando "SELECT * FROM gente":
   if(mysql_query(myData, "SELECT * FROM gente")) {
      // Error al realizar la consulta:
      cout << "ERROR: " <<  mysql_error(myData) << endl;
      mysql_close(myData);
      rewind(stdin);
      getchar();
      return 2;
   }
```

Recuperar datos de una base de datos

Ya hemos hecho una consulta mediante SELECT, ahora mostraresmos el proceso para recuperar los resultados de esa consulta. Por cierto, otras consultas también generan conjuntos de resultados, como SHOW TABLES o SHOW DATABASES, y el tratamiento de esos resultados es el mismo.

Lo primero es recuperar el conjunto de resultados procedente del servidor **MySQL**, para ello se usa la función mysql_store_result.

La función mysql_store_result requiere como parámetro un objeto MYSQL, con la conexión actual y devuelve un puntero a una estructura MYSQL_RES.

Del mismo modo, una vez procesados los resultados, se deben liberar mediante una llamada a la función mysql_free_result. Esto liberará la memoria usada por el conjunto de resultados, es decir, por la estructura MYSQL_RES.

```
// Almacenar el resultado de la consulta, lo necesitaremos después:
if((res = mysql_store_result(myData))) {
   // Procesar resultados
   ...
   // Liberar el resultado de la consulta:
   mysql_free_result(res);
}
```

Una vez obtenido el conjunto de resultados, disponemos de varias funciones que podemos aplicar sobre él.

Procesar un conjunto de resultados

Hay un conjunto de funciones que se aplican a un conjunto de resultados para obtener información sobre él.

Número de elementos de un conjunto de resultados

Una de las primeras cosas que puede interesarnos es saber cuantos elementos contiene el conjunto de resultados. Para obtener ese valor se puede usar la función mysql_num_rows.

Número de columnas por fila de un conjunto de resultados

Otro parámetro interesante es el número de columnas por fila, para obtenerlo se puede usar la función mysql_num_fields.

Ambas funciones requieren como parámetro un puntero a una estructura MYSQL_RES, y ambas retornan un entero.

```
   // Obtener el número de registros seleccionados:
   i = (int) mysql_num_rows(res);
   j = (int) mysql_num_fields(res);
   // Mostrar el número de registros seleccionados:
   cout << "Consulta:  SELECT * FROM gente" << endl;
   cout << "Numero de filas encontradas:  " << i << endl;
   cout << "Numero de columnas por fila:  " << j << endl;
```

Información sobre columnas

También podemos obtener información sobre cada una de las columnas del conjunto de resultados. Para ello disponemos de las funciones mysql_fetch_field, mysql_fetch_fields y mysql_fetch_field_direct.

La primera obtiene información sobre una de las columnas, en una estructura MYSQL_FIELD. La primera vez que se use devuelve la información sobre la primera columna, la segunda vez sobre la siguiente, etc.

```
   for(l = 0; l < j; l++) {
      columna = mysql_fetch_field(res);
      cout << "Nombre: " << columna->name << endl;
      cout << "Longitud: " << columna->length << endl;
      cout << "Valor por defecto: " << (columna->def ? columna->def :
"NULL") << endl;
   }
```

La función mysql_fetch_fields devuelve la información sobre todas las columnas a la vez, en un array de estructuras.

MYSQL_FIELD.

```
   columna = mysql_fetch_fields(res);
   for(l = 0; l < j; l++) {
      cout << "Nombre: " << columna[l].name << endl;
      cout << "Longitud: " << columna[l].length << endl;
```

```
      cout << "Valor por defecto: " << (columna[l].def ? columna[l].
def : "NULL") << endl;
   }
```

La tercera función, mysql_fetch_field_direct, devuelve la información sobre la columna indicada por el número especificado como segundo parámetro:

```
   // Información sobre columnas n:
   cout << endl << "Informacion sobre columna 1:" << endl;
   columna = mysql_fetch_field_direct(res, 1);
   cout << "Nombre: " << columna->name << endl;
   cout << "Longitud: " << columna->length << endl;
   cout << "Valor por defecto: " << (columna->def ? columna->def :
"NULL") << endl;
   cout << endl;
```

Contenido de las filas de un conjunto de resultados

Por supuesto, también nos interesa obtener cada una de las filas contenidas en el conjunto de resultados. Para esa tarea se usa se la función mysql_fetch_row. Esta función también requiere como parámetro una estructura MYSQL_RES, y da como salida una estructura MYSQL_ROW.

```
   for(l = 0; l < i; l++) {
      row = mysql_fetch_row(res);
      cout << "Registro no. " << l+1 << endl;
      // Mostrar cada campo:
      for(k = 0 ; k < j ; k++)
         cout << ((row[k]==NULL) ? "NULL" : row[k]) << endl;
   }
```

También podemos usar como condición el valor de retorno de la función mysql_fetch_row, ya que tal valor es *NULL* si no quedan filas por recuperar.

El tipo MYSQL_ROW no es más que un array de cadenas, (un char**), que contiene los valores de todas las columnas en la forma de un array de cadenas. Si alguno de los valores de un atributo es *NULL*, el puntero correspondiente a esa columna será *NULL*.

Obtener longitudes de columnas

Mediante la función mysql_fetch_lengths podemos obtener las longitudes de todas las columnas de la fila actual de un conjunto de resultados.

Estos valores se pueden usar para copiar esos valores sin necesidad de calcular sus longitudes, o cuando algunas de las columnas contengan valores binarios, en cuyo caso no podremos usar *strlen* para calcular esas longitudes.

```
// Leer registro a registro y mostrar:
l=1;
for(l = 0; l < i; l++) {
   row = mysql_fetch_row(res);
   lon = mysql_fetch_lengths(res);
   cout << "Registro no. " << l+1 << endl;
   // Mostrar cada campo y su longitud:
   for(k = 0 ; k < j ; k++) {
      cout << ((row[k]==NULL) ? "NULL" : row[k]);
      cout << " longitud: " << lon[k] << endl;
   }
}
```

Acceder a filas del conjunto de forma aleatoria

Mediante mysql_fetch_row accedemos a las filas del conjunto de resultados de forma secuencial. Sin embargo, todas las filas del conjunto de resultados están en memoria, de modo que podemos acceder a ellas en cualquier orden, o acceder sólo a algunas de ellas.

Para acceder a una fila arbitraria se usa la función mysql_data_seek. Esta función precisa dos parámetros. El primero es el conjunto de resultados y el segundo un desplazamiento entre 0 y mysql_num_rows(result)-1.

```
// 3ª fila:
mysql_data_seek(res, 2);
row = mysql_fetch_row(res);
cout << "Tercera fila" << endl;
for(k = 0 ; k < j ; k++) {
   cout << ((row[k]==NULL) ? "NULL" : row[k]) << endl;
}
```

También podemos usar la función mysql_row_tell para obtener el valor de desplazamiento de la fila actual.

Ese valor se puede usar como argumento en llamadas a la función mysql_row_seek, pero el valor usado por estas dos funciones no es un número de fila, por lo tanto, no se puede usar en la función mysql_data_seek.

```
MYSQL_ROW_OFFSET pos;
pos = mysql_row_tell(res);
// lecturas de filas
mysql_row_seek(res, pos);
row = mysql_fetch_row(res);
cout << "Fila guardada" << endl;
for(k = 0 ; k < j ; k++) {
   cout << ((row[k]==NULL) ? "NULL" : row[k]) << endl;
}
```

Trabajar con conjuntos de resultados muy grandes

Si el conjunto de resultados contiene muchas filas puede ser poco práctico usar la función mysql_store_result, ya que se requerirá mucha memoria para almacenar el conjunto completo, posiblemente más de la disponible.

Cuando eso suceda, o cuando preveamos que esa situación puede suceder, podemos usar la función mysql_use_result en su lugar.

Esta función no carga el conjunto de resultados en memoria, y se usa la función mysql_fetch_row para recuperar las filas una por una.

MySQL recomienda no realizar procesos muy largos con cada fila, ya que la tabla está bloqueada para otros usuarios hasta que se llame a mysql_free_result.

```
// El mismo proceso usando mysql_use_result:
// Hacer una consulta con el comando "SELECT * FROM gente":
if(mysql_query(myData, "SELECT * FROM gente")) {
   // Error al realizar la consulta:
   cout << "ERROR: " <<  mysql_error(myData) << endl;
   mysql_close(myData);
   return 2;
}
if((res = mysql_use_result(myData))) {
   j = (int) mysql_num_fields(res);
   while(row = mysql_fetch_row(res)) {
```

```
         for(k = 0 ; k < j ; k++)
            cout << ((row[k]==NULL) ? "NULL" : row[k]) << endl;
      }

      // Liberar el resultado de la consulta:
      mysql_free_result(res);
   }
```

En este ejemplo hacemos uso del hecho de que el valor de retorno de mysql_fetch_row es *NULL* si no quedan filas por leer. No tenemos otra opción, ya que la función mysql_num_rows no funcionará correctamente hasta que se terminen de procesar todas las filas.

Las funciones mysql_data_seek, mysql_row_seek y mysql_row_tell no se pueden usar cuando se recupera un conjunto de resultados con la función mysql_use_result.

Ejemplo

Veamos un ejemplo completo en el que se aplican estas funciones.

```
/*
  Name: MySQL Ejemplo 1
  Author: Salvador Pozo Coronado, salvador@conclase.net
  Date: 07/05/2005
  Description: Ejemplo para mostrar contenidos de bases de datos
  usando MySQL.
  Nota: incluir la opción "-lmysql" en las opciones del linker
  dentro de las opciones de proyecto.
*/

// Includes...
#include <iostream>
#include <windows.h>
#include <mysql/mysql.h>
#include <mysql/mysqld_error.h>
#include <cstring>
#include <cstdio>

using namespace std;

// Programa principal
int main()
{
   // Variables
```

```
   MYSQL           *myData;
   MYSQL_RES       *res;
   MYSQL_ROW        row;
   MYSQL_FIELD     *columna;
   int              i, j, k, l;
   unsigned long   *lon;
   MYSQL_ROW_OFFSET pos;

   // Intentar iniciar MySQL:
   if(!(myData = mysql_init(0))) {
      // Imposible crear el objeto myData
      cout << "ERROR: imposible crear el objeto myData." << endl;
      rewind(stdin);
      getchar();
      return 1;
   }

   // Debe existir un usuario "curso" con clave de acceso "clave" y
   // con al menos el privilegio "SELECT" sobre la tabla
   // "prueba.gente"
   if(!mysql_real_connect(myData, NULL, "curso", "clave", "prueba",
MYSQL_PORT, NULL, 0)) {
      // No se puede conectar con el servidor en el puerto especificado.
      cout << "Imposible conectar con servidor mysql en el puerto "
           << MYSQL_PORT << " Error: " << mysql_error(myData) << endl;
      mysql_close(myData);
      rewind(stdin);
      getchar();
      return 1;
   }

   // Conectar a base de datos.
   if(mysql_select_db(myData, "prueba")) {
      // Imposible seleccionar la base de datos, posiblemente no existe.
      cout << "ERROR: " << mysql_error(myData) << endl;
      mysql_close(myData);
      rewind(stdin);
      getchar();
      return 2;
   }

   // Hacer una consulta con el comando "SELECT * FROM gente":
   if(mysql_query(myData, "SELECT * FROM gente")) {
      // Error al realizar la consulta:
      cout << "ERROR: " <<  mysql_error(myData) << endl;
      mysql_close(myData);
      rewind(stdin);
      getchar();
```

```
      return 2;
   }

   // Almacenar el resultado de la consulta:
   if((res = mysql_store_result(myData))) {
      // Obtener el número de registros seleccionados:
      i = (int) mysql_num_rows(res);
      // Obtener el número de columnsa por fila:
      j = (int) mysql_num_fields(res);

      // Mostrar el número de registros seleccionados:
      cout << "Consulta:  SELECT * FROM gente" << endl;
      cout << "Numero de filas encontradas:  " << i << endl;
      cout << "Numero de columnas por fila:  " << j << endl;

      // Información sobre columnas usando mysql_fetch_field:
      cout << endl << "Informacion sobre columnas:" << endl;
      for(l = 0; l < j; l++) {
         columna = mysql_fetch_field(res);
         cout << "Nombre: " << columna->name << endl;
         cout << "Longitud: " << columna->length << endl;
         cout << "Valor por defecto: " << (columna->def ? columna->def :
"NULL") << endl;
      }
      cout << endl;

      // Información sobre columnas usando mysql_fetch_fields:
      cout << endl << "Informacion sobre columnas:" << endl;
      columna = mysql_fetch_fields(res);
      for(l = 0; l < j; l++) {
         cout << "Nombre: " << columna[l].name << endl;
         cout << "Longitud: " << columna[l].length << endl;
         cout << "Valor por defecto: " << (columna[l].def ? columna[l].
def : "NULL") << endl;
      }
      cout << endl;

      // Información sobre columnas n, usando mysql_fetch_field_direct:
      cout << endl << "Informacion sobre columna 1:" << endl;
      columna = mysql_fetch_field_direct(res, 1);
      cout << "Nombre: " << columna->name << endl;
      cout << "Longitud: " << columna->length << endl;
      cout << "Valor por defecto: " << (columna->def ? columna->def :
"NULL") << endl;
      cout << endl;

      // Leer registro a registro y mostrar:
      l=1;
```

```
      for(l = 0; l < i; l++) {
         row = mysql_fetch_row(res);
         lon = mysql_fetch_lengths(res);
         cout << "Registro no. " << l+1 << endl;
         // Mostrar cada campo y su longitud:
         for(k = 0 ; k < j ; k++) {
            cout << ((row[k]==NULL) ? "NULL" : row[k]);
            cout << " longitud: " << lon[k] << endl;
         }
      }

      cout << endl;

      // Mostrar sólo 3ª fila:
      mysql_data_seek(res, 2);
      row = mysql_fetch_row(res);
      cout << "Tercera fila" << endl;
      for(k = 0 ; k < j ; k++) {
         cout << ((row[k]==NULL) ? "NULL" : row[k]) << endl;
      }

      // Almacenar posición de la fila actual (la 4ª):
      pos = mysql_row_tell(res);
      // Lectura de filas hasta el final:
      while(mysql_fetch_row(res));
      // Recuperar la posición guardada:
      mysql_row_seek(res, pos);
      row = mysql_fetch_row(res);
      cout << "Cuarta fila" << endl;
      for(k = 0 ; k < j ; k++) {
         cout << ((row[k]==NULL) ? "NULL" : row[k]) << endl;
      }

      cout << endl;

      // Liberar el resultado de la consulta:
      mysql_free_result(res);
   }

   // El mismo proceso usando mysql_use_result:
   // Hacer una consulta con el comando "SELECT * FROM gente":
   if(mysql_query(myData, "SELECT * FROM gente")) {
      // Error al realizar la consulta:
      cout << "ERROR: " <<  mysql_error(myData) << endl;
      mysql_close(myData);
      rewind(stdin);
      getchar();
      return 2;
```

```
   }

   // Mostrar todas las filas:
   if((res = mysql_use_result(myData))) {
      j = (int) mysql_num_fields(res);
      while(row = mysql_fetch_row(res)) {
         for(k = 0 ; k < j ; k++)
            cout << ((row[k]==NULL) ? "NULL" : row[k]) << endl;
      }

      // Liberar el resultado de la consulta:
      mysql_free_result(res);
   }
   // Cerrar la conexión
   mysql_close(myData);

   // Esperar a que se pulse una tecla y salir.
   rewind(stdin);
   getchar();
   return 0;
}
```

3 Crear una base de datos

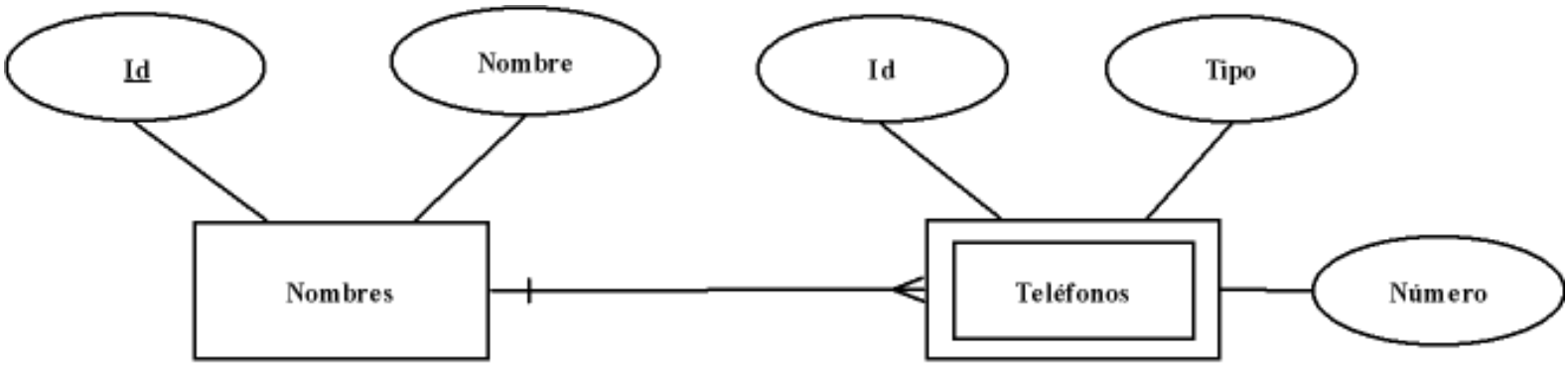
Hemos empezado por lo más simple: leer datos desde una base de datos existente. En este capítulo veremos cómo crear y eliminar bases de datos y tablas.

En este capítulo y el siguiente crearemos una aplicación de ejemplo para gestionar una base de datos simple de contactos, con sólo nombres y números de teléfono.

La base de datos se llamará "agenda", y para poder empezar a trabajar con ella necesitaremos un usuario con acceso a esa base de datos y con todos los privilegios. De modo que empezaremos por abrir una sesión **MySQL** y conceder al usuario "curso" todos los privilegios sobre la tabla "agenda". Recordemos que esto se puede hacer aunque tal base de datos no exista:

```
mysql> GRANT ALL ON agenda.* TO curso;
Query OK, 0 rows affected (0.08 sec)
```

La base de datos "agenda" contendrá dos tablas, una con los nombres de los contactos, y una segunda con los números de teléfono. Ambas tablas estarán interrelacionadas por una clave de indentificación de usuario:



La estructura de la tabla de nombres es:

- id: clave identificadora. Usaremos un entero autoincrementado.
- nombre: nombre de contacto. Usaremos una cadena de longitud variable de 40 caracteres.

En cuanto a la tabla de teléfonos, la estructura es:

- id: clave identificadora de contacto. Clave foránea de la tabla de nombres. Crearemos una referencia de modo que las modificaciones de clave se transmitan en cascada y los borrados también.
- tipo: tipo de teléfono. Indicará si es un celular, fijo, particular, etc. Usaremos una cadena de longitud variable de 20 caracteres.
- numero: número de teléfono. Usaremos una cadena de longitud variable de 15 caracteres.

Por supuesto, usaremos tablas **InnoDB**.

Averiguar si existe una base de datos

Antes de empezar a trabajar con una base de datos puede ser interesante saber si tal base de datos existe o no. Esto nos permitirá crear la base de datos si es la primera vez que se ejecuta la aplicación o si se ha eliminado la base de datos desde la última ejecución.

Para averiguar si una base de datos existe intentaremos activarla, y si se produce un error, verificaremos si ese error es debido a que la base de datos no existe. Para ello comprobaremos si el valor de error es ER_BAD_DB_ERROR. Vamos a crear una función C++ para esta tarea:

```
bool ExisteDB(MYSQL *myData, char *db)
{
   // Conectar a base de datos.
   if(mysql_select_db(myData, db)) {
      if(ER_BAD_DB_ERROR == mysql_errno(myData)) return false;
   }
   return true;
}
```

Otra alternativa, quizás mucho mejor, es verificar si existe la base de datos usando una sentencia SHOW DATABASES. Si mostramos sólo las bases de datos cuyo nombre coincida con el de la que deseamos verificar, bastará con comprobar el número de filas retornadas. Si ese número es cero significa que la base de datos no existe:

```
bool ExisteDB(MYSQL *myData, char *db)
{
   char *consulta;
   char *plantilla = "SHOW DATABASES LIKE \'%s\'";
   MYSQL_RES *res;
   bool valorret = true;

   consulta = new char[strlen(db)+strlen(plantilla)];
   sprintf(consulta, plantilla, db);

   if(!mysql_query(myData, consulta)) {
      if((res = mysql_store_result(myData))) {
         // Procesar resultados
         if(!mysql_num_rows(res)) valorret = false;
         // Liberar el resultado de la consulta:
         mysql_free_result(res);
      }
   }
   delete[] consulta;
   return valorret;
}
```

Hemos tenido que hacer uso de memoria dinámica para preparar la cadena para hacer la consulta. Esto será muy habitual, sobre todo cuando las consultas se compliquen, y requieran selecciones y proyecciones (seleccionar

columnas y filas). Más adelante crearemos una función para hacer este proceso más sencillo.

Crear una base de datos

A pesar de que existe una función en el API C de **MySQL** para crear bases de datos, mysql_create_db, su uso está desaconsejado, y es preferible usar mysql_query para lanzar una consulta CREATE DATABASE.

```
   if(!ExisteDB(myData, "agenda")) {
      cout << "La base de datos \"agenda\" no existe." << endl;
      cout << "La creamos..." << endl;
      mysql_query(myData, "CREATE DATABASE agenda");
   }
```

Esto creará la base de datos "agenda" si no existe previamente, pero no creará ninguna tabla.

Comprobar si una tabla existe

Para comprobar si existe una tabla podemos hacer algo parecido a lo que hemos hecho con la base de datos. Intentaremos hacer una consulta sobre la tabla, y si fracasamos, verificaremos si el error es ER_NO_SUCH_TABLE. En tal caso, la tabla no existe.

Crearemos otra función C++ para esta tarea:

```
bool ExisteTabla(MYSQL *myData, char *db, char *tabla)
{
   char *consulta;
   char *plantilla = "SELECT * FROM %s.%s";
   bool valorret = true;

   consulta = new char[strlen(db)+strlen(tabla)+strlen(plantilla)-1];
   sprintf(consulta, plantilla, db, tabla);
   if(mysql_query(myData, consulta)) {
      if(ER_NO_SUCH_TABLE == mysql_errno(myData)) valorret = false;
   }
   delete[] consulta;
   return valorret;
}
```

Otra opción, mucho más recomendable, es hacer una consulta usando la sentencia SHOW TABLES, de un modo análogo al usado con las bases de datos:

```
bool ExisteTabla(MYSQL *myData, char *db, char *tabla)
{
   char *consulta;
   char *plantilla = "SHOW TABLES FROM %s LIKE \'%s\'";
```

```
   MYSQL_RES *res;
   bool valorret = true;

   consulta = new char[strlen(db)+strlen(tabla)+strlen(plantilla)-1];
   sprintf(consulta, plantilla, db, tabla);

   if(!mysql_query(myData, consulta)) {
      if((res = mysql_store_result(myData))) {
         // Procesar resultados
         if(!mysql_num_rows(res)) valorret = false;
         // Liberar el resultado de la consulta:
         mysql_free_result(res);
      }
   }
   delete[] consulta;
   return valorret;
}
```

Crear una tabla

Para crear una tabla usaremos la sentencia CREATE TABLE. El proceso se explica con detalle en el del curso de **MySQL**.

Para crear las tablas que necesitamos para esta base de datos usaremos las siguientes consultas SQL:

```
mysql> CREATE TABLE agenda.nombres (
    -> id INT NOT NULL AUTO_INCREMENT,
    -> nombre VARCHAR(40),
    -> PRIMARY KEY (id))
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (0.61 sec)

mysql> CREATE TABLE agenda.telefonos (
    -> id INT NOT NULL,
    -> tipo VARCHAR(20),
    -> numero VARCHAR(15),
    -> FOREIGN KEY (id) REFERENCES nombres(id)
    -> ON DELETE CASCADE
    -> ON UPDATE CASCADE)
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (0.14 sec)

mysql>
```

En nuestro programa C++ crearemos las tablas que no existan usando este código:

```
   if(!ExisteTabla(myData, "agenda", "nombres")) {
      cout << "La tabla \"Nombres\" no existe." << endl;
```

```
      cout << "La creamos..." << endl;
      mysql_query(myData, "CREATE TABLE agenda.nombres ("
         "id INT NOT NULL AUTO_INCREMENT, "
         "nombre VARCHAR(40), "
         "PRIMARY KEY (id)) "
         "ENGINE=InnoDB");
   }

   if(!ExisteTabla(myData, "agenda", "telefonos")) {
      cout << "La tabla \"Telefonos\" no existe." << endl;
      cout << "La creamos..." << endl;
      mysql_query(myData, "CREATE TABLE agenda.telefonos ("
         "id INT NOT NULL, "
         "tipo VARCHAR(20), "
         "numero VARCHAR(15), "
         "FOREIGN KEY (id) REFERENCES nombres(id) "
         "ON DELETE CASCADE "
         "ON UPDATE CASCADE) "
         "ENGINE=InnoDB");
   }
```

Eliminar una tabla

Análogamente podemos eliminar tablas usando consultas con la sentencia DROP TABLE:

```
   cout << "Eliminar tabla de nombres" << endl;
   mysql_query(myData, "DROP TABLE agenda.nombres");
```

Eliminar una base de datos

Lo mismo sirve para eliminar bases de datos, usando la sentencia DROP DATABASE:

```
   cout << "Eliminar base de datos" << endl;
   mysql_query(myData, "DROP DATABASE agenda");
```

4 Añadir, modificar y eliminar datos

En este capítulo veremos cómo manipular datos contenidos en una base de datos, añadir, modificar o eliminar datos de tablas.

Insertar datos en una tabla

Evidentemente, la forma más simple de introducir datos en una tabla es usar una sentencia INSERT:

```
mysql_query(myData, "INSERT INTO nombres (nombre) VALUES "
   "('Carlos'), "
   "('Felipe'), "
   "('Irene')");
```

Pero esto no es muy práctico, ya que generalmente tendremos que introducir datos suministrados por el usuario de la aplicación, y eso significa que debemos leer valores de atributos y crear consultas con valores variables.

En una consulta INSERT necesitaremos, generalmente, dos listas de valores. La primera es una lista de atributos, y la segunda una lista de valores para esos atributos. En el ejemplo usamos una lista de listas de valores de atributos, pero podemos simplificar el proceso haciendo una consulta para cada inserción en la tabla.

En nuestra base de datos "agenda", la tabla de nombres tiene dos atributos, pero el primero de ellos es un campo *AUTO_INCREMENT*, por lo que no será necesario especificar un valor cuando se inserten datos, ese valor se calcula de forma automática. Por lo tanto, la lista de atributos contiene un único valor: 'nombre'.

Veamos un ejemplo:

```
// Insertar datos:
cout << "introducir datos" << endl;
char nombre[41];
char *consulta;
char resp;
char patronnombre[] = "INSERT INTO nombres (nombre) VALUES(\'%s\')";
do {
   // Leer contacto:
```

```
      cout << "Nombre de contacto: " << flush;
      cin.getline(nombre, 41);
      consulta = new char[strlen(patronnombre)+strlen(nombre)-1];
      sprintf(consulta, patronnombre, nombre);
      mysql_query(myData, consulta);
      delete[] consulta;
      cout << "Otro contacto?: " << flush;
      cin >> resp;
      cin.ignore();
   } while (toupper(resp) == 'S');
```

Obtener el último valor autoincrementado insertado

Frecuentemente, como sucede en nuestra base de datos de ejemplo, tenemos que insertar datos en una tabla interrelacionada con otra. Para hacerlo necesitamos el valor del atributo referenciado. En nuestro ejemplo necesitamos conocer el último valor de 'id' generado para insertar los registros de teléfonos usando el mismo valor.

Para conseguir ese valor hay dos posibilidades. Una consiste en hacer una consulta para obtener el valor de la función LAST_INSERT_ID(). La otra, que nos resultará mucho más práctica, consiste en usar la función del API mysql_insert_id, que hace lo mismo, pero nos evita procesar un conjunto de resultados:

```
   // Insertar datos:
   cout << "introducir datos" << endl;
   char nombre[41];
   char tipo[21];
   char numero[15];
   char *consulta;
   char resp;
   my_ulonglong id;
   char patronnombres[] = "INSERT INTO nombres (nombre) VALUES(\'%s\')";
   char patrontelefonos[] = "INSERT INTO telefonos (id, tipo, numero) "
      "VALUES(\'%11d\', \'%s\', \'%s\')";
   do {
      // Leer contacto:
      cout << "Nombre de contacto: " << flush;
      cin.getline(nombre, 41);
      consulta = new char[strlen(patronnombres)+strlen(nombre)-1];
      sprintf(consulta, patronnombres, nombre);
      mysql_query(myData, consulta);
      delete[] consulta;
      id = mysql_insert_id(myData);
```

```
      // Leer teléfonos:
      do {
         cout << "Tipo de telefono: " << flush;
         cin.getline(tipo, 21);
         cout << "Numero de telefono: " << flush;
         cin.getline(numero, 15);
         consulta = new char[strlen(patrontelefonos)+11+strlen(tipo)+
            strlen(numero)-7];
         sprintf(consulta, patrontelefonos, static_cast(id),
            tipo, numero);
         mysql_query(myData, consulta);
         delete[] consulta;
         cout << "Otro telefono?: " << flush;
         cin >> resp;
         cin.ignore();
      } while (toupper(resp) == 'S');
      cout << "Otro contacto?: " << flush;
      cin >> resp;
      cin.ignore();
   } while (toupper(resp) == 'S');
```

Eliminar datos de una tabla

Eliminar filas de una tabla es igualmente simple. Nos limitaremos a hacer una consulta DELETE con una cláusula *WHERE* de modo que sólo eliminemos las filas que queramos.

En nuestro ejemplo "agenda", las filas de teléfonos están relacionadas mediante una clave foránea a las filas de nombres, de modo que si se elimina una fila de nombres, automáticamente se eliminan todas las filas de teléfonos con el mismo valor de 'id'. Esto nos facilita la tarea de eliminar contactos completamente:

```
   // Eliminar un contacto:
   char patrondelete[] = "DELETE FROM nombres WHERE nombre = \"%s\"";
   cout << "Nombre a eliminar de agenda: " << flush;
   cin.getline(nombre, 41);
   consulta = new char[strlen(patrondelete)+strlen(nombre)-1];
   sprintf(consulta, patrondelete, nombre);
   mysql_query(myData, consulta);
   delete[] consulta;
```

Puede ser interesante conocer el número de filas afectadas por la sentencia DELETE. Se puede obtener una información análoga a la presentada en el cliente **MySQL** usando la función del API

mysql_affected_rows:

```
mysql> DELETE FROM nombres WHERE nombre="Pepito";
Query OK, 1 row affected (0.01 sec)
```

Por ejemplo:

```
   cout << "Filas eliminadas: " << (int)mysql_affected_rows(myData) <<
endl;
```

Actualizar filas de una tabla

El proceso para modificar filas se explica con detalle en el del curso de **MySQL**. Para hacerlo desde un programa C/C++ bastará con hacer las consultas UPDATE o REPLACE adecuadas.

Tabla de contenido.

Indice de Funciones y tipos API C SQL (72)

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

En mayúsculas aparecen los tipos definidos en **MySQL**, en minúsculas, las funciones del API C.

-S-

-T-

-U-

-W-

Tipo MYSQL

```
typedef struct st_mysql {
  NET           net;                    /* Communication parameters */
  gptr          connector_fd;           /* ConnectorFd for SSL */
  char          *host,*user,*passwd,*unix_socket,*server_version,
*host_info,*info;
  char          *db;
  struct charset_info_st *charset;
  MYSQL_FIELD   *fields;
  MEM_ROOT      field_alloc;
  my_ulonglong affected_rows;
  my_ulonglong insert_id;               /* id if insert on table with
NEXTNR */
  my_ulonglong extra_info;              /* Used by mysqlshow */
  unsigned long thread_id;              /* Id for connection in server */
  unsigned long packet_length;
  unsigned int  port;
  unsigned long client_flag,server_capabilities;
  unsigned int  protocol_version;
  unsigned int  field_count;
  unsigned int  server_status;
  unsigned int  server_language;
  unsigned int  warning_count;
  struct st_mysql_options options;
  enum mysql_status status;
  my_bool       free_me;                /* If free in mysql_close */
  my_bool       reconnect;              /* set to 1 if automatic reconnect
*/

  /* session-wide random string */
  char          scramble[SCRAMBLE_LENGTH+1];

 /*
   Set if this is the original connection, not a master or a slave we have
   added though mysql_rpl_probe() or mysql_set_master()/ mysql_add_slave()
 */
  my_bool rpl_pivot;
  /*
    Pointers to the master, and the next slave connections, points to
    itself if lone connection.
  */
  struct st_mysql* master, *next_slave;

  struct st_mysql* last_used_slave; /* needed for round-robin slave pick */
```

```
 /* needed for send/read/store/use result to work correctly with
replication */
  struct st_mysql* last_used_con;

  LIST  *stmts;                     /* list of all statements */
  const struct st_mysql_methods *methods;
  void *thd;
  /*
    Points to boolean flag in MYSQL_RES  or MYSQL_STMT. We set this flag
    from mysql_stmt_close if close had to cancel result set of this object.
  */
  my_bool *unbuffered_fetch_owner;
} MYSQL;
```

Función mysql_affected_rows()

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Devuelve el número de filas afectadas por la última sentencia UPDATE, las borradas por la última sentencia DELETE o insertadas por la última sentencia INSERT. Debe ser llamada inmediatamente después de la llamada a mysql_query() para las sentencias UPDATE, DELETE o INSERT. Para sentencias SELECT, **mysql_affected_rows()** funciona igual que mysql_num_rows().

Valores de retorno

Un entero mayor que cero indica el número de filas afectadas o recuperadas. Cero indica que ninguna fila fue actualizada para una sentencia UPDATE, que no hay filas coincidentes con la cláusula *WHERE* de la consulta o que no se ha ejecutado ninguna consulta. -1 indica que la consulta ha retornado un error o que, para una consulta SELECT, **mysql_affected_rows()** fue llamada antes de una llamada a la función mysql_store_result(). Como **mysql_affected_rows()** devuelve un valor sin signo, se puede verificar el valor -1 comparando el valor de retorno con (my_ulonglong)-1 (o con (my_ulonglong)~0, que es equivalente).

Errores

No tiene.

Ejemplo

```
mysql_query(&mysql,"UPDATE products SET cost=cost*1.25 WHERE group=10");
printf("%ld products updated",(long) mysql_affected_rows(&mysql));
```

Si se especifica la opción **CLIENT_FOUND_ROWS** cuando se conecta a **mysqld**, **mysql_affected_rows()** devolverá el número de filas coincidentes con la cláusula *WHERE* para la sentencia UPDATE.

Hay que tener en cuenta que si se usa un comando REPLACE, **mysql_affected_rows()** devuelve 2 si la nueva fila reemplaza a una antigua. Esto es porque en ese caso la fila es insertada después de que la duplicada sea borrada.

Si se usa INSERT ... ON DUPLICATE KEY UPDATE para insertar una fila, **mysql_affected_rows()** devuelve 1 si la fila es insertada como una nueva y 2 si se acutaliza una fila existente.

Función mysql_autocommit()

```
my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)
```

Activa el modo *autocommit* si *mode* es 1, o lo desactiva si *mode* es 0.

Esta función se añadió en MySQL 4.1.0.

Valores de retorno

Cero si tiene éxito. Distinto de cero si se produce un error.

Errores

Ninguno.

Función mysql_change_user()

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char
*password, const char *db)
```

Cambia el usuario y hace que la base de datos especificada por *db* sea la base de datos por defecto (la actual) en la conexión especificada por *mysql*. En consultas sucesivas, esta base de datos será la base de datos por defecto para referencias a tablas que no incluyan un especificador de base de datos explícito.

Esta función se añadió en MySQL 3.23.3.

mysql_change_user() falla si el usuario conectado no puede ser autentificado o si no tiene permiso para usar las base de datos. En ese caso, el usuario y la base de datos no se cambian.

El parámetro *db* debe ser *NULL* si no se quiere tener una base de datos por defecto.

A partir de MySQL 4.0.6 este comando producirá siempre un ROLLBACK en cualquier transacción activa, cierra todas las tablas temporales, desbloquea todas las tablas bloqueadas y resetea el estado como si se hubiese hecho una nueva conexión. Esto ocurrirá aunque el usuario no se cambie.

Valores de retorno

Cero si tiene éxisto. Distinto de cero si ocurre algún error.

Errores

Los mismos que se obtienen de la función mysql_real_connect().

CR_COMMANDS_OUT_OF_SYNC: Los comandos fueron ejecutados en un orden incorrecto.

CR_SERVER_GONE_ERROR: El servidor MySQL no está presente.

CR_SERVER_LOST: La conexión con el servidor se ha perdido durante la consulta.

CR_UNKNOWN_ERROR: Se ha producido un error desconocido.

ER_UNKNOWN_COM_ERROR: El servidor MySQL no implementa este comando (probablemente es

un servidor antiguo).

ER_ACCESS_DENIED_ERROR: El usuario o la constraseña son incorrectos.

ER_BAD_DB_ERROR: La base de datos no existe.

ER_DBACCESS_DENIED_ERROR: El usuario no tiene derechos de acceso a la base de datos.

ER_WRONG_DB_NAME: El nombre de la base de datos es demasiado largo.

Ejemplo

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
   fprintf(stderr, "Imposible cambiar de usuario.  Error: %s\n",
           mysql_error(&mysql));
}
```

Función mysql_character_set_name()

```
const char *mysql_character_set_name(MYSQL *mysql)
```

Devuelve el conjunto de caracteres por defecto para la conexión actual.

Valores de retorno

El conjunto de caracteres por defecto.

Errores

No tiene.

Función mysql_close()

```
void mysql_close(MYSQL *mysql);
```

Cierra una conexión previamente abierta. **mysql_close** también libera de memoria el manipulador apuntado por *mysql* si ese manipulador fue creado automáticamente por mysql_init o mysql_connect.

Parámetros

- **mysql**: dirección de una estructura MYSQL existente.

Valor de retorno

No tiene

Errores

No se pueden producir.

Función mysql_commit()

```
my_bool mysql_commit(MYSQL *mysql)
```

Acomete la transacción actual.

Esta función se añadió en MySQL 4.1.0.

Valores de retorno

Cero si tiene éxito, un valor distinto de cero si se produce un error.

Errores

Ninguno.

Función mysql_connect()

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *usuario,
const char *password);
```

Esta función está fuera de uso, es preferible usar mysql_real_connect.

mysql_connect intenta establecer una conexión con un motor de bases de datos MySQL ejecutándose en *host*. **mysql_connect** debe completarse con éxito antes de que se pueda ejecutar cualquier otra función del API, con excepción de la función mysql_get_client_info.

El significado de los parámetros es el mismo que para los correspondientes en la función mysql_real_connect con la diferencia de que el parámetro *mysql* puede ser NULL. En ese caso el API reserva memoria de forma automática para la estructura connection y la libera cuando se llama a mysql_close. La desventaja de esta solución es que no es posible obtener un mensaje de error si la conexión falla. (Para obtener información de errores con las funciones mysql_errno o mysql_error, se debe proporcionar un puntero MYSQL válido.)

Valor de retorno

El mismo que para mysql_real_connect.

Errores

Los mismos que para mysql_real_connect.

Función mysql_create_db()

```
int mysql_create_db(MYSQL *mysql, const char *database);
```

Crea la base de datos con el nombre del parámetro *database*.

Esta función está desaconsejada. Es preferible usar mysql_query para ejecutar una instrucción SQL CREATE DATABASE en su lugar.

Parámetros

- **mysql**: El primer parámetro debe ser la dirección de una estructura MYSQL existente.
- **database**: nombre de la base de datos a crear.

Valor de retorno

El valor de retorno es cero si la base de datos fue creada correctamente. Un valor distinto de cero indica que ha ocurrido un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: el servidor MySQL no está presente.

CR_SERVER_LOST: la conexión con el servidor se perdió durante la consulta.

CR_UNKNOWN_ERROR: ha ocurrido un error desconocido.

Ejemplo

```
if(mysql_create_db(&mysql, "my_database"))
{
   fprintf(stderr, "Imposible crear la nueva base de datos.  Error: %s\n",
           mysql_error(&mysql));
}
```

Tipo MYSQL_DATA

```
typedef struct st_mysql_data {
  my_ulonglong rows;
  unsigned int fields;
  MYSQL_ROWS *data;
  MEM_ROOT alloc;
#if !defined(CHECK_EMBEDDED_DIFFERENCES) || defined(EMBEDDED_LIBRARY)
  MYSQL_ROWS **prev_ptr;
#endif
} MYSQL_DATA;
```

Función mysql_data_seek()

```
void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)
```

Busca una fila arbitraria en un conjunto de resultados de una consulta. El valor de *offset* o desplazamiento es un número de fila y debe estar en el rango de 0 a mysql_num_rows(result)-1.

Esta función requiere que la estructura del conjunto de resultados contenga el resultado entero de la consulta, de modo que **mysql_data_seek()** puede ser usado sólo junto con mysql_store_result(), no con mysql_use_result().

Valores de retorno

Ninguno.

Errores

Ninguno

Función mysql_debug()

```
void mysql_debug(const char *debug)
```

Realiza un *DBUG_PUSH* con la cadena dada. **mysql_debug()** usa la librería de depuración **Fred Fish debug**. Para usar esta función, se debe compilar la librería del cliente para que soporte depuración.

Valores de retorno

Ninguno.

Errores

Ninguno.

Ejemplo

La llamada mostrada aquí hace que la librería de cliente genere una traza en el fichero '/tmp/client.trace' de la máquina actual:

```
mysql_debug("d:t:O,/tmp/client.trace");
```

Función mysql_drop_db()

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

Elimina la base de datos con el nombre *db*.

Esta función está desaconsejada. Es preferible usar mysql_query() para lanzar una sentencia SQL DROP DATABASE en su lugar.

Valores de retorno

Cero si la base de datos fue eliminada. Distinto de cero si se produjo un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: Los comandos fueron ejecutados en un orden incorrecto.

CR_SERVER_GONE_ERROR: El servidor MySQL no está presente.

CR_SERVER_LOST: La conexión con el servidor se ha perdido durante la consulta.

CR_UNKNOWN_ERROR: Se ha producido un error desconocido.

Ejemplo

```
if(mysql_drop_db(&mysql, "my_database"))
  fprintf(stderr, "Error al eliminar la base de datos: Error: %s\n",
          mysql_error(&mysql));
```

Función mysql_dump_debug_info()

```
int mysql_dump_debug_info(MYSQL *mysql)
```

Indica al servidor que escriba alguna información de depuración en el diario. Para que funcione, el usuario conectado debe tener el privilegio *SUPER*.

Valores de retorno

Cero si el comando se ha ejecutado correctamente. Distinto de cero si se produjo un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: Los comandos fueron ejecutados en un orden incorrecto.

CR_SERVER_GONE_ERROR: El servidor MySQL no está presente.

CR_SERVER_LOST: La conexión con el servidor se ha perdido durante la consulta.

CR_UNKNOWN_ERROR: Se ha producido un error desconocido.

Función mysql_eof()

```
my_bool mysql_eof(MYSQL_RES *result)
```

Esta función está desaconsejada. mysql_errno() o mysql_error() pueden usarse en su lugar.

mysql_eof() determina si la última fila de un conjunto de resultados ha sido leída.

Si se obtiene un conjunto de resultados a partir de una llamada exitosa a mysql_store_result(), el cliente recibe el conjunto completo en una operación. En ese caso, un valor *NULL* de una llamada a mysql_fetch_row() siempre significa que el final del conjunto de resultados ha sido alcanzado y es innecesario llamar a **mysql_eof()**. Cuando se usa con mysql_store_result(), **mysql_eof()** siempre devuelve verdadero.

Por otra parte, si se usa mysql_use_result() para iniciar una recuperación de un conjunto de resultados, las filas del conjunto se obtienen del servidor una a una como si se hubiese llamado a mysql_fetch_row() repetidamente. Debido a que se puede producir un error en la conexión durante este proceso, un valor de retorno *NULL* desde mysql_fetch_row() no significa necesariamente que en final del conjunto de resultados ha sido alcanzado con normalidad. En ese caso, se puede usar **mysql_eof()** para determinar qué ha sucedido. **mysql_eof()** devuelve un valor distinto de cero si se ha alcanzado el final del conjunto de resultados y cero si ha ocurrido un error.

Historicamente, **mysql_eof()** precede a las funciones de error estándar de MySQL mysql_errno() y mysql_error(). Ya que estas funciones de error proporcionan la misma información, su uso es preferible sobre **mysql_eof()**, que actualmente está desaconsejada. (De hecho, esas funciones proporcionan más información, ya que **mysql_eof()** devuelve sólo un valor booleano mientras que las funciones de error indican un motivo para el error cuando uno ocurre.)

Valores de retorno

Cero si no ha ocurrido un error. Distinto de cero si se ha alcanzado el final del conjunto de resultados.

Errores

Ninguno

Ejemplo

El siguiente ejemplo muestra cómo se debe usar **mysql_eof()**:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // hacer algo con los datos
}
if(!mysql_eof(result))  // mysql_fetch_row() fallo debido a un error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

Sin embargo, se puede conseguir el mismo efecto con las funciones de error estándar de MySQL:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // hacer algo con los datos
}
if(mysql_errno(&mysql))  // mysql_fetch_row() fallo debido a un error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

Función mysql_errno()

```
unsigned int mysql_errno(MYSQL *mysql);
```

Para la conexión especificada por *mysql*, devuelve el código de error para la función del API invocada más recientemente, tanto si tuvo éxito como si no. Un valor de retorno cero significa que no ocurrió un error.

Los números de mensajes de error del cliente se listan en el fichero de cabecera 'errmsg.h'. Los del servidor en `mysqld_error.h'. En el fichero de distribución de MySQL se puede encontrar una lista completa de mensajes de error y números de error en el fichero 'Docs/mysqld_error.txt'.

Hay algunas funciones, como mysql_fetch_row que no activan el número de error si tienen éxito.

Una regla para esto es que todas las funciones que tienen que preguntar al servidor por información resetean el número de error si tienen éxito.

Parámetros

- **mysql**: El primer parámetro debe ser la dirección de una estructura MYSQL existente.

Valor de retorno

Un valor de código de error para la última llamada a una función mysql_xxx, si ha fallado. Cero significa que no ha ocurrido un error.

Función mysql_error()

```
const char *mysql_error(MYSQL *mysql)
```

Para la conexión especificada por *mysql*, **mysql_error()** devueleve una cadena terminada en cero consistente en el mensaje de error para la invocación más reciente de una función del APIo que haya fallado. Si ninguna función ha fallado, el valor de retorno de **mysql_error()** puede ser el error previo o una cadena vacía para indicar que no hay error.

Una regla para esto es que todas las funciones que tienen que preguntar al servidor por información resetean el número de error si tienen éxito.

Para funciones que anulen el valor de mysql_errno(), las dos comprobaciones siguientes son equivalentes:

```
if(mysql_errno(&mysql))
{
    // se ha producido un error
}

if(mysql_error(&mysql)[0] != '\0')
{
    // se ha producido un error
}
```

El lenguaje de los mensajes de error del cliente puede cambiarse recompilando la librería del cliente MySQL. Actualmente no es posible elegir mensajes de error en varios lenguajes distintos.

Valores de retorno

Una cadena terminada en cero que describe el error. Una cadena vacía si no se ha producido un error.

Errores

Ninguno.

Función mysql_escape_string()

Se debe usar la función mysql_real_escape_string() en su lugar.

Esta función es idéntica a mysql_real_escape_string() excepto que mysql_real_escape_string() toma un manipulador de conexión como primer argumento y escapa la cadena de acuerdo con el conjunto de caracteres actual. **mysql_escape_string()** no requiere un argumento de conexión y no tiene en cuenta la asignación actual del conjunto de caracteres.

Función mysql_fetch_field()

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

Devuelve la definición de una columna de un conjunto de resultados como una estructura MYSQL_FIELD. Hay que llamar a esta función repetidamente para recuperar la información sobre todas las columnas del conjunto de resultados. **mysql_fetch_field()** devuelve *NULL* cuando no quedan más campos.

mysql_fetch_field() se resetea para devolver la información sobre la primera columna cada vez que se ejecute una nueva consulta SELECT. El campo que se devolverá por **mysql_fetch_field()** también se ve afectado por llamadas a mysql_field_seek().

Si se ha llamado a mysql_query() para realizar un SELECT en una tabla pero no se ha llamado a mysql_store_result(), MySQL devuelve el tamaño por defecto de bloque (8KB) si se llama a mysql_fetch_field() para preguntar por la longitud de una columna *BLOB*. (El tamaño de 8KB se elige porque MySQL no conoce el tamaño máximo para el *BLOB*. Esto se hará configurable en el futuro.) Una vez que se ha recuperado el conjunto de resultados, *field->max_length* contiene la longitud del valor más largo para esa columna en la consulta específica.

Valores de retorno

La estructura MYSQL_FIELD para la columna actual. *NULL* si no quedan columnas por recuperar.

Errores

Ninguno.

Ejemplo

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("Nombre de campo %s\n", field->name);
}
```

Función mysql_fetch_fields()

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

Devuelve un array con todas las estructuras MYSQL_FIELD para un conjunto de resultados. Cada estructura proporciona una definición de campo para una columna del conjunto de resultados.

Valores de retorno

Un array de estructuras MYSQL_FIELD para todas las columnas del conjunto de resultados.

Errores

Ninguno.

Ejemplo

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
   printf("Columna %u es %s\n", i, fields[i].name);
}
```

Función mysql_fetch_field_direct()

```
MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int
fieldnr)
```

Dado un número de campo *fieldnr* para una columna dentro de un conjunto de resultados, devuelve la definición de esa columna como una estructura MYSQL_FIELD. Se debe usar esta función para decuperar la definición de una columna arbitaria. El valor de *fieldnr* debe estar en el rango de 0 a mysql_num_fields(result)-1.

Valores de retorno

La estructura MYSQL_FIELD para la columna especificada.

Errores

Ninguno.

Ejemplo

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Columna %u es %s\n", i, field->name);
}
```

Función mysql_fetch_lengths()

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

Devuelve las longitudes de las columnas de la fila actual de un conjunto de resultados. Si se planea copiar valores de campos, esta información de longitud es muy útil para la optimización, ya que se puede evitar la llamada a strlen(). Además, si el conjunto de resultados contiene datos binarios, se debe usar esta función para determinar el tamaño de los datos, porque strlen() devolverá resultados incorrectos para cualquier campo que contenga caracteres nulos.

La longitud para cadenas vacías y para columnas que contengan valores *NULL* es cero. Para distinguir entre ambos casos, ver la descripción de la función mysql_fetch_row().

Valores de retorno

Un array de enteros *long* sin signo que representas los tamaños de cada columna (sin incluir el carácter nulo terminador). *NULL* si se produce un error.

Errores

mysql_fetch_lengths() sólo es válido para la fila actual del conjunto de resultados. Devuelve *NULL* si es llamada antes de llamar a mysql_fetch_row() o después de recuperar todas las filas del resultado.

Ejemplo

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
         printf("Columna %u tiene %lu bytes de longitud.\n", i, lengths
```

```
 [i]);
     }
 }
```

Función mysql_fetch_row()

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result);
```

Recupera la siguiente fila de un conjunto de resultados. Cuando se usa después de mysql_store_result, **mysql_fetch_row** devuelve *NULL* si no quedan más filas por recuperar. Cuando se usa después de mysql_use_result, mysql_fetch_row devuelve *NULL* si no quedan filas por recuperar o si se ha producido un error.

El número de valores en la fila viene dado por mysql_num_fields(result). Si la fila contiene el valor devuelto por una llamada a **mysql_fetch_row**, los punteros a los valores serán accesibles como *row[0]* a *row[mysql_num_fields(result)-1]*. Los valores *NULL* en la fila se indican mediante punteros nulos.

Las longitudes de los valores de los campos en la fila se pueden obtener mediante una llamada a la función mysql_fetch_lengths. Los campos vacíos y los que contengan *NULL* tendrán longitud 0; se pueden distinguir comprobando el puntero para el valor del campo. Si el puntero es *NULL*, el campo es *NULL*; en otro caso, el campo estará vacío.

Valores de retorno

Una estructura MYSQL_ROW para la siguiente fila. *NULL* si no hay más filas para recuperar o si se ha producido un error.

Errores

Los errores no se resetean entre sucesivas llamadas a **mysql_fetch_row**.

CR_SERVER_LOST: la conexión con el servidor se perdió durante la consulta.

CR_UNKNOWN_ERROR: ha ocurrido un error desconocido.

Ejemplo

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;
```

```
num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result))) {
   unsigned long *lengths;
   lengths = mysql_fetch_lengths(result);
   for(i = 0; i < num_fields; i++) {
      printf("[%.*s] ", (int) lengths[i], row[i] ? row[i] : "NULL");
   }
   printf("\n");
}
```

Tipo MYSQL_FIELD

```
typedef struct st_mysql_field {
  char *name;                 /* Name of column */
  char *org_name;             /* Original column name, if an alias */
  char *table;                /* Table of column if column was a field */
  char *org_table;            /* Org table name, if table was an alias */
  char *db;                   /* Database for table */
  char *catalog;              /* Catalog for table */
  char *def;                  /* Default value (set by mysql_list_fields)
*/
  unsigned long length;       /* Width of column (create length) */
  unsigned long max_length;   /* Max width for selected set */
  unsigned int name_length;
  unsigned int org_name_length;
  unsigned int table_length;
  unsigned int org_table_length;
  unsigned int db_length;
  unsigned int catalog_length;
  unsigned int def_length;
  unsigned int flags;         /* Div flags */
  unsigned int decimals;      /* Number of decimals in field */
  unsigned int charsetnr;     /* Character set */
  enum enum_field_types type; /* Type of field. See mysql_com.h for types
*/
} MYSQL_FIELD;
```

Función mysql_field_count()

```
unsigned int mysql_field_count(MYSQL *mysql)
```

Si se está usando una versión de MySQL previa a la 3.22.24, se debe usar en su lugar unsigned int mysql_num_fields(MYSQL *mysql).

Devuelve el número de columnas para la consulta más reciente de la conexión *mysql*.

El uso habitual de esta función es cuando mysql_store_result() devuelve *NULL* (y entonces no se dispone de un puntero al conjunto de resultados). En ese caso, se puede llamar a **mysql_field_count()** para determinar si mysql_store_result() debería haber producido un resultado no vacío. Esto permite al programa cliente tomar las acciones adecuadas sin saber si la consulta fue una sentencia SELECT (o parecida). El ejemplo mostrado aquí ilustra como puede hacerse esto.

Ver mysql_store_result() para ver por qué a veces esta función devuelve *NULL* después de que mysql_query() retorne con éxito.

Valores de retorno

Un entero sin signo que representa el número de columnas en el conjunto de resultados.

Errores

Ninguno

Ejemplo

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // consulta exitosa, procesar cualquier dato retornado
```

```
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // recuperar filas, y después llamar a mysql_free_result(result)
    }
    else  // mysql_store_result() no ha devuelto nada; ¿debería haberlo
hecho?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // la consulta no devuelve datos
            // (no fue un SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
        else // mysql_store_result() ha devuelto datos
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
    }
}
```

Una alternativa consiste en reemplazar la llamada a mysql_field_count(&mysql) con mysql_errno (&mysql). En ese caso, se estará comprobando de forma directa un error de mysql_store_result() en lugar de deducir del valor de retorno de mysql_field_count() si la sentencia fue un SELECT.

Función mysql_field_seek()

```
MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET
offset)
```

Asigna al cursor de campo el desplazamiento *offset*. La siguiente llamada a mysql_fetch_field() recuperará la definición de campo de la columna asociada con ese *offset*.

Para colocar el cursor al principio de la fila, hay que indicar un desplazamiento de cero.

Valores de retorno

El valor previo del cursor de campo.

Errores

Ninguno.

Función mysql_field_tell()

```
MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)
```

Devuelve la posición del cursor de campo usado por la última llamada a mysql_fetch_field(). Este valor puede ser usado como argumento para la función mysql_field_seek().

Valores de retorno

El valor actuale del desplazamiento del cursor de campo.

Errores

Ninguno.

Función mysql_free_result()

```
void mysql_free_result(MYSQL_RES *result);
```

Libera la memoria reservada para un conjunto de resultados *result* por una función mysql_store_result, mysql_use_result, mysql_list_dbs, etc. Cuando se haya terminado de trabajar con un conjunto de resultados, se debe liberar la memoria que usa mediante una llamada a **mysql_free_result**.

No se debe intentar acceder a un conjunto de resultados después de haberlo liberado.

Valores de retorno

No tiene

Errores

No se pueden producir.

Función mysql_get_client_info()

```
char *mysql_get_client_info(void)
```

Devuelve una cadena que representa la versión de la librería de cliente.

Valores de retorno

Una cadena de caracteres que representa la versión de la librería de cliente MySQL.

Errores

Ninguno.

Función mysql_get_client_version()

```
unsigned long mysql_get_client_version(void)
```

Devuelve un entero que representa la versión de la librería de cliente. El valor tiene el formato XYYZZ, donde X es la versión, YY es el nivel de *release* y ZZ es el número de versión dentro del nivel de *release*. Por ejemplo, un valor 40102 representa una versión de librería de cliente 4.1.2.

Esta función fue añadida en MySQL 4.0.16.

Valores de retorno

Un entero que representa la versión de librería de cliente MySQL.

Errores

Ninguno.

Función mysql_get_host_info()

```
char *mysql_get_host_info(MYSQL *mysql)
```

Devuelve una cadena que describe el tipo de conexión actual, incluyendo el nombre del servidor.

Valores de retorno

Una cadena de caracteres que representa el nombre del servidor y el tipo de conexión.

Errores

Ninguno.

Función mysql_get_proto_info()

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

Devuelve la versión de protocolo para la conexión actual.

Valores de retorno

Un valor entero sin signo que represente la versión de protocolo usada por la conexión actual.

Errores

Ninguno

Función mysql_get_server_info()

```
char *mysql_get_server_info(MYSQL *mysql)
```

Devuelve una cadena que representa el número de versión del servidor.

Valores de retorno

Una cadena de caracteres que representa el número de versión del servidor.

Errores

Ninguno.

Función mysql_get_server_version()

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Devuelve el número de versión del servidor como un entero.

Esta función se añadió en MySQL 4.1.0.

Valores de retorno

Un número que representa la versión del servidor MySQL en este formato:

```
major_version*10000 + minor_version *100 + sub_version
```

Por ejemplo, 4.1.2 se devuelve como 40102.

Esta función es práctica en programas cliente para determinar rápidamente si existe alguna capacidad específica de determinada versión.

Errores

Ninguno.

Función mysql_hex_string()

```
unsigned long mysql_hex_string(char *to, const char *from, unsigned long
length)
```

Esta función se usa para crear una cadena SQL legal que se puede usar en una sentencia SQL.

La cadena *from* se codifica a formato hexadecimal, con cada carácter codificado como dos dígitos hexadecimales. El resultado se coloca en la cadena *to* y se añade el carácter nulo terminador.

La cadena apuntada por *by* debe tener *length* bytes de longitud. Se debe conseguir memoria para el buffer *to* para que tenga al menos *length**2+1 bytes de longitud. Cuando **mysql_hex_string()** regresa, el contenido de *to* será una cadena terminada con nulo. El valor de retorno será la longitud de la cadena codificada, sin incluir el carácter nulo terminador.

El valor de retorno puede ser colocado en una sentencia SQL usando tanto el formato 0xvalor o X'valor'. Sin embargo, el valor devuelto no incluye el 0x o X'...'. El usuario debe añadir cualquiera de ambos, tal como desee.

mysql_hex_string() fue añadido en MySQL 4.0.23 y 4.1.8.

Ejemplo

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
end = strmov(end,"0x");
end += mysql_hex_string(end,"Qué es esto",11);
end = strmov(end,",0x");
end += mysql_hex_string(end,"datos binarios: \0\r\n",19);
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Error al insertar fila, Error: %s\n",
           mysql_error(&mysql));
}
```

La función **strmov()** usada en este ejemplo se incluye en la librería mysqlclient y funciona como **strcpy**

() pero devolviendo un puntero al terminador nulo del primer parámetro.

Valores de retorno

La longitud del valor colocado en *to*, sin incluir el carácter nulo terminador.

Errores

Ninguno.

mysql_info()

```
char *mysql_info(MYSQL *mysql)
```

Recupera una cadena que proporciona información sobre la consulta ejecutada más recientemente, pero sólo para las sentencias listadas aquí. Para otras sentencias, **mysql_info()** devuelve *NULL*. El formato de la cadena varía dependiendo del tipo de consulta, tal como se describe. Los números son sólo ilustrativos; la cadena contendrá valores apropiados para cada consulta.

INSERT INTO ... SELECT ...
Formato de cadena: `Records: 100 Duplicates: 0 Warnings: 0`

INSERT INTO ... VALUES (...),(...),(...)...
Formato de cadena: `Records: 3 Duplicates: 0 Warnings: 0`

LOAD DATA INFILE ...
Formato de cadena: `Records: 1 Deleted: 0 Skipped: 0 Warnings: 0`

ALTER TABLE
Formato de cadena: `Records: 3 Duplicates: 0 Warnings: 0`

UPDATE
Formato de cadena: `Rows matched: 40 Changed: 40 Warnings: 0`

Hay que tener en cuenta que **mysql_info()** devuelve un valor no nulo para INSERT ... VALUES sólo para el formato de la sentencia de múltiples filas (es decir, sólo si se especifica una lista de múltiples valores).

Valores de retorno

Una cadena de caracteres que representa información adicional sobre la consulta ejecutada más recientemente. *NULL* si no hay información disponible para la consulta.

Errores

Ninguno.

Función mysql_init()

```
MYSQL *mysql_init(MYSQL *mysql);
```

Crea o inicializa un objeto MYSQL, que posteriormente puede ser usado por la función mysql_real_connect. Si el parámetro es NULL, la función crea, inicializa y devuelve un objeto nuevo. En otro caso, el objeto es inicializado y se devuelve su dirección. Si la función crea un objeto nuevo, será liberado cuando se invoque a la función mysql_close para cerrar la conexión.

Para evitar pérdidas de memoria, usar el procedimiento siguiente, que debe ser hecho cada vez que la aplicación se enlace con la librería *libmysqlclient* o *libmysqld*:

- Llamar a mysql_library_init() antes de la primera llamada a **mysql_init()**.
- Llamar a mysql_library_end() después de que la aplicación haya cerrado cualquier conexión abierta que haya sido hecha usando el API C de MySQL.
- Si se desea, la llamada a mysql_library_init() puede omitirse, porque **mysql_init()** la invocará automáticamente si es necesario.

Valor de retorno

Un manipulador inicializado. NULL si no existe memoria suficiente para crear un objeto nuevo.

Errores

En caso de memoria insuficiente, se devuelve NULL.

Función mysql_insert_id()

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

Devuelve el valor generado por una columna *AUTO_INCREMENT* para la sentencia INSERT o UPDATE previa. Usar esta función después de realizar una sentecia INSERT en una tabla que contenga una columna *AUTO_INCREMENT*.

Más precisamente, **mysql_insert_id()** se actualiza en las siguientes condiciones:

- Sentencias INSERT que almacenen un valor en una columna *AUTO_INCREMENT*. Esto será cierto tanto si el valor fue generado automáticamente almacenando los valores especiales NULL ó 0 en la columna, o si se ha usado un valor explícito no especial.
- En el caso de una sentencia INSERT de múltiples filas, **mysql_insert_id()** devuelve el primer valor *AUTO_INCREMENT* generado automáticamente; si no se a generado ninguno, se devuelve el último valor insertado explícitamente en la columna *AUTO_INCREMENT*.
- Sentencias INSERT que generen un valor *AUTO_INCREMENT* mediante la inserción del valor LAST_INSERT_ID(expr) en cualquier columna.
- Sentencias INSERT que generen un valor *AUTO_INCREMENT* mediante la actualización de cualquier columna con el valor LAST_INSERT_ID(expr).
- El valor de **mysql_insert_id()** no se ve afectado por sentencias como SELECT, que devuelvan un conjunto de resultados.
- Si la última sentencia a devuelto un error, el valor de **mysql_insert_id()** está indefinido.

mysql_insert_id() devuelve 0 si la sentencia previa no usa un valor *AUTO_INCREMENT*. Si se necesita guardar el valor para usarlo más tarde, asegurarse de llamar a **mysql_insert_id()** inmediatamente después de la sentencia que genera el valor.

El valor de **mysql_insert_id()** se ve afectado sólo por sentencias lanzadas dentro de la conexión de cliente actual. No se ve afectado por sentencias lanzadas por otros clientes.

También hay que tener en cuenta que la función SQL LAST_INSERT_ID() siempre contiene el valor *AUTO_INCREMENT* generado más recientemente, y no se resetea entre sentencias porque el valor de esa función es mantenido por el servidor. Otra diferencia es que LAST_INSERT_ID() no se actualiza si de asigna a una columna *AUTO_INCREMENT* un valor específico no especial.

El motivo de estas diferencias entre LAST_INSERT_ID() y **mysql_insert_id()** es que LAST_INSERT_ID() se ha diseñado para que sea fácil de usar en scripts mientras que **mysql_insert_id**

() intenta proporcionar una información algo más exacta de lo que ocurre en una columna *AUTO_INCREMENT*.

Valores de retorno

Los descritos anteriormente.

Errores

Ninguno.

Función mysql_kill()

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Pregunta al servidor para matar un hilo especifiacdo por *pid*.

Valores de retorno

Cero si tiene éxito. Distinto de cero si ocurre un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: el servidor MySQL no está presente.

CR_SERVER_LOST: la conexión con el servidor se perdió durante la última consulta.

CR_UNKNOWN_ERROR: se ha producido un error desconocido.

Función mysql_library_end()

```
void mysql_library_end(void)
```

Es un sinónimo de la función mysql_server_end(). Se añadió en MySQL 4.1.10 y 5.0.3.

Función mysql_library_init()

```
int mysql_library_init(int argc, char **argv, char **groups)
```

Es un sinónimo de la función mysql_server_init(). Se añadió en MySQL 4.1.10 y 5.0.3.

Función mysql_list_dbs()

```
MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)
```

Devuelve un conjunto de resultados consistente en los nombres de las bases de datos en el servidor que coinciden con la expresión regular simple especificada por el parámetro *wild*. *wild* puede contener los caracteres comodín `%' o `_', o puede ser un puntero *NULL* para buscar todas las bases de datos. Llamar a **mysql_list_dbs()** es similar que ejecutar la consulta SHOW DATABASES [LIKE wild].

Se debe liberar el conjunto de resultados mediante mysql_free_result().

Valores de retorno

Un conjunto de resultados MYSQL_RES si tiene éxito. *NULL* si ocurre un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: los comandos fueron ejecutados en un orden inapropiado.

CR_OUT_OF_MEMORY: falta memoria.

CR_SERVER_GONE_ERROR: el servidor MySQL no está presente.

CR_SERVER_LOST: la conexión con el servidor se perdió durante la última consulta.

CR_UNKNOWN_ERROR: se ha producido un error desconocido.

Función mysql_list_fields()

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char
*wild)
```

Devuelve un conjunto de resultados que consiste en los nombres de los campos en la tabla *table*, que coincidan con la expresión regular simple especificada por el parámetro *wild*. *wild* puede contener los caracteres comodín '%' o '_', o puede ser un puntero *NULL* para obtener todos los campos. Llamar a **mysql_list_fields()** es similar a ejecutar la consulta SHOW COLUMNS FROM tbl_name [LIKE wild].

Es mejor usar SHOW COLUMNS FROM tbl_name en lugar de **mysql_list_fields()**.

Se debe liberar el conjunto de resultados usando la función mysql_free_result().

Valores de retorno

Un conjunto de resultados MYSQL_RES si tiene éxito. *NULL* si se produce un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: el servidor MySQL no está presente.

CR_SERVER_LOST: la conexión con el servidor se perdió durante la última consulta.

CR_UNKNOWN_ERROR: se ha producido un error desconocido.

Función mysql_list_processes()

```
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
```

Devuelve un conjunto de resultados que describen los procesos actuales del servidor. Es el mismo tipo de información que el devuelto por *mysqladmin processlist* o una consulta SHOW PROCESSLIST.

Se debe liberar el conjunto de resultados usando la función mysql_free_result().

Valores de retorno

Un conjunto de resultados MYSQL_RES si tiene éxito. *NULL* si se produce un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: el servidor MySQL no está presente.

CR_SERVER_LOST: la conexión con el servidor se perdió durante la última consulta.

CR_UNKNOWN_ERROR: se ha producido un error desconocido.

Función mysql_list_tables()

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

Devuelve un conjunto de resultados que consiste en los nombres de las tablas en la base de datos actual que coinciden con la expresión regular simple especificada por el parámetro *wild*. *wild* puede contener los caracteres comodín '%' o '_', o puede ser un puntero *NULL* para recuperar todas las tablas. Llamar a **mysql_list_tables()** es similar a ejecutar la consulta SHOW TABLES [LIKE wild].

Se debe liberar el conjunto de resultados usando la función mysql_free_result().

Valores de retorno

Un conjunto de resultados MYSQL_RES si tiene éxito. *NULL* si se produce un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: el servidor MySQL no está presente.

CR_SERVER_LOST: la conexión con el servidor se perdió durante la última consulta.

CR_UNKNOWN_ERROR: se ha producido un error desconocido.

Función mysql_more_results()

```
my_bool mysql_more_results(MYSQL *mysql)
```

Devuelve verdadero si existen más resultados en la consulta actualmente en ejecución, y la aplicación debe llamar a mysql_next_result() para recuperar los resultados.

Esta función fue añadida en MySQL 4.1.0.

Valores de retorno

TRUE (1) si existen más resultados. *FALSE* (0) si no existen.

En la mayoría de los casos, se puede llamar a mysql_next_result() en lugar comprobar si existen más resultados e iniciar una recuperación si es así.

Errores

Ninguno.

Función mysql_next_result()

```
int mysql_next_result(MYSQL *mysql)
```

Si existen más resultados de consultas, **mysql_next_result()** lee los siguientes resultados de consulta y devuelve el estado a la aplicación.

Se debe llamar a mysql_free_result() para la consulta anterior si devolvión un conjunto de resultados.

Después de llamar a **mysql_next_result()** el estado de la conexión es el mismo que si se hubiese llamado a mysql_real_query() o a mysql_query() para la siguiente consulta. Esto significa que se puede llamar a mysql_store_result(), mysql_warning_count(), mysql_affected_rows(), etc.

Si **mysql_next_result()** devuelve un error, no se ejecutará ninguna otra sentencia y no hay más resultados a recuperar.

Esta función se añadió en MySQL 4.1.0.

Valores de retorno

Valor de retorno	Descripción
0	Éxito y hay más resultados
-1	Éxito y no hay más resultados
>0	Se ha producido un error

Errores

CR_COMMANDS_OUT_OF_SYNC: Los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: El servidor MySQL no está presente.

CR_SERVER_LOST: La conexión al servidor se perdió durante la consulta.

CR_UNKNOWN_ERROR: Se ha producido un error desconocido.

Función mysql_num_fields()

```
unsigned int mysql_num_fields(MYSQL_RES *result);
```

O

```
unsigned int mysql_num_fields(MYSQL *mysql);
```

El segundo formato no funciona en la versión 3.22.24 de MySQL y siguientes. Para pasar un argumento **MYSQL***, se debe usar en su lugar la función mysql_field_count.

Devuelve el número de columnas en un conjunto de resultados.

También se puede obtener el número de columnas desde un puntero al conjunto de resultados o a un manipulador de conexión. Se puede usar el manipulador de conexión si mysql_store_result o mysql_use_result devuelve NULL (y por ese motivo no se dispone de un puntero al conjunto de resultados). En ese caso, se puede llamar a mysql_field_count para determinar si mysql_store_result ha producido un resultado no vacío. Esto permite al programa cliente tomar la acción apropiada sin saber si la consulta ha sido una sentencia SELECT (o del tipo *SELECT*). El ejemplo incluido cómo se puede realizar esto.

Valores de retorno

Un entero sin signo que representa el número de campos en un conjunto de resultados.

Errores

Ninguno.

Ejemplo

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;
```

```
if (mysql_query(&mysql,query_string)) {
   // error
}
else { // consulta exitosa, procesar cualquier dato devuelto por ella
    result = mysql_store_result(&mysql);
    if (result) {  // no hay filas
       num_fields = mysql_num_fields(result);
       // recuperar filas, y después llamar a mysql_free_result(result)
    }
    else { // mysql_store_result() no devolvió nada; ¿debería?
       if (mysql_errno(&mysql)) {
          fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
       }
       else if (mysql_field_count(&mysql) == 0) {
          // la consulta no ha devuelto datos
          // (no era un SELECT)
          num_rows = mysql_affected_rows(&mysql);
       }
    }
}
```

Una alternativa (si se sabe que la consulta debe devolver un conjunto de resultados) es remplazar la llamada *mysql_errno(&mysql)* por una comprobación de si *mysql_field_count(&mysql)* es igual a 0. Esto sólo ocurrirá si algo ha salido mal.

Función mysql_num_rows()

```
my_ulonglong mysql_num_rows(MYSQL_RES *result);
```

Devuelve el número de filas en el conjunto de resultados.

El uso de **mysql_num_rows** depende si se ha usado mysql_store_result o mysql_use_result para obtener el conjunto de resultados. Si se ha usado mysql_store_result, **mysql_num_rows** puede ser llamada inmediatamente. Si se ha usado mysql_use_result, **mysql_num_rows** no devuelve el valor correcto hasta que todas las filas del conjunto de resultados hayan sido recuperadas.

Valor de retorno

El número de filas en el conjunto de resultados.

Errores

Ninguno.

Función mysql_options()

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const char *arg)
```

Puede usarse para activar opciones de conexión extra y afectar al comportamiento de una conexión. Esta función puede ser llamada muchas veces para cambiar varias opciones.

mysql_options() debe ser llamada después de mysql_init() y antes de mysql_connect() o mysql_real_connect().

El argumento *option* es la opción que se quiere activar; el argumento *arg* es el valor de la opción. Si la opción es un entero, entonces *arg* debe apuntar al valor del entero.

Posible valores de opciones:

Opción	Tipo de argumento	Función
MYSQL_INIT_COMMAND	char *	Comando a ejecutar cuando se conecte al servidor MySQL. Será reejecutado automáticamente cuando se reconecte.
MYSQL_OPT_COMPRESS	Not used	Usar el protocolo comprimido cliente/ servidor.
MYSQL_OPT_CONNECT_TIMEOUT	unsigned int *	Tiempo límite para la conexión en segundos.
MYSQL_OPT_LOCAL_INFILE	puntero opcional a uint	Si no se proporciona un puntero o si el puntero apunta a un unsigned int != 0 el comando LOAD LOCAL INFILE estará permitido.
MYSQL_OPT_NAMED_PIPE	No usado	Usar tuberías con nombre para conectar a un servidor MySQL en NT.
MYSQL_OPT_PROTOCOL	unsigned int *	Tipo de protocolo a usar. Debe ser uno de los valores enumerados mysql_protocol_type definido en 'mysql.h'. Nuevo en 4.1.0.

MYSQL_OPT_READ_TIMEOUT	unsigned int *	Tiempo límite para lecturas desde el servidor (actualmente sólo funciona en Windows con conexiones TCP/IP). Nuevo en 4.1.1.
MYSQL_OPT_WRITE_TIMEOUT	unsigned int *	Tiempo límite para escrituras en el servidor (actualmente sólo funciona en Windows con conexiones TCP/IP). Nuevo en 4.1.1.
MYSQL_READ_DEFAULT_FILE	char *	Lee las opciones desde el fichero nombrado en lugar de hacerlo desde 'my.cnf'.
MYSQL_READ_DEFAULT_GROUP	char *	Lee las opciones desde el grupo nombrado de 'my.cnf' o desde el fichero especificado con MYSQL_READ_DEFAULT_FILE.
MYSQL_REPORT_DATA_TRUNCATION	my_bool *	Activa o desactiva informes de errores de truncado de datos para sentencias preparadas vía MYSQL_BIND.error. (Por defecto: desactivada) Nuevo en 5.0.3.
MYSQL_SECURE_AUTH	my_bool*	Si para conectar al servidor no está soportado el nuevo tipo de contraseñas 4.1.1. Nuevo en 4.1.1.
MYSQL_SET_CHARSET_DIR	char*	El nombre de camino del directorio que contiene los ficheros de definición de juegos de caracteres.
MYSQL_SET_CHARSET_NAME	char*	El nombre del juego de caracteres a usar como juego de caracteres por defecto.
MYSQL_SHARED_MEMORY_BASE_NAME	char*	Nombre del objeto de memoria compartida para comunicarse con el servidor. Debe ser la misma que para la opción -shared-memory-base-name usada para el servidor mysqld al que se quiere conectar. Nuevo en 4.1.0.

Hay que tener en cuenta que el grupo del cliente siempre es leído si se usa MYSQL_READ_DEFAULT_FILE o MYSQL_READ_DEFAULT_GROUP.

El grupo especificado en el fichero de opciones debe contener las opciones siguientes:

Opción	Descripción
connect-timeout	Tiempo límite de conexión en segundos. En **Linux** este tiempo también se usa para esperar la primera respuesta del servidor.
compress	Usar el protocolo cliente/servidor comprimido.
database	Conectar a esta base de datos si no se especifica una en el comando de conexión.
debug	Opciones de depuración.
disable-local-infile	Deshabilitar el uso de LOAD DATA LOCAL.
host	Nombre de ordenador por defecto.
init-command	Comando a ejecutar cuando se conecte al servidor MySQL. Será reejecutado automáticamente cuando se reconecte.
interactive-timeout	Lo mismo que especificar CLIENT_INTERACTIVE en mysql_real_connect().
local-infile[=(0|1)]	Si no hay argumento o si argumento != 0 se habilita el uso de LOAD DATA LOCAL.
max_allowed_packet	Tamaño máximo de paquete que el cliente puede leer del servidor.
multi-results	Permite múltiples conjuntos de resultados desde ejecuciones de sentencias múltiples o procedimientos de almacenamiento. Nuevo en 4.1.1.
multi-statements	Permite al cliente enviar múltiples sentencias en una única cadena (separadas por ';'). Nuevo en 4.1.9.
password	Contraseña por defecto.
pipe	Usar tuberías con nombre para conectar al servidor MySQL en NT.
protocol={TCP | SOCKET | PIPE | MEMORY}	Protocolo a usar cuando se conecte al servidor (Nuevo en 4.1)
port	Número de puerto por defecto.
return-found-rows	Hace que mysql_info() devuelva las filas encontradas en lugar de las actualizadas cuando se usa UPDATE.

shared-memory-base-name=name	Nombre de memoria compartida a usar para conectar al servidor (por defecto es "MYSQL"). Nuevo en MySQL 4.1.
socket	Fichero *socket* por defecto.
user	Usuario por defecto.

Nótese que *timeout* ha sido remplazado por *connect-timeout*, pero *timeout* seguirá funcionado durante todavía.

Valores de retorno

Cero si tiene éxito. Distinto de cero si se usa una opción desconocida.

Ejemplo

```
MYSQL mysql;

(&mysql);
 mysql_options(&mysql,MYSQL_OPT_COMPRESS,0);
 mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
 if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,
NULL,0))
 {
     fprintf(stderr, "Error al conectar a la base de datos: Error: %s\n",
           mysql_error(&mysql));
 }
```

Este código pide al cliente que use el protocolo cliente/servidor comprimido y lee las opciones adicionales desde la sección *odbc* del fichero 'my.cnf'.

Función mysql_ping()

```
int mysql_ping(MYSQL *mysql)
```

Comprueba si la conexión con el servidor está funcionando. Si la conexión ha caído, se intenta una reconexión automática.

Esta función puede ser usada por clientes que permanecen inactivas por mucho tiempo, para comprobar si el servidor ha cerrado la conexión y reconectarla si es necesario.

Valores de retorno

Cero si el servidor responde. Distinto de cero si ha ocurrido un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: Los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: El servidor MySQL no está presente.

CR_UNKNOWN_ERROR: Se ha producido un error desconocido.

Función mysql_query()

```
int mysql_query(MYSQL *mysql, const char *query);
```

Ejecuta una consulta SQL apuntada por la cadena terminada con cero del parámetro *query*. La consulta debe consistir en una sentencia SQL simple. No se debe añadir el punto y coma al final (`;') o \g a la cadena. Si la ejecución de múltiples sentencias está permitida, la cadena puede contener varias sentencias separadas por punto y coma.

mysql_query no puede se usado para consultas que contengan datos binarios; en esos casos se debe usar mysql_real_query. (Los datos binarios pueden contener el carácter `\0', que **mysql_query** interpreta como el final de la cadena de consulta.)

Si se quiere averiguar si la consulta a devuelto un conjunto de resultados o no, se puede usar mysql_field_count para verificarlo.

Valor de retorno

El valor de retorno es cero si la consulta se ha completado correctamente. Un valor distinto de cero indica que ha ocurrido un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: el servidor MySQL no está presente.

CR_SERVER_LOST: la conexión con el servidor se perdió durante la consulta.

CR_UNKNOWN_ERROR: ha ocurrido un error desconocido.

Función mysql_real_connect()

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char
*usuario, const char *password,
   const char *database, unsigned int puerto, const char *unix_socket,
unsigned long client_flag);
```

mysql_real_connect intenta establecer una conexión con un motor de bases de datos MySQL ejecutándose en *host*. **mysql_real_connect** debe completarse con éxito antes de que se puedan ejecutar otras funciones del API, con la excepción de mysql_get_client_info.

Los parámetros a especificar son los siguientes:

- **mysql**: El primer parámetro debe ser la dirección de una estructura MYSQL existente. Antes de llamar a **mysql_real_connect** se debe llamar a mysql_init para inicializar la estructura MYSQL. Se pueden modificar muchas opciones de conexión mediante la función mysql_options.
- **host**: El valor de *host* puede ser tanto un nombre como una dirección IP. Si *host* es NULL o la cadena "localhost", se asume que se trata de una conexión al host local. Si el sistema operativo soporta sockets (Unix) o tuberías con nombre (Windows), se en lugar del protocolo TCP/IP para conectar con el servidor.
- **usuario**: contiene el identificador de login del usuario MySQL. Si el user es NULL o una cadena vacía "", se asume el usuario actual. Bajo Unix, es el nombre de login actual. Bajo Windows ODBC, el nombre de usuario debe especificarse explícitamente.
- **password** contiene el password del usuario. Si es NULL, sólo se verificarán usuarios de la tabla que tengan como password una cadena vacía. Esto permite al administrador de la base de datos configurar el sistema de privilegios de MySQL de modo que existan usuarios con diferentes privilegios dependiendo del password especificado. Nota: no se debe intentar encriptar el password antes de llamar a **mysql_real_connect**; la encriptación del password se hace automáticamente por el API del cliente.
- **database** es el nombre de la base de datos. Si *database* no es NULL, la conexión hará que esa sea la base de datos por defecto.
- **puerto** si no es 0, el valor se usará como número de puerto en la conexión TCP/IP. Hay que tener en cuenta que el parámetro *host* determina el tipo de conexión.
- **unix_socket** si no es NULL, la cadena especifica el socket o tubería con nombre que se usará. Hay que tener en cuenta que el parámetro *host* determina el tipo de conexión.
- **client_flag** es normalmente 0, pero puede usarse una combinación de los siguientes flags en circunstancias muy especiales:

Flag	Descripción

CLIENT_COMPRESS	Usar un protocolo de compresión.
CLIENT_FOUND_ROWS	Devuelve el númer de líneas encontradas (coincidentes) en lugar del número de líneas afectadas.
CLIENT_IGNORE_SPACE	Permite espacios después de los nombres de función. Hace de todos los nombres de función palabras reservadas.
CLIENT_INTERACTIVE	Permite segundos "interactive_timeout" (en lugar de segundos "wait_timeout") de inactividad antes de cerrar la conexión.
CLIENT_LOCAL_FILES	Permite manipulación LOAD DATA LOCAL.
CLIENT_MULTI_STATEMENTS	Informa al servidor de que el cliente puede enviar consultas multilínea (separadas con `;'). Si este flag no se activa, las consultas multilínea serán desactivadas. Nuevo en versión 4.1.
CLIENT_MULTI_RESULTS	Informa al servidor de que el cliente puede manipular conjuntos de resultados multiples procedentes de multi-consultas o procedimientos almacenados. Esto es agrupado automáticamente si CLIENT_MULTI_STATEMENTS es activado. Nuevo en 4.1.
CLIENT_NO_SCHEMA	No permite la sintaxis *db_name.tbl_name.col_name*. Esto es para **ODBC**. Hace que el analizador sintáctico genere un error si se usa tal sintaxis, lo que resulta útil para encontrar errores en algunos programas **ODBC**.
CLIENT_ODBC	El cliente es un cliente ODBC. Esto hace que mysqld se adapte más a ODBC.
CLIENT_SSL	Usar SSL (protocolo de encriptado). Esta opción no debe ser activada por aplicaciones; se activa internamente por la librería del cliente.

Valores de retorno

Si la conexión ha tenido éxito, el valor de retorno es un manipulador de conexión MYSQL*, NULL si la conexión no ha tenido éxito. Para una conexión exitosa, el valor de retorno es el mismo que el valor del primer parámetro.

Errores

CR_CONN_HOST_ERROR: fallo al conectar con el servidor MySQL.

CR_CONNECTION_ERROR: fallo al conectar al servidor local MySQL.

CR_IPSOCK_ERROR: fallo al crear un socket IP.

CR_OUT_OF_MEMORY: falta memoria.

CR_SOCKET_CREATE_ERROR: fallo al crear un socket Unix.

CR_UNKNOWN_HOST: fallo al encontrar una dirección IP para el nombre de host.

CR_VERSION_ERROR: al intentar conectar con el servidor se ha producido un error de protocolo por el uso de una librería de cliente que usa un protocolo de una versión diferente. Esto puede ocurrir si se usa una librería de cliente muy antigua para conectarse con un servidor que no se ha arrancado con la opción "--old-protocol".

CR_NAMEDPIPEOPEN_ERROR: fallo al crear una tubería con nombre en Windows.

CR_NAMEDPIPEWAIT_ERROR: fallo al esperar uan tubería con nombre en Windows.

CR_NAMEDPIPESETSTATE_ERROR: fallo al tomar un manipulador de tubería en Windows.

CR_SERVER_LOST: si el connect_timeout > 0 y se tarda más de connect_timeout segundos en conectar con el servidor o si el servidor ha muerto mientras se ejecuta el comando de inicialización.

Ejemplo

```
 MYSQL mysql;

(&mysql);
(&mysql,MYSQL_READ_DEFAULT_GROUP,"your_prog_name");
 if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,
NULL,0))
 {
     fprintf(stderr, "Fallo al conectar con la base de datos: Error: %s\n",
           mysql_error(&mysql));
 }
```

Mediante el uso de mysql_options() la librería MySQL leerá las secciones [client] y [your_prog_name] del fichero 'my.cnf', lo que asegura que el programa funcionará, aunque alquien haya arrancado MySQL de algún modo no estándar.

Notar que una vez conectado, **mysql_real_connect()** activa la opción de reconexión (parte de la estructura MYSQL) a un valor de 1 en versiones del API anteriores a 5.0.3, y de 0 en versiones más recientes. Un valor 1 para esta opción indica, en el caso de que una consulta no pueda ser completada por una pérdida de conexión, se intente reconectar al servidor antes de abandonar.

Función mysql_real_escape_string()

```
unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char
*from, unsigned long length)
```

El parámetro *mysql* debe ser una conexión abierta válida. Esto es necesario ya que el escapado depende del conjunto de caracteres en uso por el servidor.

Esta función se usa para crear una cadena SQL legal que se puede usar en una sentencia SQL.

La cadena en *from* se codifica como una cadena SQL escapada, teniendo en cuenta el conjunto de caracteres actual de la conexión. El resultado se coloca en *to* y se añade un carácter nulo terminador. Los caracteres codificados son NUL (ASCII 0), '\n', '\r', '\', ''', '"' y Control-Z. (Estrictamente hablando, MySQL sólo requiere que se escapen los caracteres de barra invertida y el carácter de comilla usado para entrecomillar la cadena. Esta función entrecomilla los otros caracteres para que sean más fáciles de leer en ficheros de diario.)

La cadena apuntada por *from* debe tener *long* bytes de longitud. Además, se debe crear un buffer *to* con al menos *length**2+1 bytes de longitud. (En el peor caso, cada carácter necesitará ser codificado usando dos bytes, y se necesita espacio para el carácter terminador nulo.) Cuando **mysql_real_escape_string()** regresa, el contenido de *to* será una cadena terminada en nulo. El valor de retorno es la longitud de la cadena codificada, sin incluir el carácter nulo terminador.

Ejemplo

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"Qué es esto",11);
*end++ = '\'';
*end++ = ',';
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"datos binarios: \0\r\n",19);
*end++ = '\'';
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
```

```
    fprintf(stderr, "Fallo al insertar fila, Error: %s\n",
            mysql_error(&mysql));
}
```

La función **strmov()** usada en el ejemplo está incluida en la librería mysqlclient y trabaja igual que **strcpy()** pero devuelve un puntero al terminador nulo del primer parámetros.

Valores de retorno

La longitud del valor colocado en *to*, sin incluir el carácter nulo terminador.

Errores

Ninguno.

Función mysql_real_query()

```
int mysql_real_query(MYSQL *mysql, const char *query, unsigned long length)
```

Ejecuta la consulta SQL apuntada por *query*, que debe ser una cadena de *length* bytes de longitud. Normalmente, la cadena debe consistir en una sentencia SQL simple y no se debe añadir el punto y coma (`;') terminador o \g a la sentencia. Si la ejecución de múltiples sentencias está permitida, la cadena puede contener varias sentencias separadas con punto y coma.

Se debe usar **mysql_real_query()** en lugar de mysql_query() para consultas que contengas datos binarios, porque los datos binarios pueden contener el carácter '\0'. Además, **mysql_real_query()** es más rápido que mysql_query() porque no llama a **strlen()** para la cadena *query*.

Si se quiere saber si la consulta debe devolver un conjunto de resultados, se puede usar mysql_field_count() para comprobarlo.

Valores de retorno

Cero si la consulta tuvo éxito. Distinto de cero si se produjo algún error.

Errores

CR_COMMANDS_OUT_OF_SYNC: Los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: El servidor MySQL no está presente.

CR_SERVER_LOST: La conexión al servidor se perdió durante la consulta.

CR_UNKNOWN_ERROR: Se ha producido un error desconocido.

Función mysql_reload()

```
int mysql_reload(MYSQL *mysql)
```

Obliga al servidor MySQL a recargar las tablas de privilegios. El usuario conectado debe tener el privilegio *RELOAD*.

Esta función está desaconsejada. Es preferible usar mysql_query() para lanzar una sentencia SQL FLUSH PRIVILEGES en su lugar.

Valores de retorno

Cero si tiene éxito. Distinto de cero si se produjo un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: Los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: El servidor MySQL no está presente.

CR_SERVER_LOST: La conexión al servidor se perdió durante la consulta.

CR_UNKNOWN_ERROR: Se ha producido un error desconocido.

Tipo MYSQL_RES

```
typedef struct st_mysql_res {
  my_ulonglong row_count;
  MYSQL_FIELD   *fields;
  MYSQL_DATA    *data;
  MYSQL_ROWS    *data_cursor;
  unsigned long *lengths;              /* column lengths of current row */
  MYSQL         *handle;               /* for unbuffered reads */
  MEM_ROOT      field_alloc;
  unsigned int  field_count, current_field;
  MYSQL_ROW     row;                   /* If unbuffered read */
  MYSQL_ROW     current_row;           /* buffer to current row */
  my_bool       eof;                   /* Used by mysql_fetch_row */
  /* mysql_stmt_close() had to cancel this result */
  my_bool       unbuffered_fetch_cancelled;
  const struct st_mysql_methods *methods;
} MYSQL_RES;
```

Función mysql_rollback()

```
my_bool mysql_rollback(MYSQL *mysql)
```

Vuelva atrás la transacción actual.

Esta función se añadió en MySQL 4.1.0.

Valores de retorno

Cero si tiene éxito. Distinto de cero si ocurre algún error.

Errores

Ninguno.

Tipo MYSQL_ROW

```
typedef char **MYSQL_ROW;                /* return data as array of strings
*/
```

Tipo MYSQL_ROWS

```
typedef struct st_mysql_rows {
  struct st_mysql_rows *next;           /* list of rows */
  MYSQL_ROW data;
  unsigned long length;
} MYSQL_ROWS;
```

Función mysql_row_seek()

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
```

Coloca el cursor de fila en una fila arbitraria dentro de un conjunto de resultados de una consulta. El valor *offset* es un desplazamiento que debe ser devuelto por mysql_row_tell() o por mysql_row_seek(). Este valor no es un número de fila; si se quiere situar en una fila dentro de un conjunto de resultados mediante un número, usar mysql_data_seek() en su lugar.

Esta función requiere que la estructura del conjunto de resultados contenga el resultado completo de una consulta, de modo que **mysql_row_seek()** sólo puede ser usado en conjunción con mysql_store_result(), y no con mysql_use_result().

Valores de retorno

El valor previo del cursor de fila. Este valor puede ser pasado a una llamada sucesiva a **mysql_row_seek()**.

Errores

Ninguno.

Función mysql_row_tell()

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

Devuelve la posición actual del cursor de fila para la última llamada a mysql_fetch_row(). Este valor puede ser usado como argumento en una llamada a mysql_row_seek().

Se debe usar **mysql_row_tell()** sólo después de una llamada a mysql_store_result(), no después de mysql_use_result().

Valores de retorno

El desplazamiento actual del cursor de fila.

Errores

Ninguno.

Función mysql_select_db()

```
int mysql_select_db(MYSQL *mysql, const char *database);
```

Hace que la base de datos *database* especificada se convierta en la base de datos por defecto (actual) en la conexión especificada mediante *mysql*. En consultas posteriores, esta base de datos será la usada para referencias a tablas que no incluyan un especificador de base de datos.

mysql_select_db falla a no ser que el usuario conectado pueda ser autentificado como poseedor de permisos para usar la base de datos.

Valor de retorno

Cero si la operación ha tenido éxito, otro valor si no es así.

Errores

CR_COMMANDS_OUT_OF_SYNC: los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: el servidor MySQL no está presente.

CR_SERVER_LOST: la conexión con el servirod se ha perdido durante la consulta.

CR_UNKNOWN_ERROR: ha ocurrido un error desconocido.

Otros errores:

ER_BAD_DB_ERROR: la base de datos especificada no existe.

Función mysql_set_server_option()

```
int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option
option)
```

Activa o desactiva una opción para la conexión *mysql*. *option* puede ser uno de los siguientes valores:

MYSQL_OPTION_MULTI_STATEMENTS_ON	Activar soporte para sentencias múltiples.
MYSQL_OPTION_MULTI_STATEMENTS_OFF	Desactivar soporte para sentencias múltiples.

Esta función fue añadida en MySQL 4.1.1.

Valores de retorno

Cero si tiene éxito, distinto de cero si se produjo un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: Los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: El servidor MySQL no está presente.

CR_SERVER_LOST: La conexión al servidor se perdió durante la consulta.

CR_UNKNOWN_COM_ERROR: El servidor no soporta **mysql_set_server_option()** (este es el caso cuando el servidor es anterior a 4.1.1) o el servidor no soporta la opción que se intenta activar.

Función mysql_shutdown()

```
int mysql_shutdown(MYSQL *mysql, enum enum_shutdown_level shutdown_level)
```

Indica al servidor de bases de datos que se apague. El usuario conectado debe tener el privilegio *SHUTDOWN*. El argumento de *shutdown_level* se añadió en MySQL 4.1.3 (y 5.0.1). El servidor MySQL actualmente soporta sólo un tipo (nivel de cortesía) de apagado; *shutdown_level* debe ser igual a SHUTDOWN_DEFAULT. Más adelante se añadirán más niveles y entonces el argumento *shutdown_level* permitirá elegir el nivel deseado. Los servidores MySQL y clientes MySQL anteriores y posteriores a 4.1.3 son compatibles; los servidore MySQL más recientes que 4.1.3 aceptan llamadas **mysql_shutdown(MYSQL *mysql)**, y servidores MySQL anteriores a 4.1.3 aceptan la nueva llamada **mysql_shutdown()**. Pero los ejecutable enlazados dinámicamente que hayan sido compilados con versiones de cabeceras antiguar de libmysqlclient, y que llamen a **mysql_shutdown()**, tienen que ser usados con la vieja librería dinámica libmysqlclient.

Valores de retorno

Cero si tiene éxito, distinto de cero si se produjo un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: Los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: El servidor MySQL no está presente.

CR_SERVER_LOST: La conexión al servidor se perdió durante la consulta.

CR_UNKNOWN_ERROR: Se ha producido un error desconocido.

Función mysql_sqlstate()

```
const char *mysql_sqlstate(MYSQL *mysql)
```

Devuelve una cadena terminada en null que contiene el código de error SQLSTATE para el último error. El código de error consiste en cinco caracteres. '00000' significa "sin error". Los valores están especificados por ANSI SQL y ODBC.

Hay que tener en cuenta que no todos los errores de MySQL has sido ya mapeados a errores SQLSTATE. El valor 'HY000' (error general) se usa para errores sin mapear.

Esta función fue añadida en MySQL 4.1.1.

Valores de retorno

Una cadena terminada en nulo que contiene el código de error SQLSTATE.

Ver también

mysql_errno(), mysql_error() y mysql_stmt_sqlstate().

Función mysql_ssl_set()

```
int mysql_ssl_set(MYSQL *mysql, const char *key,
  const char *cert, const char *ca, const char *capath, const char *cipher)
```

mysql_ssl_set() se usa para establecer conexiones seguras usando SSL. Debe ser llamada antes de mysql_real_connect().

mysql_ssl_set() no hace nada a no ser que el soporte OpenSSL esté activo en la librería del cliente.

mysql es el manipulador de conexión devuelto por from mysql_init(). El resto de los parámetros se especifican como sigue:

- *key* es el camino para el fichero llave.
- *cert* es el camino para el fichero de certificado.
- *ca* es el camino para el fichero de certificado de autoridad.
- *capath* es el camino al directorio que contiene los certificados de confianza SSL CA en formato pem.
- *cipher* es una lista de claves disponibles para usar para encriptado SSL.

Cualquier parámetro SSL sin usar debe ser dado como NULL.

Valores de retorno

Esta función siempre devuelve 0. Si el sistema SSL es incorrecto, **mysql_real_connect()** devolverá un error cuando se intente la conexión.

Función mysql_stat()

```
char *mysql_stat(MYSQL *mysql)
```

Devuelve una cadena de caracteres que contiene información similar a la proporcionada por el comando *mysqladmin status*. Esto incluye el tiempo activo en segundos y el número de procesos en ejecución, preguntas, recargas y tablas abiertas.

Valores de retorno

Una cadena de caracteres que describe el estado del servidor. NULL si se produce un error.

Errores

CR_COMMANDS_OUT_OF_SYNC: Los comandos fueron ejecutados en un orden inapropiado.

CR_SERVER_GONE_ERROR: El servidor MySQL no está presente.

CR_SERVER_LOST: La conexión al servidor se perdió durante la consulta.

CR_UNKNOWN_ERROR: Se ha producido un error desconocido.

Función mysql_store_result()

```
MYSQL_RES *mysql_store_result(MYSQL *mysql);
```

Se debe usar **mysql_store_result()** o mysql_use_result() para cada consulta que haya recuperado datos (SELECT, SHOW, DESCRIBE, EXPLAIN, CHECK TABLE, etc).

No es necesario llamarlas para otras consultas, pero no harán ningún daño ni resultará una pérdida apreciable si se llama a **mysql_store_result()** en todos los casos. Se puede comprobar si la consulta no ha proporcionado resultados comprobando si **mysql_store_result()** devuelve cero.

Si se quiere saber si una consulta debe devolver un conjunto de resultados o no, se puede usar mysql_field_count().

mysql_store_result() lee el resultado total de una consulta al cliente, asigna una estructura MYSQL_RES y coloca el resultado en esa estructura.

mysql_store_result() devuelve un puntero nulo si la consulta no ha devuelto un conjunto de resultados (si la consulta ha sido, por ejemplo, una sentencia INSERT).

mysql_store_result() también devuelve un puntero nulo si la lectura del conjunto de resultados falla. Se puede verificar si se trata de un error comprobando si mysql_error() devuelve una cadena no vacía, si mysql_errno() devuelve un valor distinto de cero o si mysql_field_count() devuelve un valor cero.

Se obtiene un conjunto de resultados vacío si no hay filas devueltas. (No es lo mismo un conjunto de resultados vacío que un puntero nulo como valor de retorno.)

Una vez que se ha llamado a **mysql_store_result()** y se ha obtenido un resultado que no sea un puntero nulo, se puede llamar a mysql_num_rows() para encontrar cuántas filas hay en el conjunto de resultados.

También se puede llamar a mysql_fetch_row() para recuperar filas desde el conjunto de resultadost, o mysql_row_seek() y mysql_row_tell() para obtener o modificar la posición actual de la fila dentro del conjunto de resultados.

Se debe llamar a mysql_free_result() una vez que se ha terminado de procesar el conjunto de resultados.

A veces **mysql_store_result()** devuelve NULL después aunque mysql_query() regrese con éxito.

- Se ha producido un error al reservar memoria dinámica, por ejemplo, si el conjunto de resultados es demasiado grande.
- Los datos no pudieron ser leídos, si ha ocurrido un error durante la conexión.
- La consulta no ha devuelto datos, por ejemplo, una consulta de tipo INSERT, UPDATE o DELETE).

Valores de retorno

Una estructura MYSQL_RES con los resultados o NULL si se ha producido un error.

Errores

mysql_store_result() resetea mysql_error() y mysql_errno() si tiene éxito.

CR_COMMANDS_OUT_OF_SYNC: los comandos fueron ejecutados en un orden inapropiado.

CR_OUT_OF_MEMORY: no hay memoria.

CR_SERVER_GONE_ERROR: el servidor MySQL no está presente.

CR_SERVER_LOST: la conexión con el servidor se perdió durante la consulta.

CR_UNKNOWN_ERROR: ha ocurrido un error desconocido.

Función mysql_thread_id()

```
unsigned long mysql_thread_id(MYSQL *mysql)
```

Devuelve el ID del proceso para la conexión actual. Este valor puede ser usado como argumento en la función mysql_kill() para matar el proceso.

Si la conexión se pierde y se reconecta con mysql_ping(), el ID del proceso cambiará. Esto significa que no se debe leer el ID del proceso y almacenarlo para usarlo más tarde. Se debe obtener en el momento en que se necesite.

Valores de retorno

El ID del proceso de la conexión actual.

Errores

Ninguno.

Función mysql_use_result()

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

Se debe usar mysql_store_result() o **mysql_use_result()** para cada consulta que haya recuperado datos (SELECT, SHOW, DESCRIBE, EXPLAIN, CHECK TABLE, etc).

mysql_use_result() inicia una recuperación de un conjunto de resultados, pero no lee el conjunto de resultados en el cliente, como hace mysql_store_result(). En su lugar, cada fila debe ser recuperada individualmente mediante llamadas a mysql_fetch_row(). Esto lee el resultado de una consulta directamente desde el servidor sin almacenarlo en una tabla temporal o en un buffer local, lo que es algo más rápido y requiere mucha menos memoria que mysql_store_result(). El cliente reservará memoria sólo para la fila actual y un buffer de comunicación que puede crecer hasta un máximo de max_allowed_packet bytes.

Por otra parte, no se debe usar **mysql_use_result()** si se va a hacer mucho proceso para cada fila en el lado del cliente, o si la salida se envía a una pantalla en la que el usuario puede usar ^S (parar el desplazamiento). Esto bloqueará el servidor e impide que otros procesos puedan actualizar las tablas para las que los datos están siendo recuperados.

Cuando se usa **mysql_use_result()**, se debe ejecutar mysql_fetch_row() hasta que devuelva un valor NULL, en caso contrario, las filas no recuperadas se devolverán como parte del conjunto de resultados de la siguiente consulta. El API C obtendrá un error "Commands out of sync; you can't run this command now if you forget to do this!".

No se debe usar mysql_data_seek(), mysql_row_seek(), mysql_row_tell(), mysql_num_rows() o mysql_affected_rows() con un resultado devuelto desde **mysql_use_result()**, tampoco se deben lanzar otras consultas hasta que **mysql_use_result()** haya finalizado. (Sin embargo, después de que se hayan recuperado todas las filas, mysql_num_rows() devolverá con precisión el número de filas recuperadas.)

Se debe llamar a mysql_free_result() una vez que se haya terminado de procesar el conjunto de resultados.

Valores de retorno

Una estructura de resultado MYSQL_RES. NULL si se produce un error.

Errores

mysql_use_result() resetea mysql_error y mysql_errno si tiene éxito.

CR_COMMANDS_OUT_OF_SYNC: los comandos fueron ejecutados en un orden inapropiado.

CR_OUT_OF_MEMORY: no hay memoria.

CR_SERVER_GONE_ERROR: el servidor MySQL no está presente.

CR_SERVER_LOST: la conexión con el servidor se perdió durante la consulta.

CR_UNKNOWN_ERROR: ha ocurrido un error desconocido.

Función mysql_warning_count()

```
unsigned int mysql_warning_count(MYSQL *mysql)
```

Devuelve el número de avisos generados durante la ejecución de la sentencia SQL previa.

Esta función fue añadida en MySQL 4.1.0.

Valores de retorno

El contador de avisos.

Errores

Ninguno.