Acceso a **MySQL** desde **Visual C++**

Ivan Cachicatari Poma

http://www.latindevelopers.com/ivancp/

ivancp@latindevelopers.com

Versión 1.5, 23 Marzo 2007

El presente articulo muestra paso a paso como crear una pequeña aplicación en Visual C++ 6.0 con acceso a una base de datos de MySQL, utilizando librerías proveídas e incluidas en las distribuciones del motor de base de datos MySQL.

latindevelopers.com

Índice

1. Introducción

Las diferentes alternativas que tenemos los desarrolladores para implementar aplicaciones de acceso a base de datos nos hacen pensar dos veces antes de elegir el cómo y con qué lenguaje nos conectamos a una base de datos. El mismo dilema se presenta al elegir el gestor de base de datos, al principio es tedioso, pero cuando uno empieza a tener confianza no duda en seguir utilizando su gestor favorito.

Este tutorial –aunque pequeño– intenta mostrar la puerta al mundo de posibilidades que existe al utilizar Visual C++ como lenguaje de programación y MySQL como gestor de datos. Les guiaré paso a paso como deben construir su primera aplicación de acceso a base de datos con Visual C++ utilizando las librerías API –escritas en C– que provee MySQL.

Empezaremos alistando los datos sobre un servidor de datos MySQL ya instalado y corriendo para luego crear un proyecto en Visual C++ 6.0 basado en dialogo el cual configuraremos inicialmente para utilizas las librerías API de MySQL y finalmente escribiremos unas cuantas lineas de código.

2. Preparar los datos

Para éste tutorial preparé una tabla que en realidad es un extracto de otra tabla que tengo por ahí, ésta tabla contiene datos suficientes para experimentar con aplicación que vamos a desarrollar, la tabla contiene campos para almacenar datos basicos de una empresa como RUC, Razón Social, Email, etc. El comando SQL de creación de la tabla y tambien los datos estan en el archivo `prueba_db.sql` que pueden descargar de la dirección de Internet que se indica al final del tutorial.

Este archivo de comandos SQL deben ejecutarlo en el servidor MySQL que tengan instalado. Puede usted elegir la mejor forma de hacerlo. Le recomiendo que lo haga mediante linea de comando, es menos tedioso y mucho mas rápido:

Comando para ejecutar el archivo SQL utilizando la linea de comando:

```
# mysql -u root < prueba_db.sql
```

Si el usuario root necesita password:

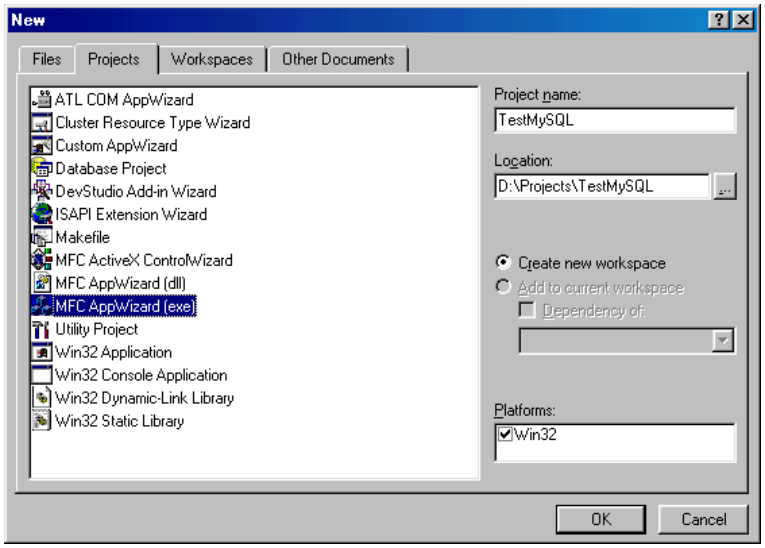
```
# mysql -u root -p < prueba_db.sql
```

Listado 1: Script de la tabla que contiene los datos en `prueba_db.sql`

```
 1:
 2:  CREATE DATABASE prueba_db;
 3:
 4:  USE prueba_db;
 5:
 6:  CREATE TABLE empresas (
 7:    RUC char(11) NOT NULL,
 8:    RazonSocial varchar(80),
 9:    Direccion varchar(60),
10:    Email varchar(50),
11:    Web varchar(60),
12:    PRIMARY KEY (RUC),
13:    KEY RazonSocial (RazonSocial)
14:  );
15:
```

3. Crear el proyecto

Posiblemente usted ya conozca el proceso de crear una aplicación basada en diálogo con Visual C++, si es asi entonces continúe con el siguiente paso.



Imágen 1. Dialogo de nuevo proyecto

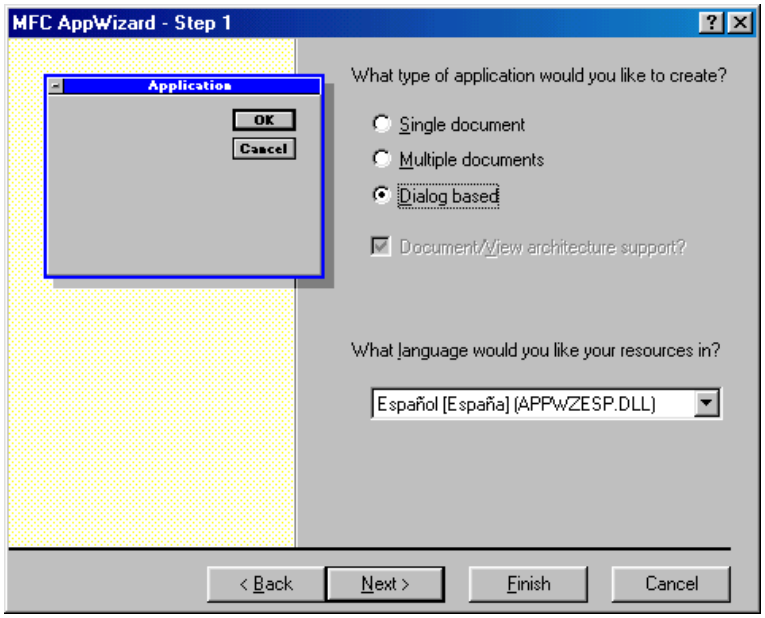
Bueno... procedemos a crear el proyecto mediante el *AppWizard* del Visual C++ 6.0, accedemos al menú: File -> New ó presionamos la combinación [Ctrl] + [N]. Aparecerá un diálogo parecido al de la Imágen 1.

Elegimos el tipo de proyecto *MFC AppWizard (.exe)*, para el proyecto elegí el

nombre “TestMySQL”, esto automáticamente creará una carpeta con todos los elementos necesarios, es recomendable que esta carpeta esté situada en un lugar estratégico, por un principio de orden.

En el siguiente dialogo elegimos el tipo de aplicación Basada en dialogo (*Dialog Based*). Ya que este es un proyecto cuyo objetivo es demostrar el uso de las librerías API de MySQL solo crearemos un proyecto de ese tipo por que no necesitamos todo el código que genera para las otras aplicaciones, en caso de que tenga que crear proyectos mas grandes puede escoger las otras opciones. Ver Imagen 2.

Una aplicación *Dialog Based* básicamente genera tres clases: una clase derivada de CWinApp que será la aplicación propiamente dicha, otra clase derivada de CDialog acompañada de un formulario en blanco para dar rienda suelta a nuestra imaginación y otra clase derivada de CDialog que contiene información acerca de la aplicación.



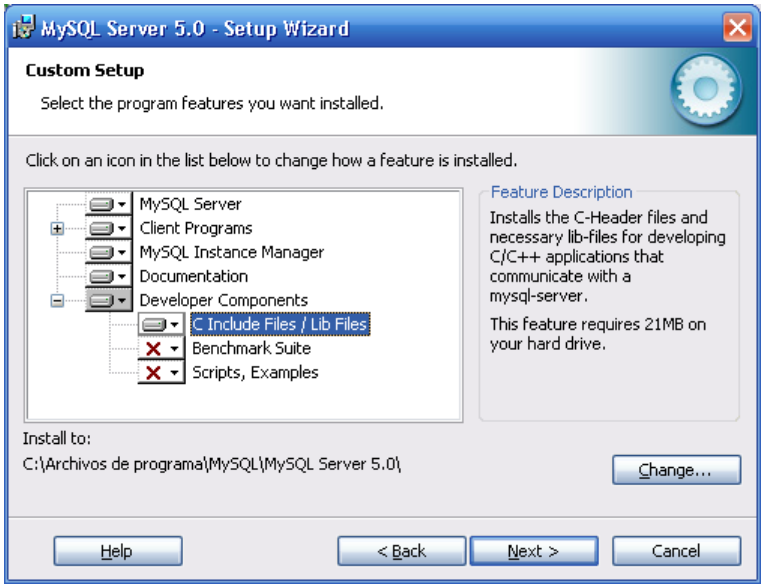
Imágen 2. AppWizard:Creando proyecto basado en dialogo

Los siguientes diálogos de opciones son irrelevantes a lo que necesitamos así es que click en *Finish* y nos olvidamos de este asunto de una buena vez. Aparecerá un diálogo que nos informará lo que estamos generando. Click en OK y con esto terminamos éste Paso.

4. Configuraciones y otros detalles

Antes de compilar el proyecto debemos asegurarnos que las configuraciones de acceso a los archivos de la librería API MySQL estén correctamente establecidas.

Al instalar una distribución de MySQL sobre Microsoft Windows debemos instalar también los Componentes de Desarrollo (*Developer Components*) específicamente la opción *C Include Files/ Lib Files* (Archivos de cabecera y librerías). Ver Imágen 3.



Imágen 3. Instalación de MySQL incluyendo las librerías API

En una instalación, por defecto, se instalan las librerías en rutas específicas:

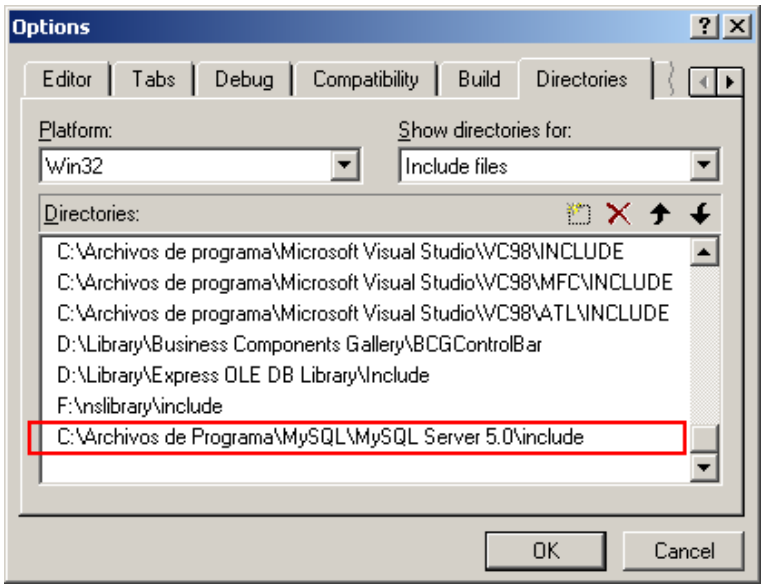
- Archivos de cabecera
  `c:\Archivos de Programa\MySQL\MySQL Server 5.0\include`
- Archivos de librería (.lib) con información para poder depurar
  `c:\Archivos de Programa\MySQL\MySQL Server 5.0\lib\debug`
- Archivos de librería (.lib) optimizados
  `c:\Archivos de Programa\MySQL\MySQL Server 5.0\lib\opt`

La carpeta `debug` contiene librerías para compilar nuestro proyecto en el modo *Debug* y la carpeta `opt` contiene los archivos para el modo *Release*. El primer modo (*Debug*) nos facilita la vida en la etapa de desarrollo pudiendo detectar problemas

y errores para poder corregirlos. El segundo modo de compilación se utiliza cuando estemos seguros que no hay mas problemas y generamos un ejecutable liviano y listo para el usuario final; aunque siempre se presenta algun maldito problema que no supimos prevenir.

Nota: No es necesario que intalen el servidor de datos completo, es posible que ustedes tengan un servidor MySQL corriendo en alguna otra parte de su red. Para ello pueden copiar las carpetas indicadas arriba desde otra instalación o simplemente instalar sólo los *Developer Components*

Dejemos el asunto ahí, sigamos con la configuración de las rutas. Ahora configuraremos Visual C++ para que agregue éstas rutas a su lista de carpetas de busqueda de archivos de inclusión (*Include Files*). Entonces debemos acceder a la Configuración del Proyecto (*Project Settings*) y agregar la ruta que corresponde Ver Imágen 4. Para accede a la Configuración del Proyecto ir a Menu: *Project -¿Setting* o presionar la combinación `[ALT]+[F7]`.

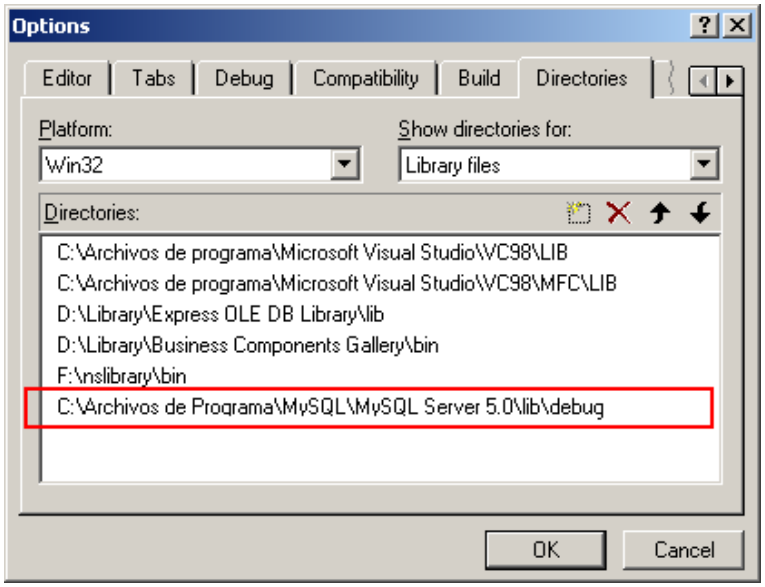


Imágen 4. Diálogo de configuración de rutas de inclusión

De éstas carpetas utilizaremos el archivo `mysql.h` en nuestras futuras aplicaciones, en éste archivo estan declaradas las funciones de conexión que nos proporciona MySQL para C.

Al igual que en el caso anterior también tenemos que agregar el directorio de librerías `lib`. El archivo `libmySQL.lib` ubicado en ésta carpeta nos sirve para enlazar nuestra aplicación, Visual C++ se encargará de enlazar todo para producir un ejecutable. Dentro de las carpetas `lib\debug` y `lib\opt` tambien existe el archivo `libmySQL.dll` el cual debemos copiarlo en la misma carpeta de nuestro

ejecutable para que pueda correr sin problemas. Pero debemos elegir el correcto, en este caso tomaremos el de la carpeta `lib\debug`, por que nuestro proyecto estará configurado en el modo *Debug* por defecto. Ver Imágen 5.



Imágen 5. Diálogo de configuración de librerías

Cuando tengan lista una aplicación para el usuario final deberán configurar el proyecto en el modo *Release* ó Lanzamiento e incluir en su distribución el archivo `libmySQL.dll` contenido en la carpeta `lib\opt`. Antes de compilar deben también configurar las rutas de las librerías hacia la carpeta `lib\opt`.

Otras configuraciones

Luego de hacer todas las tareas anteriores procedemos a establecer configuraciones adicionales:

Carpeta de salida. En la pestaña *General* de la ventana de Configuración del Proyecto ((*Project Settings*)) en la sección *Output files*: escribir `OutPut` como carpeta de salida. Esto le dice al compilador que los archivos ejecutables que produzca no los mezcle con los archivos intermedios y los "bote" en un directorio aparte (`OutPut`).

Carpeta de trabajo. La carpeta de trabajo es el lugar que tendrá nuestro programa como carpeta por defecto al ejecutarlo desde el Visual C++ (*Working directory*). En ésta carpeta debemos copiar el archivo *libmySQL.dll*. Por defecto el valor de esta propiedad esta en blanco.

Librerías. En la pestaña "Link" (de Configuración del Proyecto) agregar el texto libmySQL.lib en la sección *Object/Library modules*, ésta se usa para decirle

a compilador que utilice librerías extra al momento del la construcción final del ejecutable. Este paso nos evita ver problemas al momento de enlazar el proyecto (*Linker Errors*).

Archivos de Cabecera

Las declaraciones de las variables y funciones de MySQL C API estan en el archivo `mysql.h`, debemos declarar la inclusión de éste archivo en Stdafx.h, ver listado 2. Éstas líneas deben escribirse al final de todas las declaraciones `#include` existentes.

Listado 2: Incluir archivos en `Stdafx.h`

```
1:  #include <afxsock.h>
2:  #include <mysql.h>
```

Debido a que `mysql.h` utiliza conexiones de *Sockets* es necesario tambien incluir el archivo `afxsock.h` antes de `mysql.h` para que esten disponibles los *Window Sockets*.

5. La conexión a MySQL

En el Paso 2 creamos el proyecto, en el cual habrá notado que se han creado tres clases: `CAboutDlg`, `CTestMySQLApp` y `CTestMySQLDlg`. No tocaremos la primera clase ya que es un dialogo que no cumple otra función mas que informar acerca del proyecto.

En la clase `CTestMySQLApp` crearemos la conexión a la base de datos la que utilizaremos en todo el proyecto, de modo que será una **conexion global**.

En la clase `CTestMySQLDlg` realizaremos una consulta a la tabla `empresas` de la base de datos `prueba_db` y lo mostraremos en resultado en el control list del dialogo. Ver Imágen 6.

Pues manos a la obra, escribiremos nuestras primeras lineas de codigo asegurando la conexión a la base de datos la cual crearemos en la clase `CTestMySQLApp`, necesitaremos definir las variables y funciones como se indica en la Tabla 1.

En la variable `m_pLinkDb`, que inicialmente será `NULL` (Ver Listado 4), permanecerá la conexión a MySQL que obtendremos con `OpenConnection` al iniciar la aplicación. La función `GetConnection` nos servirá para el resto de objetos de la

Tipo	Tipo de dato	Nombre	Ámbito
Variable	`MYSQL*`	`m_pLinkDb`	protected
Función	`MYSQL*`	`GetConnection()`	public
Función	`bool`	`OpenConnection()`	protected
Función Virtual	`int`	`ExitInstance()`	public

Tabla 1: Variables y funciones para la clase `CTestMySQLApp`

aplicación que necesiten utilizar una conexion a MySQL (Diálogos, Vistas, Clases, etc.).

Podremos acceder a la conexión desde cualquier parte a través de la variable global `theApp`. Es por eso que la variable `m_pLinkDb` tiene que estar protegida, para evitar utilizarla negligentemente sin que esté inicializada. Ver Listado10 línea 14.

Luego del proceso de agregar las variables y funciones de la Tabla 1. Nuestra clase `CTestMySQLApp` quedará como el Listado 3.

Listado 3: Declaración de la clase `CTestMySQLApp`

```
 1:  class CTestMySQLApp : public CWinApp
 2:  {
 3:
 4:    protected:
 5:     bool OpenConnection();
 6:     MYSQL* m_pLinkDb;
 7:
 8:    public:
 9:     MYSQL* GetConnection();
10:     CTestMySQLApp();
11:
12:    // Overrides
13:     // ClassWizard generated virtual function overrides
14:     //{{AFX_VIRTUAL(CTestMySQLApp)
15:     public:
16:     virtual BOOL InitInstance();
17:     virtual int ExitInstance();
18:     //}}AFX_VIRTUAL
19:
20:    // Implementation
21:
22:     //{{AFX_MSG(CTestMySQLApp)
23:
24:
25:     //}}AFX_MSG
26:     DECLARE_MESSAGE_MAP()
27:  };
```

Listado 4: Constructor de la clase `CTestMySQLApp`

```
1:  CTestMySQLApp::CTestMySQLApp()
2:  {
3:     m_pLinkDb = NULL;
4:  }
```

La función `OpenConnection`

Ésta función es muy importante por que depende de ésta si seguimos con el programa o nos salimos de él sin antes mostrar un mensaje de error.

`OpenConnection` debe invocarse desde `InitInstance` de la Aplicación. Ya que necesitamos que la conexión se establezca al iniciar el programa. En la función `OpenConnection` es donde inicializamos la variable `m_pLinkDb` (Líneas 3-4) para luego conectarnos al servidor MySQL con la función de la API de MySQL `mysql_real_connect` (Líneas 11,19). Si ocurre algun error devolvemos `false` mostrando antes el mensaje de error (Líneas 21,27). Ver Listado 5.

Listado 5: Implementación de la función `OpenConnection`

```
 1:  bool CTestMySQLApp::OpenConnection()
 2:  {
 3:     m_pLinkDb = new MYSQL;
 4:     mysql_init(m_pLinkDb);
 5:
 6:     CString sHost       = _T("localhost");
 7:     CString sUser       = _T("root");
 8:     CString sPassword   = _T("");
 9:     CString sDatabase   = _T("prueba_db");
10:
11:     if (!mysql_real_connect(
12:           m_pLinkDb,
13:           sHost,
14:           sUser,
15:           sPassword,
16:           sDatabase,
17:           3306, // Puerto de conexion
18:           NULL, // Unix socket
19:           0))   // Otras opciones del cliente
20:     {
21:        CString strText;
22:
23:        strText.Format("Error: '%s'"
24:           ,mysql_error(m_pLinkDb));
25:
26:        AfxMessageBox(strText);
27:        return false;
28:     }
```

```
29:     return true;
30:  }
```

Usted puede guardar y obtener los parametros de conexión del lugar que guste –el registro de Windows por ejemplo–, es por eso que he declarado las variables de las líneas 6-9.

En la función `InitInstance` invocamos a `OpenConnection` pero verificamos que el resultado sea el que esperamos, sino retornamos `FALSE` y asi evitamos que la aplicación siga. Ver Listado 6.

Listado 6: Abriendo la conexión con `OpenConnection`

```
 1:  BOOL CTestMySQLApp::InitInstance()
 2:  {
 3:
 4:     ...
 5:
 6:     if (!OpenConnection())
 7:     {
 8:        return FALSE;
 9:     }
10:
11:     ...
12:  }
```

Cuando salgamos de la aplicación tendremos que cerrar la conexión. Al salir de una aplicación basada en MFC se invoca a la función `ExitInstance` y ahí es donde tenemos que escribir el codigo necesario para ello. Ver Listado 7.

Listado 7: Cerrando la conexión en `ExitInstance`

```
 1:  int CTestMySQLApp::ExitInstance()
 2:  {
 3:     if (m_pLinkDb)
 4:     {
 5:        mysql_close(m_pLinkDb);
 6:     }
 7:     return CWinApp::ExitInstance();
 8:  }
```

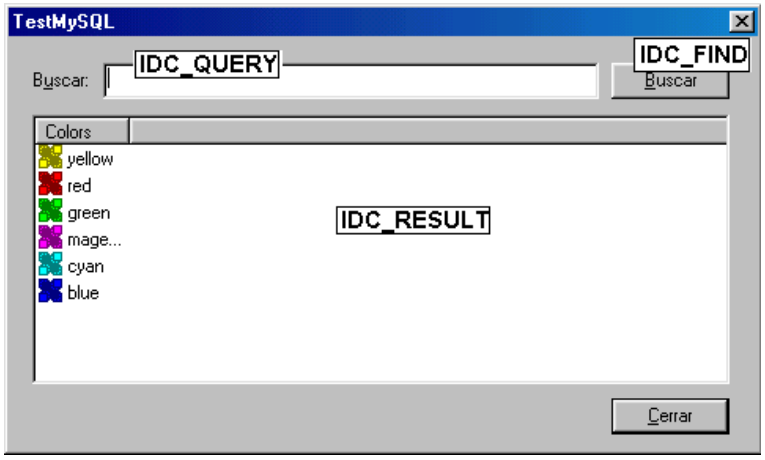
Ahora finalmente vamos a la función `GetConnection`, ésta función nos retornará un apuntador hacia la conexión global, antes de retornar dicho apuntador debemos asegurarnos que la conexión sea valida, para ello se utiliza la macro `ASSERT()`, esta macro evalúa el parametro, si es `FALSE` ó `NULL` provoca un fallo. Ver Listado 8. Con esto concluimos el asunto de la conexión a la base de datos.

Listado 8: Implementación de la función `GetConnection`

```
1:  MYSQL* CTestMySQLApp::GetConnection()
2:  {
3:     ASSERT(m_pLinkDb);
4:
5:     return m_pLinkDb;
6:  }
```

6. Preparar el diálogo

Desde la vista de recursos en la sección *Dialogs* ubicar y abrir el dialogo de la aplicación (en nuestro caso es el dialogo que tiene el ID: `IDD_TESTMYSQL_DIALOG`) para modificarlo. Agregarle y ubicar los controles hasta que quede como la Imagen 6. A cada uno de los controles le asignaremos un ID como la Tabla 2. En la misma tabla se indica los eventos y funciones que tambien debemos agregar a la clase `CTestMySQLDlg`.



Imágen 6. Formulario utilizado para la aplicación

Nota: Para agregar facilmente una variable relacionada a un control: hacer doble click sobre él manteniendo presionada la tecla `[CTRL]`, luego asignarle el nombre y tipo de variable. El ámbito de las variables y funciones serán creados automáticamente por el *ClassWizard*.

La función `OnFind()` es invocada cuando el usuario hace click sobre el boton `IDC_FIND`. Asu vez la función `OnFind()` invocará a `LoadData()` Ver Listado 9:15-18. Tambien tenemos que hacer una llamada a `LoadData()` desde la función `OnInitDialog()`. Ver Listado 9:10.

Tipo	Tipo de dato	Control	Evento	Nombre
Variable	CListCtrl	IDC_RESULT		m_report
Variable	CString	IDC_QUERY		m_sFilter
Función	void			LoadData()
Función	void	IDC_FIND	BN_CLICKED	OnFind()

Tabla 2: Variables y funciones para la clase CTestMySQLDlg

Listado 9: Funciones de la clase CTestMySQLDlg

```
 1:  BOOL CTestMySQLDlg::OnInitDialog()
 2:  {
 3:     CDialog::OnInitDialog();
 4:
 5:     ...
 6:
 7:     m_report.InsertColumn(0,_T("RUC"),LVCFMT_CENTER,90);
 8:     m_report.InsertColumn(1,_T("Razon social"),LVCFMT_LEFT,300);
 9:
10:     LoadData();
11:
12:     return TRUE;
13:  }
14:
15:  void CTestMySQLDlg::OnFind()
16:  {
17:     LoadData();
18:  }
```

La función `LoadData()` se encargará de cargar los datos y los irá volcando en el control `IDC_RESULT`(Ver Listado 10:13-44), pero antes de cargarlos ensamblará una consulta SQL para ello. La consulta SQL la ejecutaremos a través de la conexión global y contendrá en clausula `WHERE` una sentencia que nos ayudará a buscar registros basados en el contenido del control `IDC_QUERY` (Ver Listado 10:5-11).

Listado 10: La función LoadData

```
 1:  void CTestMySQLDlg::LoadData()
 2:  {
 3:     CWaitCursor x;
 4:
 5:     UpdateData();
 6:
 7:     CString sql;
 8:
 9:     sql.Format("SELECT ruc,razonsocial \
10:              FROM empresas \
11:              WHERE razonsocial LIKE '%s%%'",m_sFilter);
```

```
12:
13:     if(mysql_real_query(
14:        theApp.GetConnection()
15:        ,sql
16:        ,sql.GetLength()) == 0 )
17:     {
18:
19:        MYSQL_RES* res;
20:
21:        if ((res = mysql_store_result(theApp.GetConnection())))
22:        {
23:
24:           m_report.DeleteAllItems();
25:
26:           MYSQL_ROW Row;
27:           CString sRuc;    // Ruc
28:           CString sRazSoc; // Razon Social
29:           int item = 0;
30:
31:           while ((Row = mysql_fetch_row(res)))
32:           {
33:
34:              sRuc.Format ("%s", Row[0] ? (char*) Row[0] : "NULL");
35:              m_report.InsertItem(item,sRuc);
36:
37:              sRazSoc.Format ("%s", Row[1] ? (char*) Row[1] : "NULL");
38:              m_report.SetItemText(item,1,sRazSoc);
39:
40:              item++;
41:           }
42:           mysql_free_result(res);
43:        }
44:     }
45:     else
46:     {
47:        CString strText;
48:
49:        strText.Format("Error: '%s'"
50:           ,mysql_error(theApp.GetConnection()));
51:
52:        AfxMessageBox(strText);
53:     }
54:  }
```

Descripción de las funciones API de MySQL

`mysql_real_query`: Ejecuta una consulta SQL sobre una conexión dados pasadas como parametros, el parámetro de consulta necesita la longitud de la consulta en caracteres.

```
int mysql_real_query(
      MYSQL *mysql, // enlace a la base de datos
      const char *query, // Consulta
      unsigned long length ); // Longitud de la consulta
```

`mysql_store_result`: Retorna todos los registros de la ultima consulta ejecutada a una estructura `MYSQL_RES`. Retorna NULL si no hay resultados que mostrar.

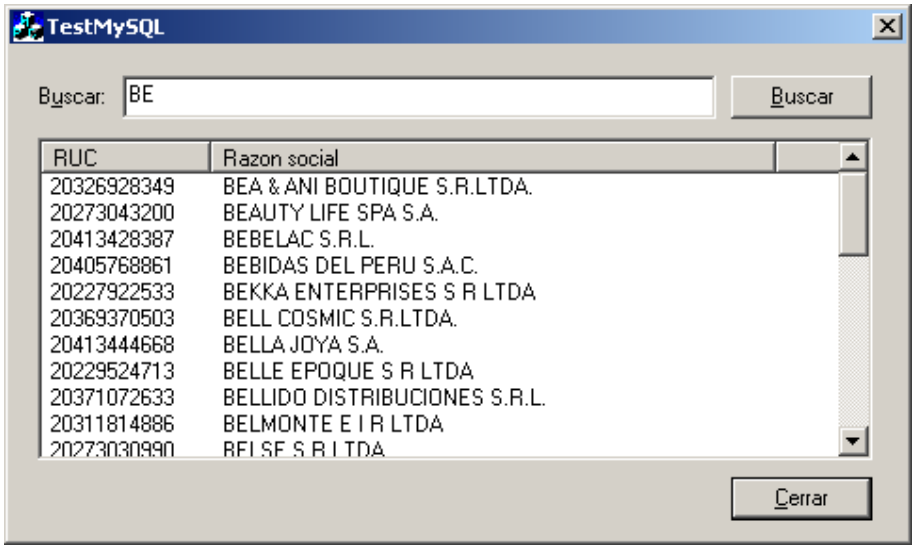
```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

`mysql_fetch_row`: Retorna el siguiente registro de una consulta, este método debe ser llamado después de `mysql_store_result`. Retorna NULL si no hay mas registros.

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

`mysql_free_result`: Libera la memoria ocupada por el resultado de una consulta obtenida mediante `mysql_store_result`.

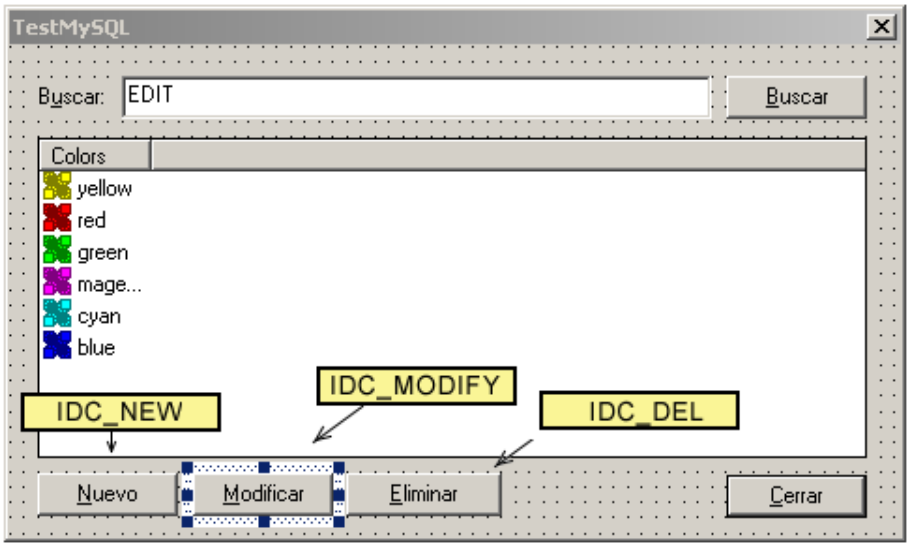
```
void mysql_free_result(MYSQL_RES *result)
```



Imágen 7. Aplicación ejecutandose

7. Modificar, Agregar, Eliminar datos

Si bien logramos recuperar datos, éstos no nos servirán de nada si no podemos realizar las otras operaciones: Insertar, Modificar, Eliminar. Ése va a ser el punto de esta sección del tutorial. Basicamente tendremos que agregar tres botones para realizar dichas operaciones en la Imagen 8 pueden apreciar los botones ya insertados.



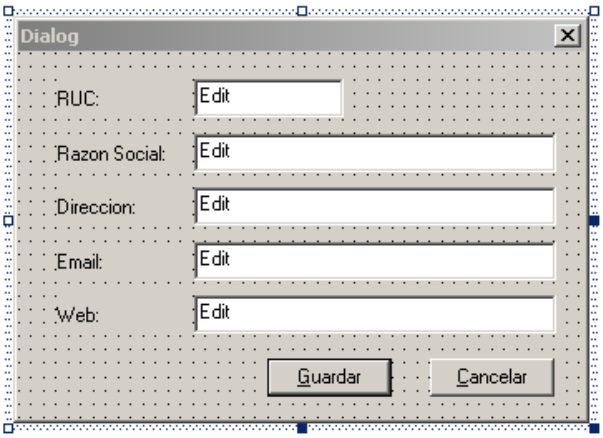
Imágen 8. Botones adicionales al dialogo

Antes de escribir el codigo de las instrucciones de los botones nuevos, tenemos que crear un dialogo para poder dar de alta o modificar un registro en la base de datos. En el dialogo actual sólo muestra dos campos (ruc, razonsocial) pero la tabla `empresas` tiene 5 (Ver Listado 1). Para ello es diálogo de la Imagen 9, a éste dialogo le puse el ID: `IDD_EMPRESA` y mediante *ClassWizard* creé la clase relacionada con éste: `CEmpresaDlg`.

La clase `CEmpresaDlg` debe tener las variables, eventos y funciones que se indican en la Tabla 3.

Las variables de tipo CString estan elazadas a un control y pueden agregarse utilizando el procedimiento que se indica en la Nota de la Imagen 6. La variable `m_bIsNew` nos servirá como *flag* para saber si estamos agregando o modificando. En la función `LoadData()` cargaremos los datos en caso de que estemos modificando. Utilizaremos `Guardar` para actualizar o dar de alta un registro de la base de datos.

Las función `OnOK` puede agregarse haciendo doble click en el boton con el ID: `IDOK`. La función `OnInitDialog` se agrega automaticamente ubicando el mensaje



Imágen 9. Diseño del dialogo para editar/agregar datos

Tipo	Tipo de dato	Control	Nombre	Ámbito
Variable	`CString`	`IDC_RUC`	`m_sRUC`	public
Variable	`CString`	`IDC_RAZONSOCIAL`	`m_sRazonSocial`	public
Variable	`CString`	`IDC_DIRECCION`	`m_sDireccion`	public
Variable	`CString`	`IDC_EMAIL`	`m_sEmail`	public
Variable	`CString`	`IDC_WEB`	`m_sWeb`	public
Variable	`bool`		`m_bIsNew`	public
Función	`void`		`LoadData()`	protected
Función	`bool`		`Guardar()`	protected
Función	`void`		`OnOK()`	protected
Función	`BOOL`		`OnInitDialog()`	protected

Tabla 3: Variables y funciones para la clase `CEmpresaDlg`

`WM_INITDIALOG` desde la pestaña *Message Maps* del *ClassWizard*, luego de ubicar el mensaje hacer click en *Add Function*.

Para terminar tenemos que modificar el contructor del dialogo `CEmpresaDlg` para que nos acepte un parametro CString, donde le pasaremos el RUC en caso de que quisieramos modificar un registro. Luego de todas esas modificaciones, la clase `CEmpresaDlg` debería quedar como el Listado 11.

Listado 11: Declaracion de la clase `CEmpresaDlg`

```
1:  class CEmpresaDlg : public CDialog
2:  {
3:  // Construction
4:  public:
5:     CEmpresaDlg(CString sRUC,CWnd* pParent = NULL);
6:
7:  // Dialog Data
8:     //{{AFX_DATA(CEmpresaDlg)
```

```
 9:     enum { IDD = IDD_EMPRESA };
10:     CString   m_sRUC;
11:     CString   m_sRazonSocial;
12:     CString   m_sDireccion;
13:     CString   m_sEmail;
14:     CString   m_sWeb;
15:     //}}AFX_DATA
16:
17:     bool m_bIsNew;
18:
19:  // Overrides
20:     // ClassWizard generated virtual function overrides
21:     //{{AFX_VIRTUAL(CEmpresaDlg)
22:     protected:
23:     virtual void DoDataExchange(CDataExchange* pDX);
24:     //}}AFX_VIRTUAL
25:
26:  // Implementation
27:  protected:
28:     bool Guardar();
29:     void LoadData();
30:
31:     // Generated message map functions
32:     //{{AFX_MSG(CEmpresaDlg)
33:     virtual BOOL OnInitDialog();
34:     virtual void OnOK();
35:     //}}AFX_MSG
36:     DECLARE_MESSAGE_MAP()
37:  };
```

Constructor de la clase `CEmpresaDlg`

El constructor recibe como primer parametro la variable `CString sRUC` la cual asignaremos a `m_sRUC`. Esto tambien nos servirá para inicializar `m_bIsNew`.

Adicionamente tenemos –tambien– que declarar la variable global `theApp`, esto para poder utilizar la conexión global a MySQL. Ver Listado 12 línea 1. Si esta declaracion provoca un error entonces también debe incluir el archivo TestMySQL.h : `#include "TestMySQL.h"`

Listado 12: Constructor de la clase `CEmpresaDlg`

```
 1:  extern CTestMySQLApp theApp;
 2:
 3:  CEmpresaDlg::CEmpresaDlg(CString sRUC,CWnd* pParent /*=NULL*/)
 4:     : CDialog(CEmpresaDlg::IDD, pParent)
 5:  {
 6:     //{{AFX_DATA_INIT(CEmpresaDlg)
 7:     m_sRUC = sRUC;
```

```
 8:     m_sRazonSocial = _T("");
 9:     m_sDireccion = _T("");
10:     m_sEmail = _T("");
11:     m_sWeb = _T("");
12:     //}}AFX_DATA_INIT
13:
14:     m_bIsNew = m_sRUC.IsEmpty() == TRUE;
15: }
```

La función `LoadData`

La función `LoadData` se parece mucho a la función `CTestMySQLDlg::LoadData` sólo que esta vez carga un solo registro, y se asigna el valor de cada campo a la variable que corresponde en el diálogo. Ver Listado 13. La consulta SQL que estamos ejecutando es `SELECT * FROM empresas WHERE ruc = m_sRUC`.

En la función `OnInitDialog` invocaremos a `LoadData` en caso de que estemos editando un registro. Ver Listado 13.

Listado 13: Funciónes `OnInitDialog` y `LoadData`

```
 1: BOOL CEmpresaDlg::OnInitDialog()
 2: {
 3:    CDialog::OnInitDialog();
 4:
 5:    if (!m_bIsNew)
 6:    {
 7:       LoadData();
 8:    }
 9:    return TRUE;
10: }
11:
12: void CEmpresaDlg::LoadData()
13: {
14:    CWaitCursor x;
15:    UpdateData();
16:    CString sql;
17:
18:    sql.Format("SELECT * FROM empresas WHERE ruc = '%s'",m_sRUC);
19:
20:    if(mysql_real_query(
21:       theApp.GetConnection()
22:       ,sql
23:       ,sql.GetLength()) == 0 )
24:    {
25:       MYSQL_RES* res;
26:
27:       if ((res = mysql_store_result(theApp.GetConnection())))
28:       {
```

```
29:
30:           MYSQL_ROW Row;
31:           if((Row = mysql_fetch_row(res)))
32:           {
33:
34:              m_sRUC         = Row[0];
35:              m_sRazonSocial = Row[1];
36:              m_sDireccion   = Row[2];
37:              m_sEmail       = Row[3];
38:              m_sWeb         = Row[4];
39:              UpdateData(FALSE);
40:           }
41:           mysql_free_result(res);
42:        }
43:     }
44:     else
45:     {
46:        CString strText;
47:
48:        strText.Format("Error: '%s'"
49:           ,mysql_error(theApp.GetConnection()));
50:
51:        AfxMessageBox(strText);
52:     }
53:  }
```

La función `Guardar`

En la función `Guardar()` ejecutará el comando SQL `INSERT` ó `UPDATE` según sea el caso. En la linea 19 del Listado 14, antes de ejecutar cualquier consulta, evaluamos el valor de la variable `m_bIsNew`. Si la consulta se ha ejecutado con éxito `Guardar()` devolverá `true`, y como es invocada desde `OnOK` el dialogo se cerrará devolviendo `IDOK`.

Listado 14: Funciónes `Guardar` y `OnOK`

```
 1:  void CEmpresaDlg::OnOK()
 2:  {
 3:     if (Guardar())
 4:     {
 5:        CDialog::OnOK();
 6:     }
 7:  }
 8:
 9:  bool CEmpresaDlg::Guardar()
10:  {
11:     CWaitCursor x;
12:
13:     UpdateData();
14:
```

```
15:     CString sql;
16:
17:     m_sRazonSocial.Replace("'","''");
18:
19:     if (m_bIsNew)  //Nueva Empresa
20:     {
21:        sql.Format("INSERT INTO empresas \
22:           VALUES ('%s','%s','%s','%s','%s')"
23:           ,m_sRUC
24:           ,m_sRazonSocial
25:           ,m_sDireccion
26:           ,m_sEmail
27:           ,m_sWeb);
28:     }
29:     else // Modificar Empresa
30:     {
31:        sql.Format("UPDATE empresas SET RazonSocial = '%s', \
32:           Direccion = '%s', Email = '%s', Web = '%s' \
33:           WHERE ruc = '%s' "
34:           ,m_sRazonSocial
35:           ,m_sDireccion
36:           ,m_sEmail
37:           ,m_sWeb
38:           ,m_sRUC);
39:     }
40:
41:     if(mysql_real_query(
42:        theApp.GetConnection()
43:        ,sql
44:        ,sql.GetLength()) == 0 )
45:     {
46:        return true;
47:     }
48:     else
49:     {
50:        CString strText;
51:
52:        strText.Format("Error: '%s'"
53:           ,mysql_error(theApp.GetConnection()));
54:
55:        AfxMessageBox(strText);
56:
57:        return false;
58:     }
59:  }
```

Ahora si: Guardar, Modificar, Eliminar

Luego de haber hecho el trabajo sucio de hacer que el diálogo `CEmpresaDlg` funcione, lo unico que nos queda es agregar las funciones correspondientes a los

nuevos botones de la Imagen 8. El Listado 15 muestra las tres funciones.

En el caso de `OnModify` y `OnDel` es necesario comprobar que un elemento del control `CListCtrl` esté seleccionado. Es importante que este declarada la inclusión del archivo EmpresaDlg.h (`#include ."EmpresaDlg.h"`) en el archivo `TestMySQLDlg.cpp` para poder utilizar la clase `CEmpresaDlg`.

Listado 15: Código para guardar, modificar y eliminar

```
 1:  void CTestMySQLDlg::OnNew()
 2:  {
 3:     CEmpresaDlg dlg("");
 4:     if (dlg.DoModal() == IDOK)
 5:     {
 6:        m_report.InsertItem(0,dlg.m_sRUC);
 7:        m_report.SetItemText(0,1,dlg.m_sRazonSocial);
 8:     }
 9:  }
10:
11:  void CTestMySQLDlg::OnModify()
12:  {
13:     POSITION pos = m_report.GetFirstSelectedItemPosition();
14:
15:     if (pos)
16:     {
17:        int item = m_report.GetNextSelectedItem(pos);
18:        CString ruc = m_report.GetItemText(item,0);
19:
20:        CEmpresaDlg dlg(ruc);
21:        if (dlg.DoModal() == IDOK)
22:        {
23:           m_report.SetItemText(item,0,dlg.m_sRUC);
24:           m_report.SetItemText(item,1,dlg.m_sRazonSocial);
25:        }
26:     }
27:     else
28:     {
29:        MessageBox("Debe seleccionar un item de la lista");
30:     }
31:  }
32:
33:  void CTestMySQLDlg::OnDel()
34:  {
35:     POSITION pos = m_report.GetFirstSelectedItemPosition();
36:     if (pos)
37:     {
38:        int item = m_report.GetNextSelectedItem(pos);
39:        CString ruc = m_report.GetItemText(item,0);
40:        CString msg = "Realmente desea Eliminar el item seleccionado?";
41:
42:        if (MessageBox(msg,"Eliminar",MB_YESNO|MB_ICONWARNING) == IDYES)
```

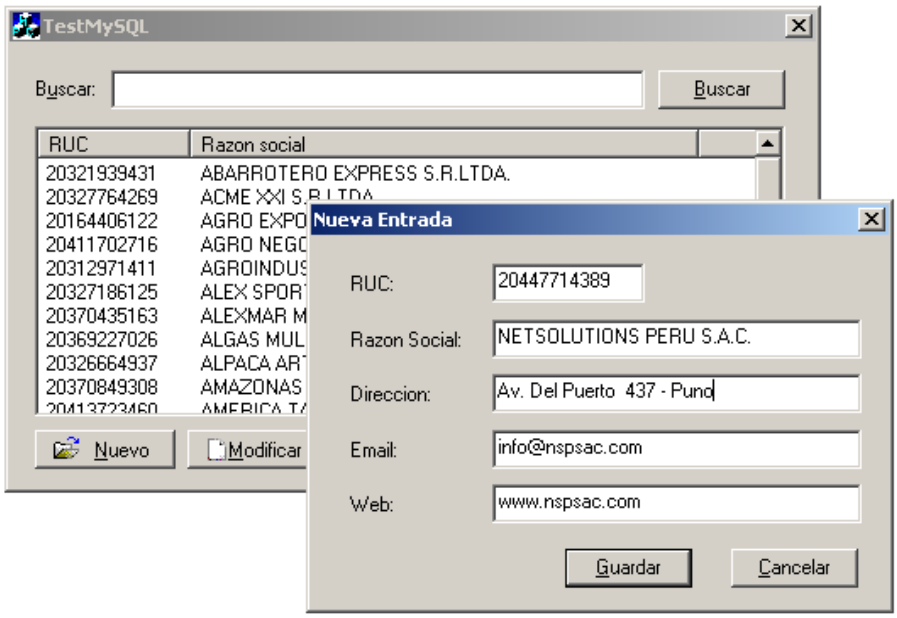
```
43:        {
44:           CString sql;
45:
46:           sql.Format("DELETE FROM empresas WHERE ruc = '%s'",ruc);
47:
48:           if(mysql_real_query(
49:              theApp.GetConnection()
50:              ,sql
51:              ,sql.GetLength()) == 0 )
52:           {
53:              m_report.DeleteItem(item);
54:           }
55:           else
56:           {
57:              CString strText;
58:
59:              strText.Format("Error: '%s'"
60:                 ,mysql_error(theApp.GetConnection()));
61:
62:              MessageBox(strText);
63:           }
64:        }
65:
66:     }
67:     else
68:     {
69:        MessageBox("Debe seleccionar un item de la lista");
70:     }
71:  }
```

Finalmente la aplicación correrá como se ve en la Imagen 10.

Notarán que los botones tienen un icono, es por que he utilizado la clase `CImageButton` del artículo que esta ubicado en:

`www.latindevelopers.com/articles/vc/imagebutton/`

He modificado la clase `CImageButton` para que acepte solo una imagen por boton. Ademas tambien le he agregado un temporizador para hacer la busqueda mas interactiva.



Imágen 10. Aplicacion Final corriendo

8. Comentarios, conclusiones, recursos , etc..

- Éste tutorial fue elaborado inicialmente en formato Microsoft Word, luego fue corregido y aumentado y re-elaborado en LaTeX.
- El proyecto Visual C++ TestMySQL y el archivo `prueba_db.sql` puede descargarlo de www.latindevelopers.com/vcpp/db/mysql_api/.
  El archivo `prueba_db.sql` contiene al rededor de 10,000 registros para que usted pueda experimentar. La dirección podría cambiar en los meses posteriores a la publicación de éste artículo.
- Sería saludable que usted me envíe un correo electrónico para resolver sus consultas y sugerencias o simplemente para saber como utiliza el material del atrículo.
- Algunas de las funciones de la librería API de MySQL posiblemente cambien segun vaya desarrollandose el gestor de base de datos MySQL. Se recomienda al lector que esté al tanto de las actualizaciones disponibles en el sitio oficial de MySQL: www.mysql.com Generalmente las actualizaciones no son muy dramaticas por lo que si está planeando implementar una aplicacion mas grande, es muy posible que sea compatible con versiones superiores de MySQL.
- En C/C++ es posible reutilizar codigo eficientemente, es lo que **no** hago en este tutorial. Las innumerables lineas de codigo fuente son simplificables.

Por citar un solo ejemplo: las consultas pueden agruparse en una función generica que las ejecute.

- Agradezco a Rosario Bustamante por realizar las correcciones ortográficas.
- El Lenguaje de programación Visual C++ es una marca y producto de Microsoft Corp. El gestor de base de datos MySQL es producido y distribuido bajo licencia GPL por MySQL AB.