Programación en C++ con

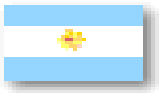
Qt bajo Entorno GNU/Linux

Martín Sande

sande.martin@gmail.com
cloud@argentina.com

http://www.gluca.com.ar

Linux User #281622



INDICE

INTRODUCCION

Se preguntaron alguna vez las diferencias entre: ¿Software Libre ó Software Propietario?

En la Informática actual las licencias de Uso nos dan la posibilidad de utilizar distintos Software (ej. Microsoft Windows, Microsoft Office, etc.) Pero no la posibilidad de cambiar el mismo adaptándolo a nuestros gustos ni la posibilidad de corregir posibles errores que los mismos contengan, nunca podremos ver el código fuente y cualquier modificación que queramos hacer será imposible, traslademos esto a nuestra vida cotidiana, imagínense que nos dieran la receta de una torta, con todos los ingredientes y la forma de hacerlo, pero nos la dieran con una "licencia de uso", nosotros la podremos usar, pero no escribir en ESA MISMA RECETA por ejemplo que agregamos otro ingrediente el cual no estaba en la receta, o cambiamos el tiempo de cocción porque nuestra cocina no alcanza esa temperatura, UNICAMENTE PODREMOS USARLA, NO MODIFICARLA, tampoco PODREMOS DISTRIBUIRLA ya no que poseemos los derechos de copyright y como nos la dieron la debemos mantener, por otro lado el Software Libre da la libertad de escribir programas bajo la licencia GPL y cobrar por este trabajo (en ningún momento se dice que deben ser gratuitos) compartir el software y modificarlo, de distribuirlo y cobrar por realizar esta tarea, da la libertad al usuario de acceder al código fuente, y de hacer las modificaciones que sean necesarias, que para el ejemplo anterior seria poder hacer nuestros cambios a la misma y distribuirla a todas las personas que quisiéramos .

Esto es Software Libre, es la libertad de saber QUE estamos usando y COMO funcionan los programas que estamos usando, no ser presos de un software ni de una compañía (se preguntaron alguna vez ¿Que software usarían si no existiese mas MS Word o si empezaran a cobrar por cada documento creado o abierto debitándolo de nuestra cuenta bancaria, SL es garantizar que la información generada es nuestra y la podremos abrir o guardar como queramos (saben como guarda Word un .doc ó Excel un .xls) con cualquier programa que queramos (por ejemplo OpenOffice.org.)

Este manual como habrán leído apunta a usar Qt para la creación de aplicaciones en GNU/Linux, es sabido los problemas de licencia que hubo anteriormente con KDE, pero Trolltech libero las librerías Qt bajo la licencia GPL y QPL (Qt Public Licence) hace tiempo para realizar aplicaciones en GNU/Linux las cuales deben ser GPL ú Open Source, no pudiendo desarrollar Aplicaciones de Software Propietario.

Los dejo seguir leyendo este manual que no creo que contenga la verdad absoluta (asi como tambien puede llegar a tener errores/falta de profundidad en algunos temas), solo las herramientas necesarias para introducirse en la programacion en el mundo GNU/Linux.

Los dejo en compañía del manual, espero que lo disfruten tanto como yo.

Martín Sande

Creditos:
Diagramacion, Edicion y Diseño Grafico: Martin Sande
Revision Ortografica y Gramatical y 2da Revisión de Diseño:
Mariana Folik marian925@hotmail.com

INTRODUCCION A QT DESIGNER

Qt Designer es una aplicación para crear la interfaz de usuario de nuestros programas usando las librerías qt, su forma de usarlo es fácil, en pocos minutos podremos crear nuestros elementos de la interfaz, asignarle los nombres y crear los eventos y las funciones que queremos que realice, para luego codificarlos usando un Lenguaje de POO como es C++

Para ejecutar el programa, en nuestra línea de comando ejecutamos el comando *designer-qt3*

En nuestro caso utilizamos la versión 3.1.2, con lo cual se debe tener en cuenta que cuando compilemos nuestro programa y lo queramos distribuir los requerimientos para las demás personas que quieran ejecutarlo es la de tener KDE 3.1.2 o contar con las siguientes librerías:

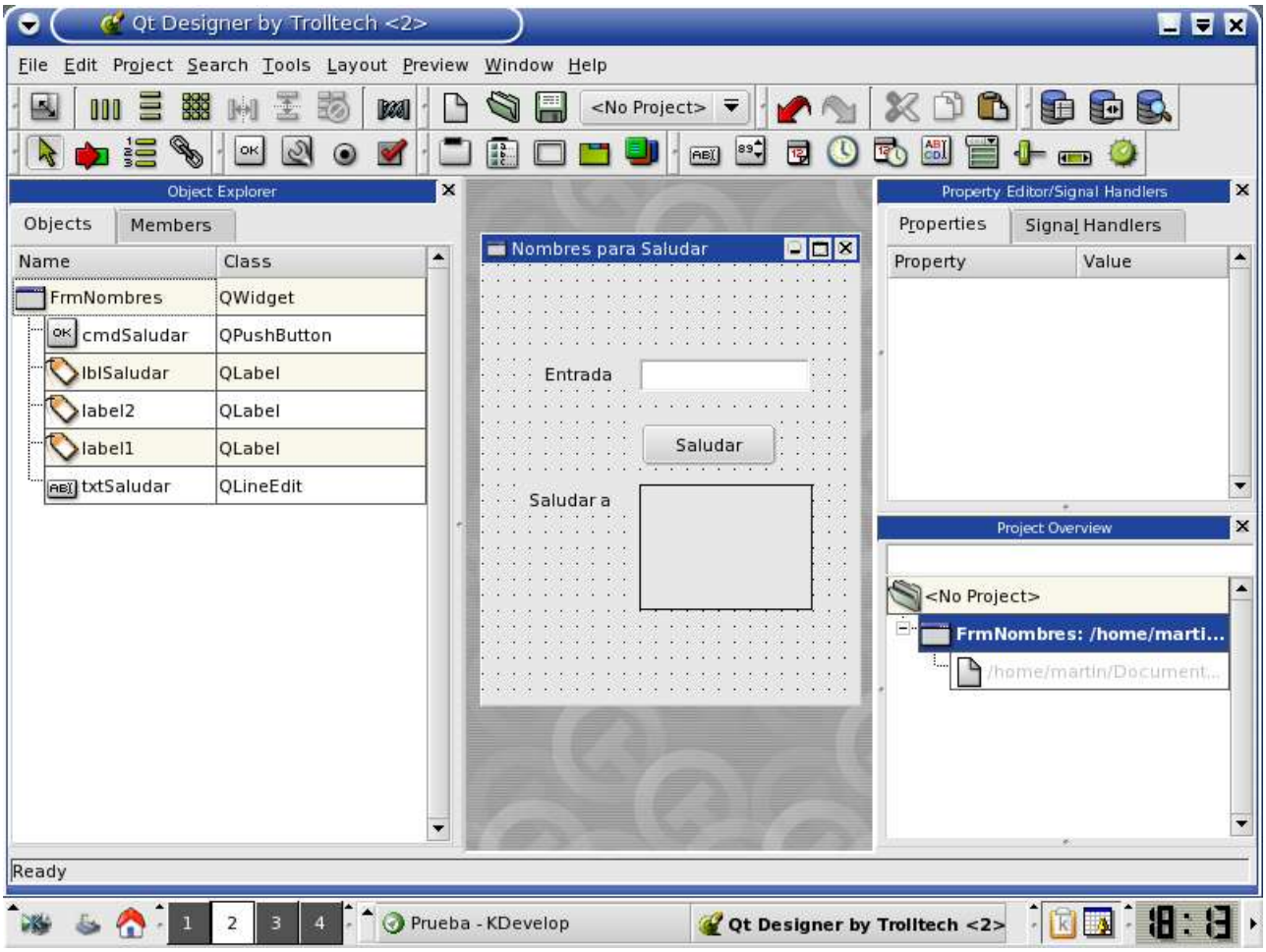
- qt-designer.3.1.2
- qt3.1.2
- qt-devel.3.1.2
- qtlibs.3.1.2
- qtlibs-devel3.1.2

Nuestro ejemplo se basará en la creación de un formulario con el cual ingresaremos nuestro nombre, y al presionar un botón nos aparecerá en una etiqueta el nombre que ingresamos, para ello debemos dividir nuestra aplicación en dos partes:

Creación de la interfaz de Usuario (o GUI) en qt Designer.
Codificación de los procedimientos y funciones en kdevelop.

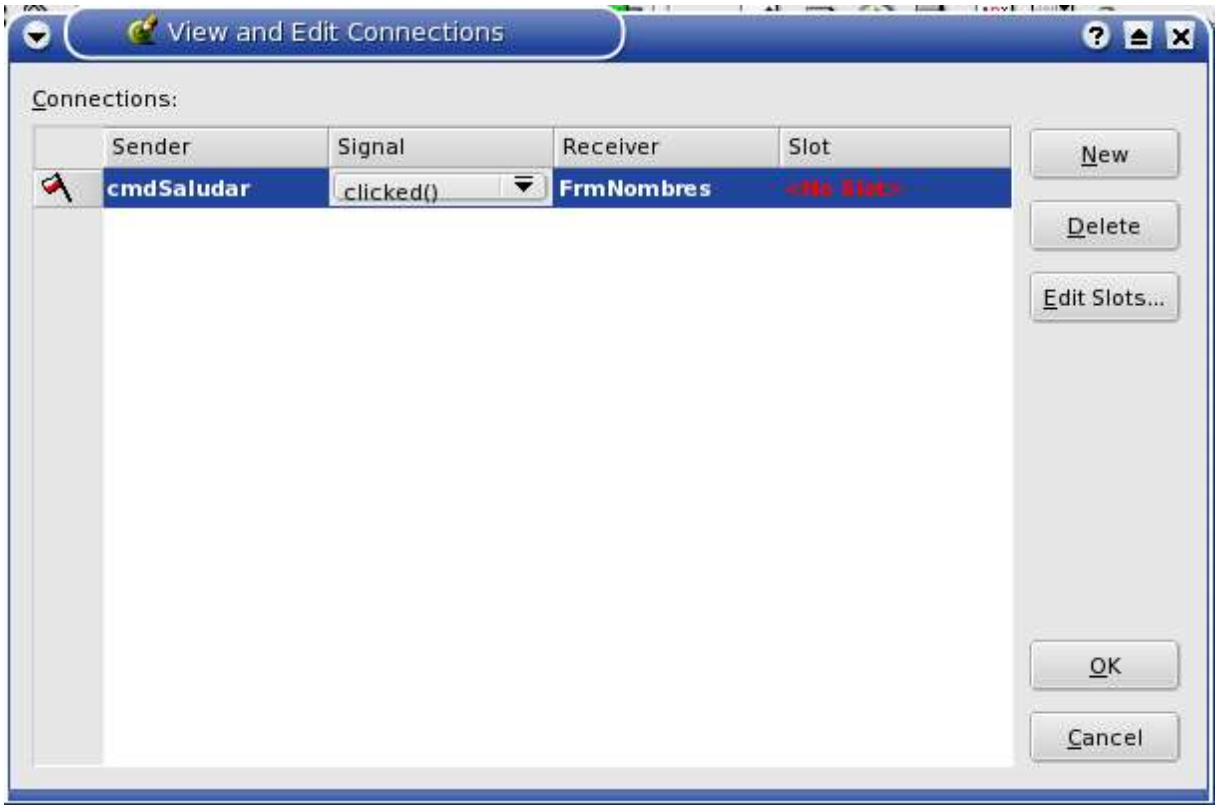
A continuación comenzaremos con la creación de la interfaz en qt.

Llamar al formulario por Frm... (lo que sea con la primera letra en mayús.)
por Ejemplo FrmPersonas.
En este ejemplo se crea un Formulario del tipo “Widget” (la clase base de toda la interfaz gráfica en Linux) como el que aparece a continuación con el siguiente detalle:

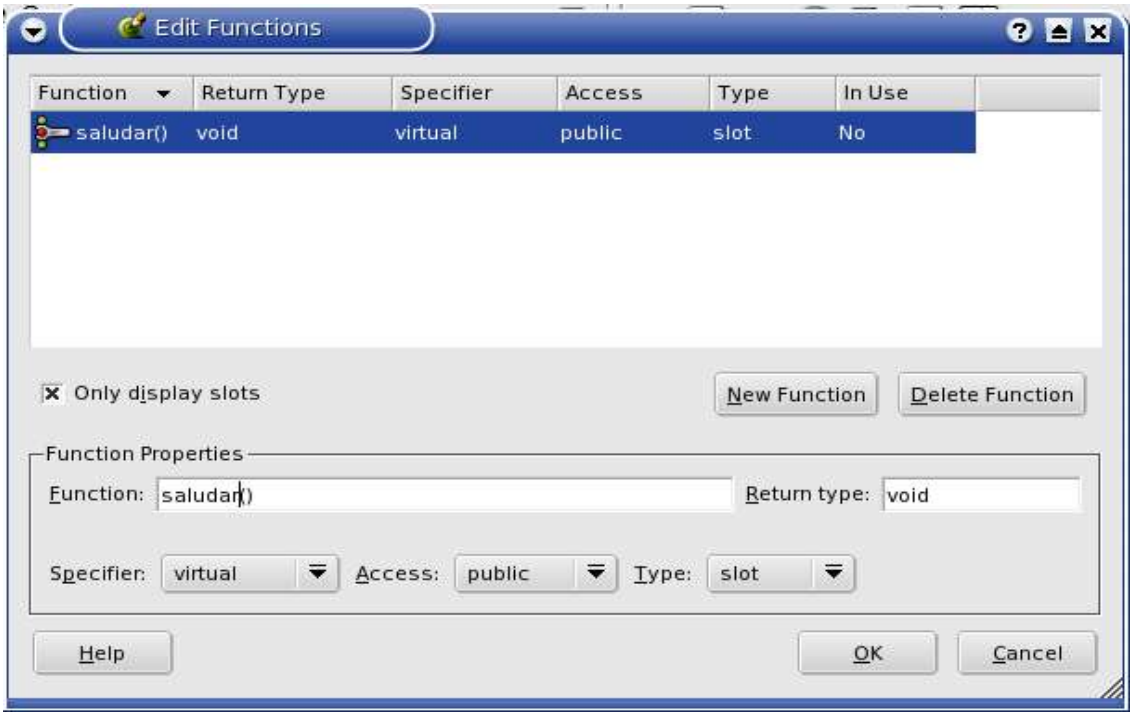


- Una Caja de Texto llamada txtSaludar
- Una Etiqueta con el borde en “box” llamada lblSaludar
- Un botón de comando llamado cmdSaludar
- Dos etiquetas que como no tienen función se llaman label1 y label2.
- El Nombre del Formulario es FrmNombres (este dato es muy importante ya que desde el punto de vista de C++ este es el nombre de la *clase* que manejaremos en Kdevelop) y como Caption la leyenda “Nombres para Saludar”.

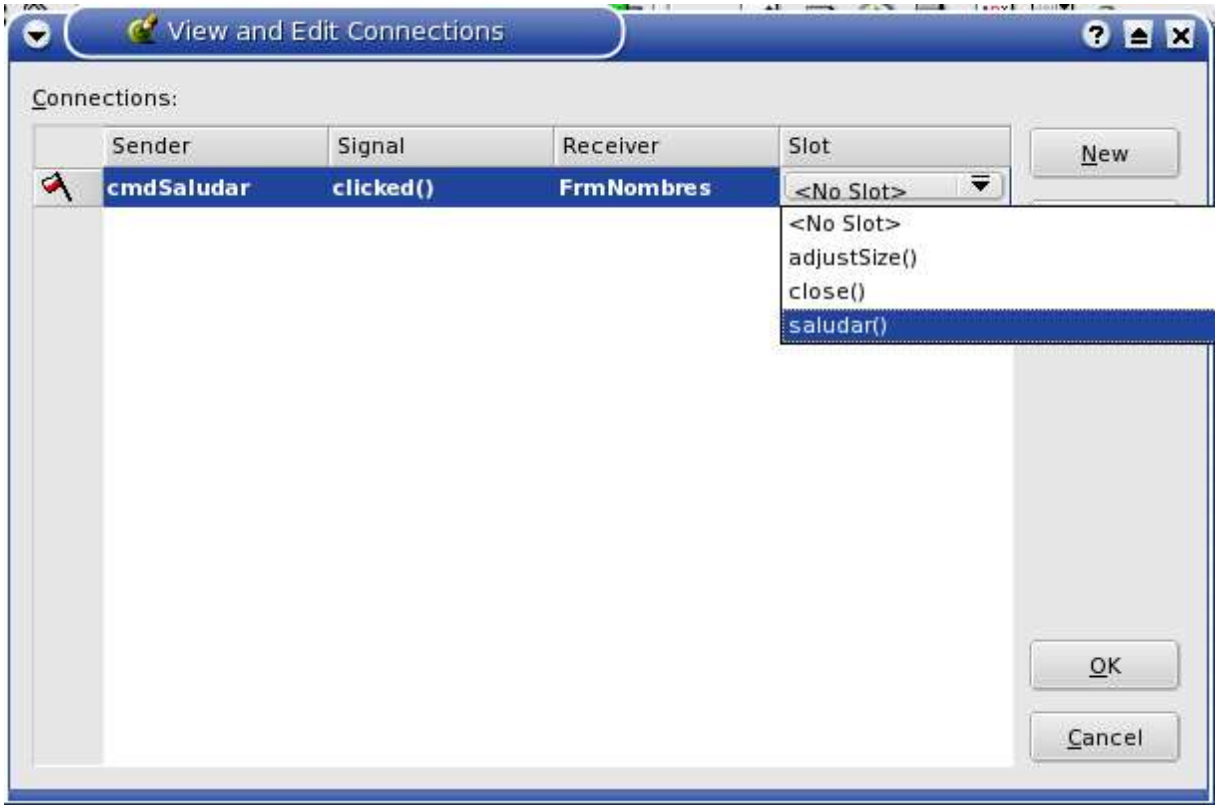
Crear las conexiones con los botones usando la signal tool de qt tal como se ve en la siguiente pantalla, una vez que colocamos los datos que aparecen en la pantalla a continuación, hacemos click en el botón “Edit Slots...”



Hacemos click en el botón New y creamos una función llamada *saludar()* como aparece en la siguiente pantalla:

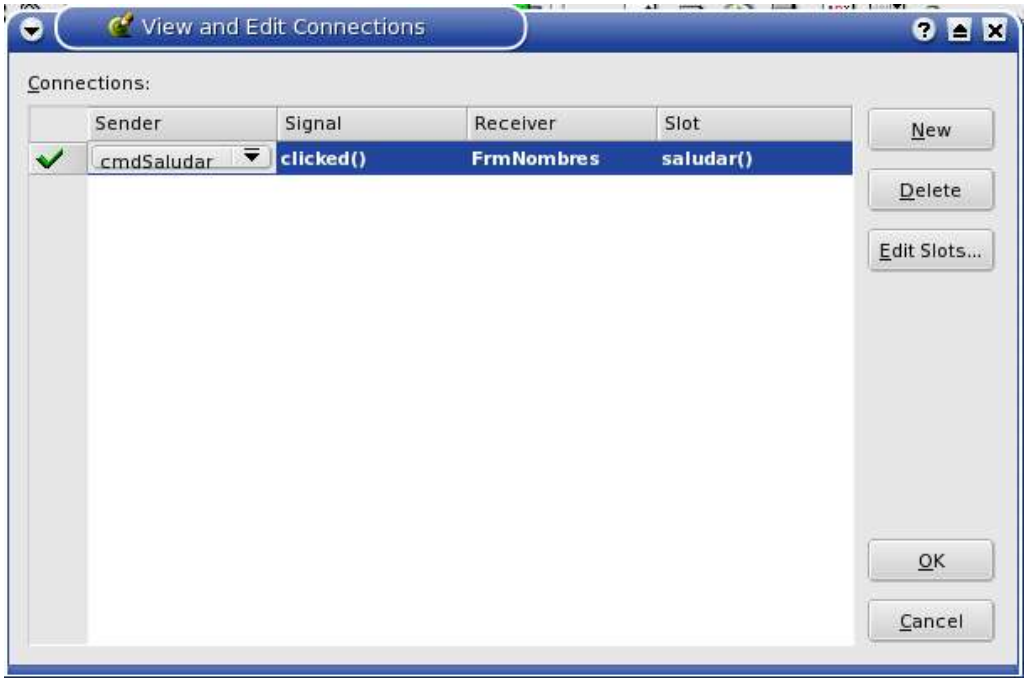


Presionamos el botón OK y le asignamos la función a nuestro slot creado con anterioridad, como aparece en la siguiente pantalla:



Elegimos la opción que aparece nueva que es *saludar()*

A las conexiones decirles que apunten a funciones que nosotros mismos creamos haciendo Conections Slots >>Edit Slots >>New Function>>Creamos la función y se la asignamos a la conexión del combo en la columna slot (antes colocamos en la conexión el sender, signal y receiver), si hicimos todo esto debería quedar de esta manera:



Guardamos el archivo hecho en Qt con el mismo nombre que el formulario pero con el nombre en minúsculas.

Abrimos una terminal, nos paramos en el directorio donde guardamos el Archivo con extensión *.ui que también debe haber uno con extensión *.ui.h y ejecutamos los siguientes comandos:

[martin@localhost bin]$ uic -o nombre_del_formulario.h nombre_del_formulario.ui

[martin@localhost bin]$ uic -o nombre_del_formulario.cpp -i nombre_del_formulario.h nombre_del_formulario.ui

Si bash no les reconoce el comando **uic**, entonces tienen que hacer lo siguiente:

Entran como root y ejecutan las siguientes líneas:

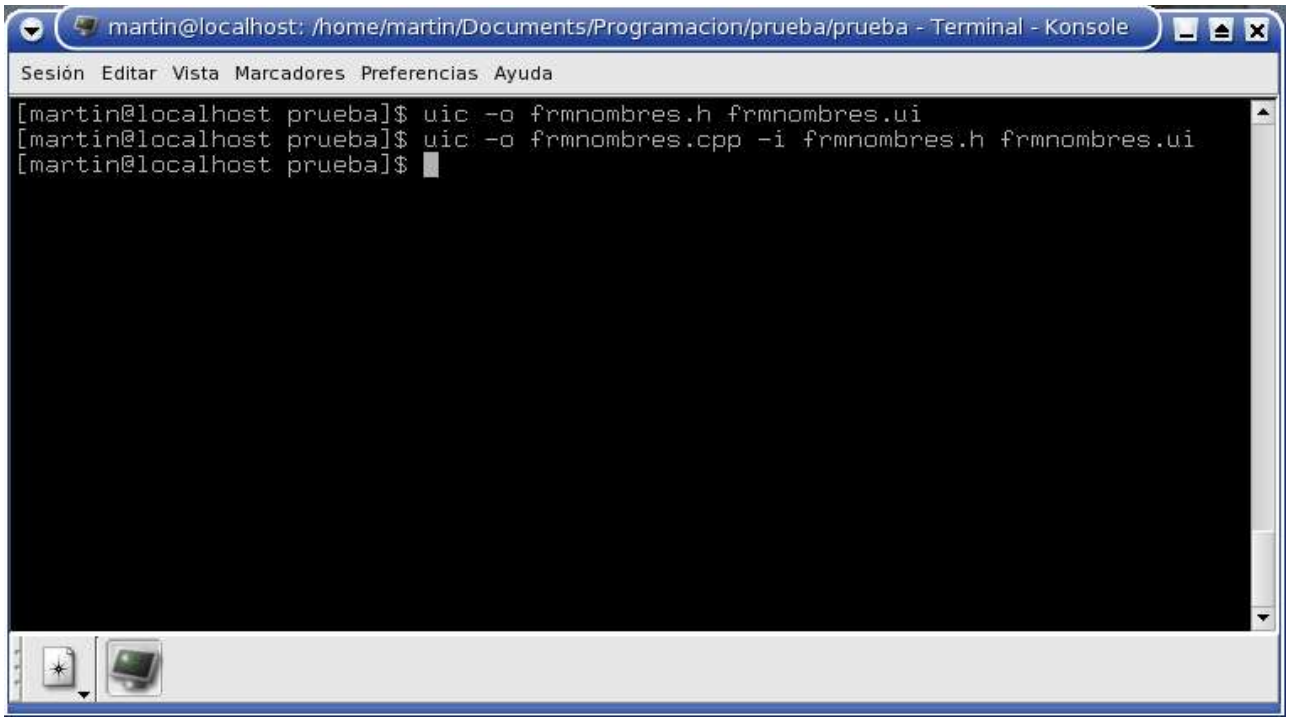
[root@localhost]# vi /root/.bashrc

Agregar PATH=/usr/lib/qt3/bin (si esta creada *:/usr/lib/qt3/bin)*
Grabar los cambios (":w) y salir grabando (:wq)

Y prueban de nuevo.

Con lo que veremos que generamos los archivos *.cpp y *.h del archivo *.ui

Ejemplo del paso anterior:



Con esto logramos compilar la interfaz gráfica (el archivo creado con qt designer).
Si quieren abrir el archivo con un editor de texto (el *.ui) notarán que no esta escrito en C++ sino que esta escrito en lenguaje XML, **uic** (**U**ser **I**nterface **C**ompiler) decodifica la información que esta en formato XML y la pasa a C++

Cabe destacar que no se han tocado todas las funcionalidades de qt designer, mas adelante en este manual se tocaran las funciones básicas para un mejor aprovechamiento de sus funciones (por ejemplo redimensionad automático de los objetos de un formulario cuando el mismo cambia de tamaño, creación de conexiones y slots, que es la forma en la que los componentes de la interface se comunican entre si)
Qt trae consigo una amplia bibliografía indicando todas las propiedades y funciones de sus objetos, al final del manual, en el apéndice A encontraran las propiedades mas usadas de los objetos utilizados en este manual.

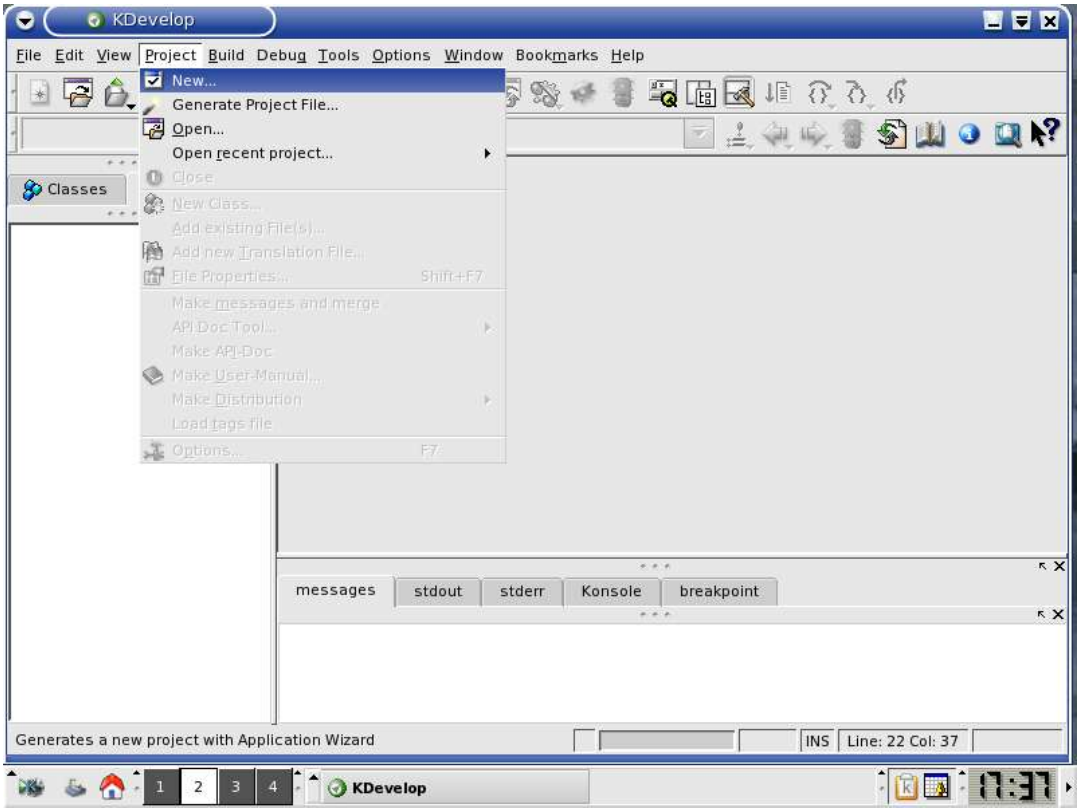
A continuación procederemos a la codificación de los eventos y a la compilación de nuestros programas, usaremos el entorno de programación Kdevelop, incluido en la distribución de KDE 3.1.2

INTRODUCCION A KDEVELOP

Kdevelop es un entorno de programación hecho en Linux para crear aplicaciones que corran en KDE, lo que no quiere decir que no pueden correr en Gnome, pero Gnome usa las librerías GTK+ y KDE usa Qt, lo que hace que las aplicaciones de Qt en KDE anden mas rápido por no tener que cargar las librerías gráficas.

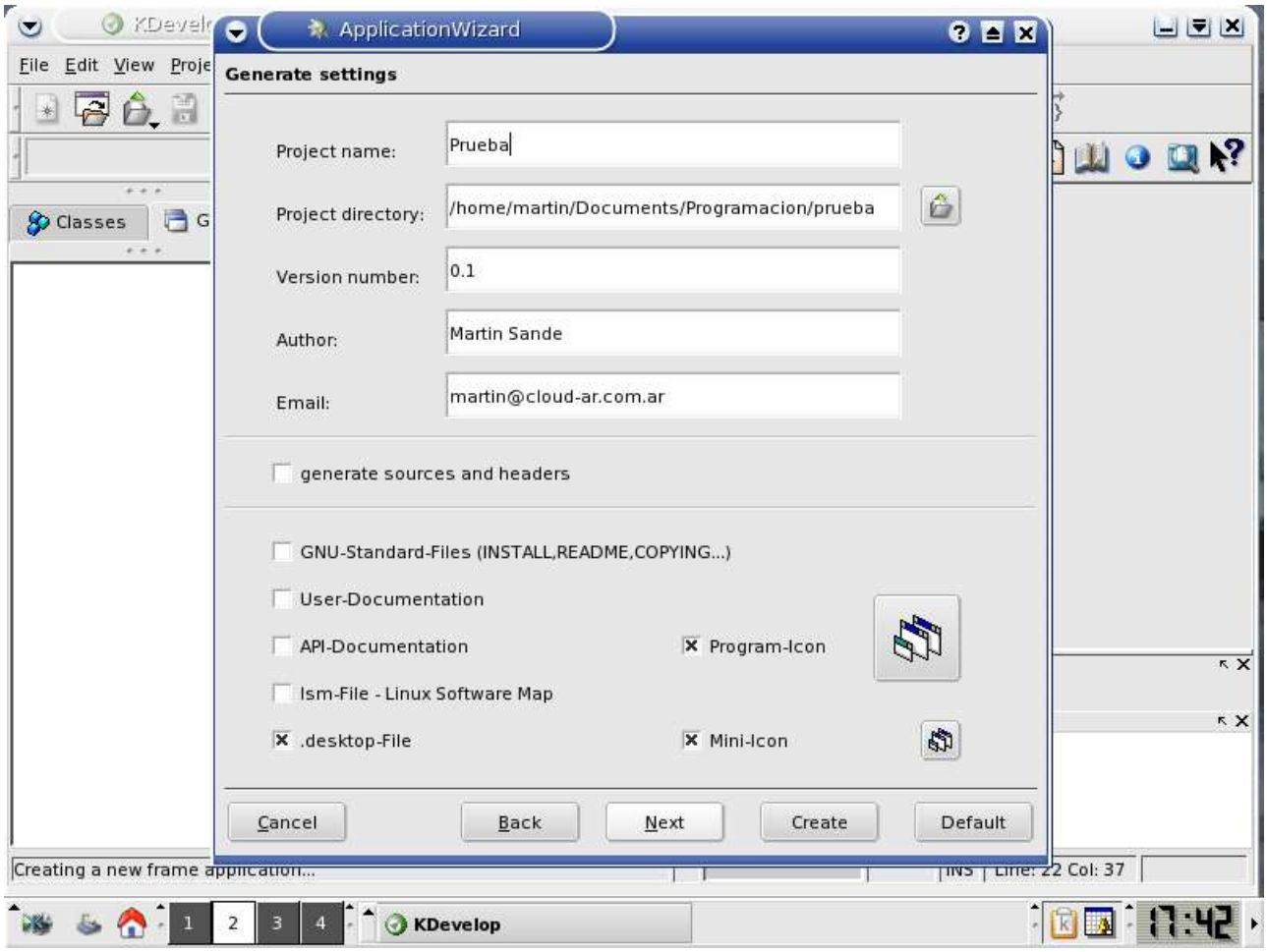
Abrimos Kdevelop (en mi caso la versión 2.1)

Vamos al menú Project -> New



Elegimos la opción ***Kde Normal*** (en este punto también podríamos elegir la opción de ***QT SDI*** ya que como se verá a continuación la interfaz gráfica la generaremos nosotros mismos, en caso de no querer hacerla nosotros mismos nos generara dependiendo la opción la interfaz gráfica que no podremos cambiar con las herramientas de este manual (Qt Designer).
En otro momento utilizaremos también la opción QT SDI ó QT MDI para observar que las opciones son las mismas (ya que tanto los archivos *.cpp , *.h y la interfaz la generaremos íntegramente nosotros..

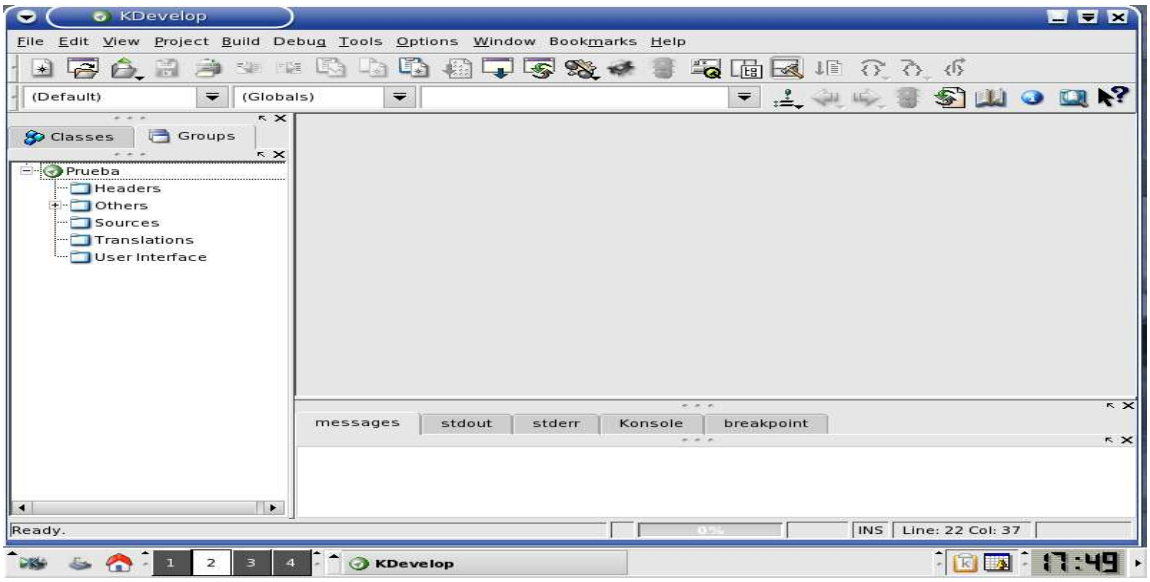
Y completamos los datos del formulario con el siguiente detalle:



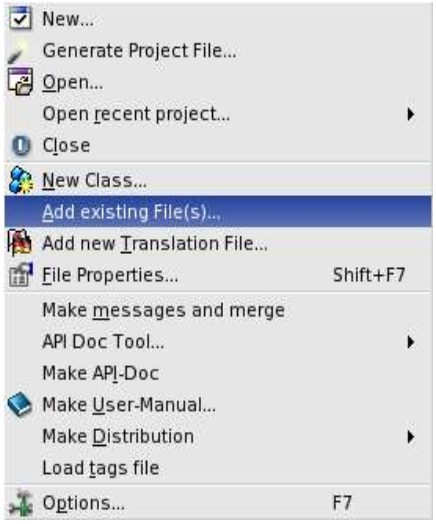
En VCS Support no lo seleccionamos (si ustedes quieren usarlos actívenlo y lean la documentación de VCS)

Lo mismo si quieren que Kdevelop le cree automáticamente los header de los archivos con la leyenda GNU y los datos del creador.

Elegimos la opción “Create” para que Kdevelop cree el entorno de programación, una vez finalizado este proceso les aparecerá la siguiente ventana



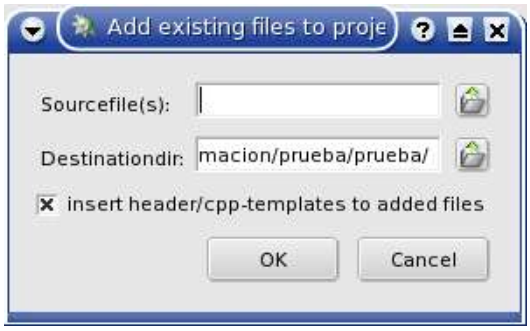
e) En el Menú “Project” Elegimos la opción “Add existing File(s)...”



Y buscamos los archivos a incluir que son los siguientes:

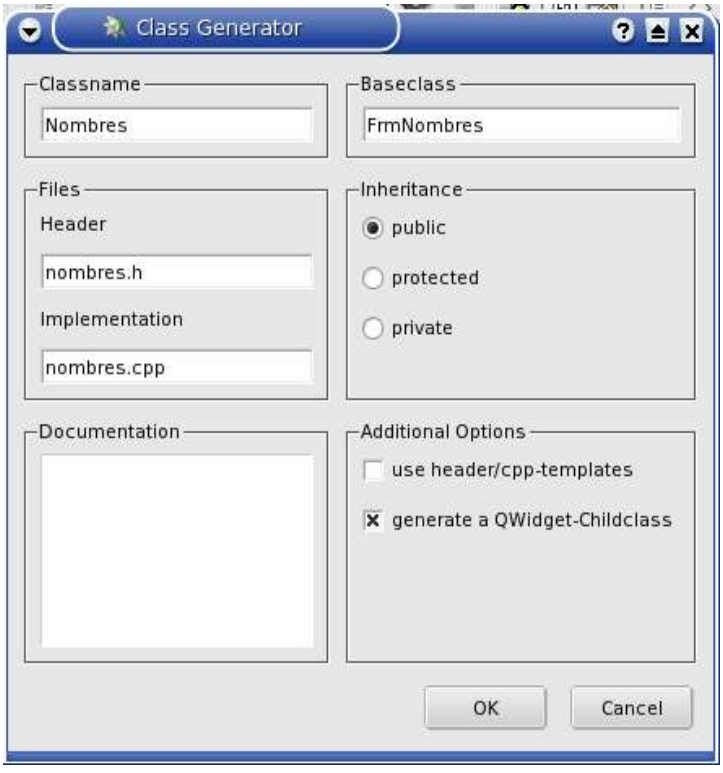
- El archivo con extensión .ui
- El archivo con extensión .h
- El archivo con extensión .cpp

Y guardamos los mismos en el mismo directorio de nuestro proyecto como muestra la siguiente figura:



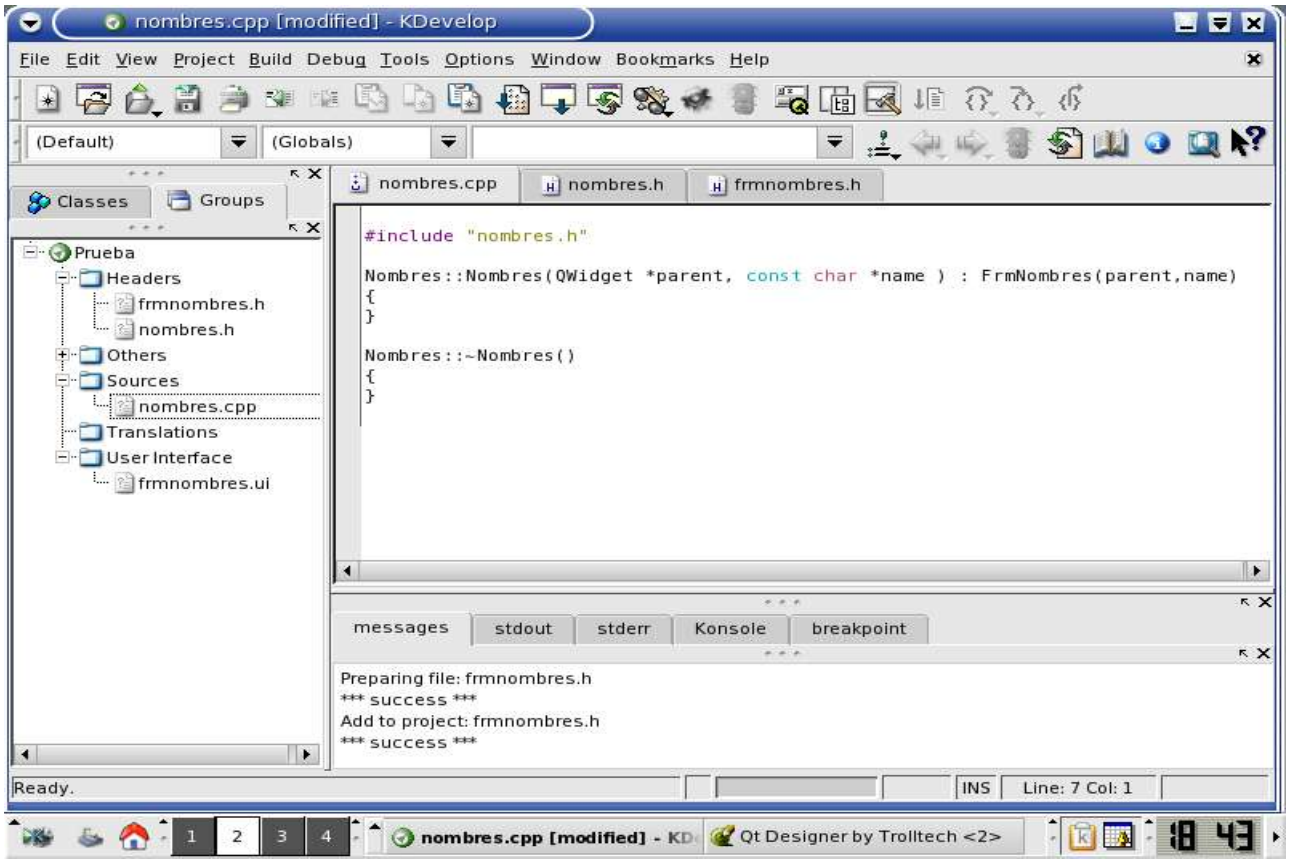
Una Vez realizado esto procedemos a la creación de los archivos para la clase que vamos a manejar, como manera de programar tomo que las clases que derivan de las interfaces se llaman igual que las interfaces sin el prefijo “Frm”, por ejemplo de FrmNombres, mi clase se llamara Nombres.

A continuación se ve como se crea una clase en Kdevelop, vamos al Menú Project -> New Class y aparece la siguiente pantalla, a llenar como aparece:



Recuerden respetar las minúsculas y mayúsculas porque linux las trata como dos nombres de archivos distintos, en esta pantalla básicamente crean la clase (Nombres) e identifican a partir de que clase se crea (FrmNombres que era el nombre del formulario, *no les dije que era importante este nombre)*, esto en C++ se llama **Herencia**

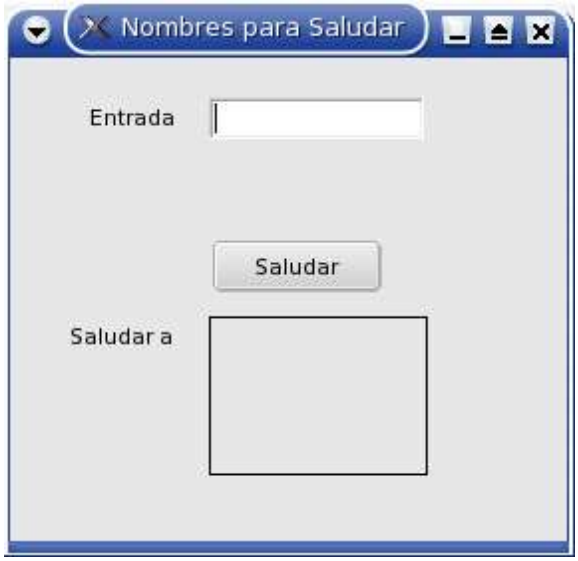
Una vez completados todos los datos le damos al botón OK con lo cual tendremos la siguiente situación en nuestro proyecto:



Vamos al Menú File -> New -> C/C++ File (.cpp) y le damos como nombre main.cpp
Dentro de ese archivo copiamos el siguiente código:

```
/* begin              : vie dic 19 2003
 copyright          : (C) 2003 by Martin Sande
 email              : cloud@argentina.com
*/

#include "nombres.h"          //nombre de la clase creada en Kdevelop
#include "frmnombres.h"       //nombre de la clase creada con uic (ídem form)
#include <qapplication.h>     //si o si base de toda aplicación de Qt

int main( int argc, char ** argv )  //procedimiento principal

{
   QApplication Main( argc, argv );              //Creo Aplicación de Qt
   Nombres Form;                                 //Creo form a partir de clase
   Form.show();                                  //Muestro form
   Form.setCaption("Nombres para Saludar");      //Asigno caption al form
   return Main.exec();
};
```

Compilamos y debería andar... es 100% seguro que anda si siguieron todos los pasos al pie de la letra, sino puede ser que en el main les falten los includes (ojo! A eso)



Programa Funcionando

PROGRAMACIÓN DE LOS EVENTOS DE LA INTERFAZ DE USUARIO

Que seria de cualquier interfaz si por mejor diagramación que tuviera no se pudiera interactuar, así que en este ejemplo tocaremos lo básico, el fin de este ejercicio es que una vez que hayamos introducido un nombre en la caja de texto, al hacer click en el botón saludar aparezca el nombre en la etiqueta (si, ya se que es básico pero dándose esta idea de como maneja C++ los objetos qt podrán hacer cualquier cosa, creanme...)

Lo primero que tenemos que hacer es escribir las siguientes líneas de código:

En nombres.h

Dentro de la clase Nombres debajo del apartado public:

```
public slots:
virtual void saludar();
```

En frmnombres.h

Dentro de la clase FrmNombres debajo del apartado public:

```
public slots:
virtual void saludar();
```

Las líneas de código para que se ubiquen deberían colocarlas después de la función de destrucción de la clase ~Nombres(); o ~FrmNombres(); , según el archivo en que se encuentren.

Una vez hecho esto en el archivo nombres.cpp agregar el siguiente código:

```
void Nombres::saludar()
{
lblSaludar->setText(txtSaludar->text());
}
```

Explicación del código

Lo que realizamos fue la asignación del texto de la caja de texto en la etiqueta, las etiquetas y cajas texto tienen dos propiedades, setText(), que sirve para asignar un nuevo texto al objeto eliminando el texto que contenía, y la propiedad text(), con la cual se lee el texto que contiene el objeto, ambas propiedades aceptan solo variables del tipo string (en qt se llama Qstring)

También se puede asignar texto mediante *lblNombres->setText("Martín");*

Recuerden que en la parte superior del archivo nombres.cpp deben estar puestos los siguientes archivos cabecera para que el programa funcione, que contiene todas las funciones de los objetos (de qt por ejemplo cajas de texto, botones, etc.)) y de las clases (por ejemplo Nombres y FrmNombres)

```
#include "nombres.h"        /Clase nuestra
#include "frmnombres.h"     //Clase base creada con qt
#include "qlineedit.h"      //textbox
#include "qlabel.h"         //label
#include "qpushbutton.h"    //botones
```

Dan Build -> Execute y su primer programa en Linux Funcionando!

AGREGANDO FUNCIONES A NUESTRO PROGRAMA

Imagínenme que los nombres que se ingresarían en la caja de texto serian siempre los mismos, para cumplir la función de la caja de texto se podría colocar un combobox en el formulario y cargarlo con nuestros nombres, lo que hay que hacer es lo siguiente:

Crear un combobox dentro de nuestra interface y darle como nombre cmbNombres.
Volver a compilar el archivo *.ui del formulario como hicimos en la pagina 5.
En Kdevelop elegir el archivo frmnombres.h, les avisará que cambio fuera del editor, si quieren que elimine la copia del buffer y lo vuelva a cargar del disco rígido, a lo que le contestan que si.

Cargando el Combo

Al igual que en Visual Basic deberemos ubicar nuestras líneas de código en el evento correcto para que cuando cargue nuestra aplicación en memoria y aparezca el formulario en pantalla ya este cargado el combo, en vb esas líneas de código van en el evento Form_load(), pero en C++ no. En C++ las clases (objetos) tienen dos funciones sin las cuales no pueden existir:

La función *Constructora* (que se llama siempre igual que la clase, ej. Nombres::Nombres() que es la encargada de asignar las variables y otros elementos en el momento que comienza el ciclo de vida de la clase.

La función *Destructora* que se caracteriza por tener antepuesto un signo ~, ej:
Nombres::~Nombres();
Cuya función aparece cuando termina el ciclo de vida de la clase, en qt no se coloca ninguna función ya que las mismas qt hacen el trabajo por nosotros.

Para nuestro ejemplo queremos que ya aparezcan cargados cuando aparezca el formulario por lo cual nuestro código ira dentro de la llamada de la función constructora, entonces, en el archivo *nombres.cpp* agregamos las siguientes líneas de código:

Entre los archivos de cabecera incluimos el del combobox para poder trabajar con las funciones que tiene programado en las qt:

```
#include "qcombobox.h"
```

Y la función constructora que debería quedar así:

```
Nombres::Nombres(QWidget *parent, const char *name ) : FrmNombres(parent,name)
{
cmbNombres->insertItem("Mariana");
cmbNombres->insertItem("Martin");
cmbNombres->insertItem("Matias");
cmbNombres->insertItem("Rosa");
cmbNombres->insertItem("Manuel");
}
```

Explicación del código

el objeto *qcombobox* tiene la propiedad *insertItem()* que sirve para insertar un nuevo elemento en la lista, si lo que quisiéramos hacer es asignar a otro objeto el texto seleccionado, el mismo se encuentra en la propiedad *text()*
Lo realizado en líneas anteriores fue la asignación de una variable del tipo Qstring dentro del combo.

Compilamos y ejecutamos , con lo cual nuestro programa quedaría de la siguiente manera:

Las acciones que soporta nuestro programa en estos momentos es la de escribir un nombre en la caja de texto, presionar el botón saludar y el nombre aparecerá en la etiqueta.
También se puede seleccionar un nombre del combobox cargado en la función constructora.

AGREGAR FUNCIONALIDADES EXTRAS AL FORMULARIO

Copiar el contenido del combo en la etiqueta y Capturar la tecla enter en la caja de texto y limpiarla

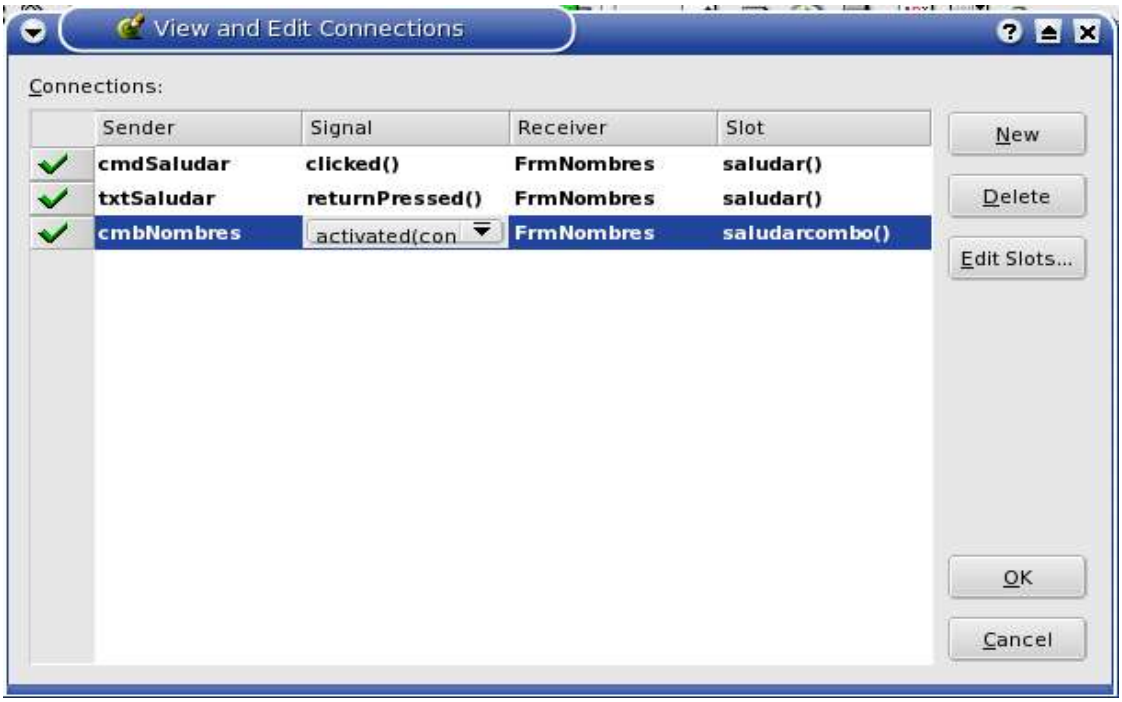
Imaginemos que queremos elegir un ítem del combo y que aparezca en la etiqueta, para lograr esto el objeto combobox trae entre sus eventos los siguientes:

activated (const. Qstring&) //devuelve el valor alfanumérico (texto) al hacer click
activated (int) //devuelve el valor numérico del ítem (índice) al hacer click
highlighted(const. Qstring&) //devuelve el valor alfanumérico (texto) al pasar por el combo
highlighted(int) //devuelve el valor numérico (índice) al pasar por el combo

Por otro lado, en las cajas de texto, llamadas Line Edit, tenemos los siguientes eventos:

lostFocus() //se dispara al perder el foco
returnPressed() //se dispara cuando se presiona la tecla Enter
selectionChanged() //se dispara cuando se cambia el contenido
textChanged(const Qstring&) //se dispara cuando se cambia el contenido letra a letra

En nuestro proyecto de Kdevelop abrimos nuestra interface (extensión *.ui) y en Qt Designer creamos los siguientes slots como se ve en la siguiente figura:



Como se ve en la siguiente figura, para el evento de apretar Enter en la caja de texto usamos nuestra función de *saludar()*, pero para el caso del combobox tenemos que crear una nueva función y asignarla a la conexión del combo, en este caso la llamamos *saludarcombo()*

Una vez que terminamos todo en Qt, volvemos a compilar la interface con el comando **uic** como se explico anteriormente, y volvemos a actualizar los archivos en kdevelop, una vez que realizamos estos pasos, nos disponemos a crear los slots públicos en los archivos *nombres.h* y *frmnombres.h*, para luego codificar el funcionamiento en el archivo *nombres.cpp*

Codificación de los eventos en nuestro proyecto de kdevelop

Pasamos al entorno Kdevelop, y hacemos las siguientes modificaciones en los archivos *nombres.h* y *nombres.cpp*

En *nombres.h*

En la declaración de la clase Nombres, en public slots : agregamos

```
virtual void saludarcombo();
```

En *nombres.cpp* codificamos la función con el siguiente código:

```
void Nombres::saludarcombo()
{
 lblSaludar->setText(cmbNombres->currentText());
}
```

Compilamos y ejecutamos, con lo cual en nuestro programa cuando elijamos un ítem del combo *cmbNombres* lo imprimirá en la etiqueta *lblSaludar*

Imagen de la aplicación:

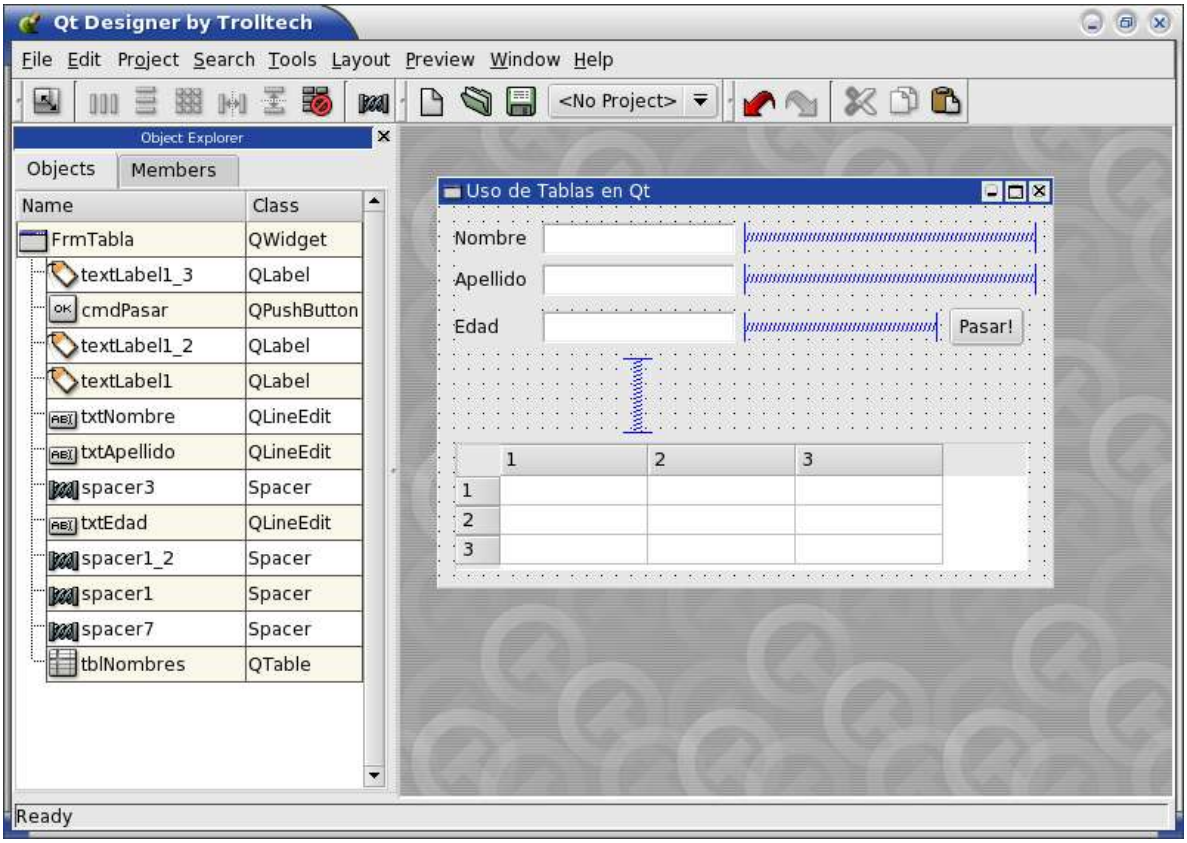
Realizado esto pasaremos a un nuevo ejemplo, el código completo del programa se encuentra en el apéndice B de este manual.

USO DE VARIOS FORMULARIOS EN NUESTRA APLICACIÓN

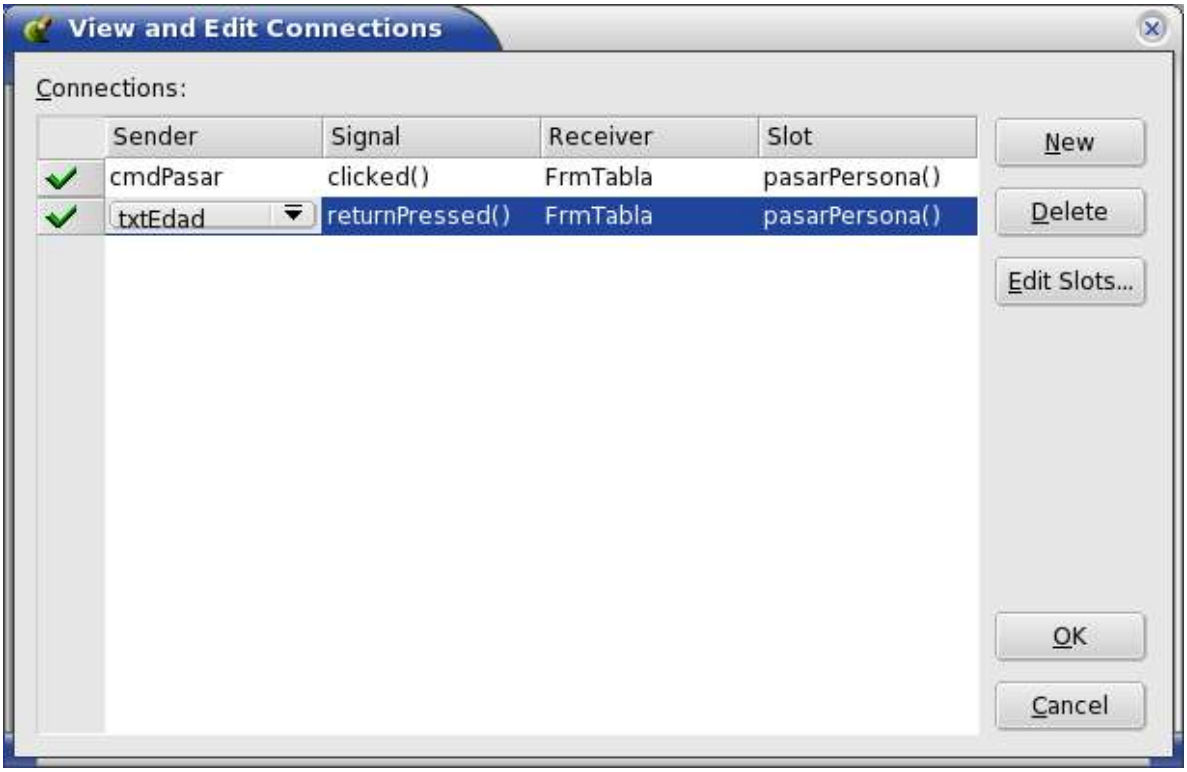
En el caso de que nuestra aplicación requiera el uso de más de una ventana, deberemos crear la interfaz en Qt Designer usando los mismos procedimientos usados anteriormente (creación de objetos, darle los nombres, crear las conexiones con sus respectivas funciones, etc.) compilar la interfaz con el comando **uic**, y agregarlo en Kdevelop.

En nuestro siguiente ejemplo empezaremos a utilizar las tablas y otros ejemplos mas de pasar datos a una etiqueta, solo que ahora tocaremos otra funcionalidad de las Qt que no vimos que es la *propiedad de cambiar nuestros objetos a medida que cambia el tamaño del formulario* (*resize*)
Para realizar esto en Qt tenemos los objetos Spacer, que sirven para decir el espacio que de debe respetar entre cada objeto, uno vez que fueron definidos se debe crear una grilla, que se logra dentro de Qt designer una vez ya colocados los spacers, se va al Menú Layout --> Lay Out in a Grid, con lo cual para nuestro ejemplo lograremos la siguiente interfaz:

En Kdevelop creamos un nuevo proyecto y los llamamos Tablas.



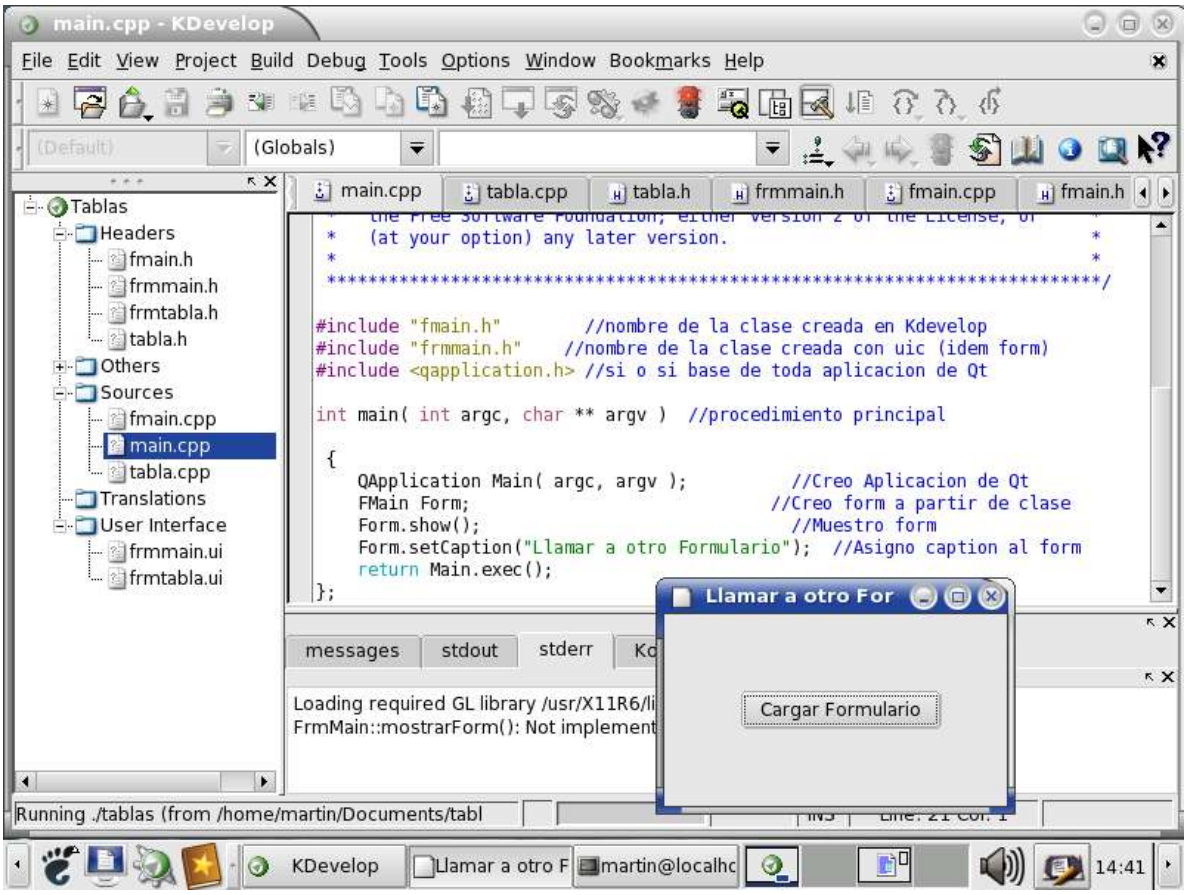
Una vez que tenemos la interfaz de nuestra aplicación procedemos a la creación de las conexiones que harán interactuar a nuestra aplicación con el usuario, las conexiones que crearemos son las siguientes:



Como se ve en la imágen usaremos la misma función para los dos eventos, el click del botón *cmdPasar* y apretar Enter en la caja de texto *txtEdad*

Guardamos el archivo de la interfaz y lo compilamos, volvemos a Kdevelop y agregamos los archivos y creámosla clase Tabla a partir de la clase FrmTabla, con lo cual tendríamos los siguientes archivos en nuestro proyecto:

Creamos otro archivo de interfaz que contenga un botón con el cual llamaremos al formulario creado anteriormente, para nuestro ejemplo el formulario se llamara FrmMain (frmmain.ui) y tendrá los archivos *frmmain.h* y *frmmain.cpp*, nosotros , en tanto, lo manejaremos creando una clase llamada FMain (por ser la palabra *main*() el primer procedimiento que se ejecuta cuando arranca nuestra aplicación y estar ya el archivo *main.cpp*, para no dar lugar a equivocaciones) el mismo contendrá un solo botón que llamaremos *cmdMain* que llamara a una función definida por nosotros llamada *mostrarForm(),* cuando el botón sea presionado (*clicked()*)
Copiamos nuestro archivo main.cpp para no tener que codificar nuevamente el archivo haciendo los cambios que se requieran para adaptarlo a nuestra nueva aplicación.
Guardamos todos nuestros cambios con lo cual nos encontramos con los siguientes archivos en Kdevelop y nuestra aplicación corriendo de esta forma:



Una vez que tenemos el proyecto como muestra la pantalla anterior, procedemos a codificar el procedimiento mostrarForm(); como ya saben tenemos que declarar el procedimiento público en la clase Fmain (fmain.h) y luego codificarlo, en nuestro archivo fmain.cpp agregamos el siguiente código:

En *frmmain.h*

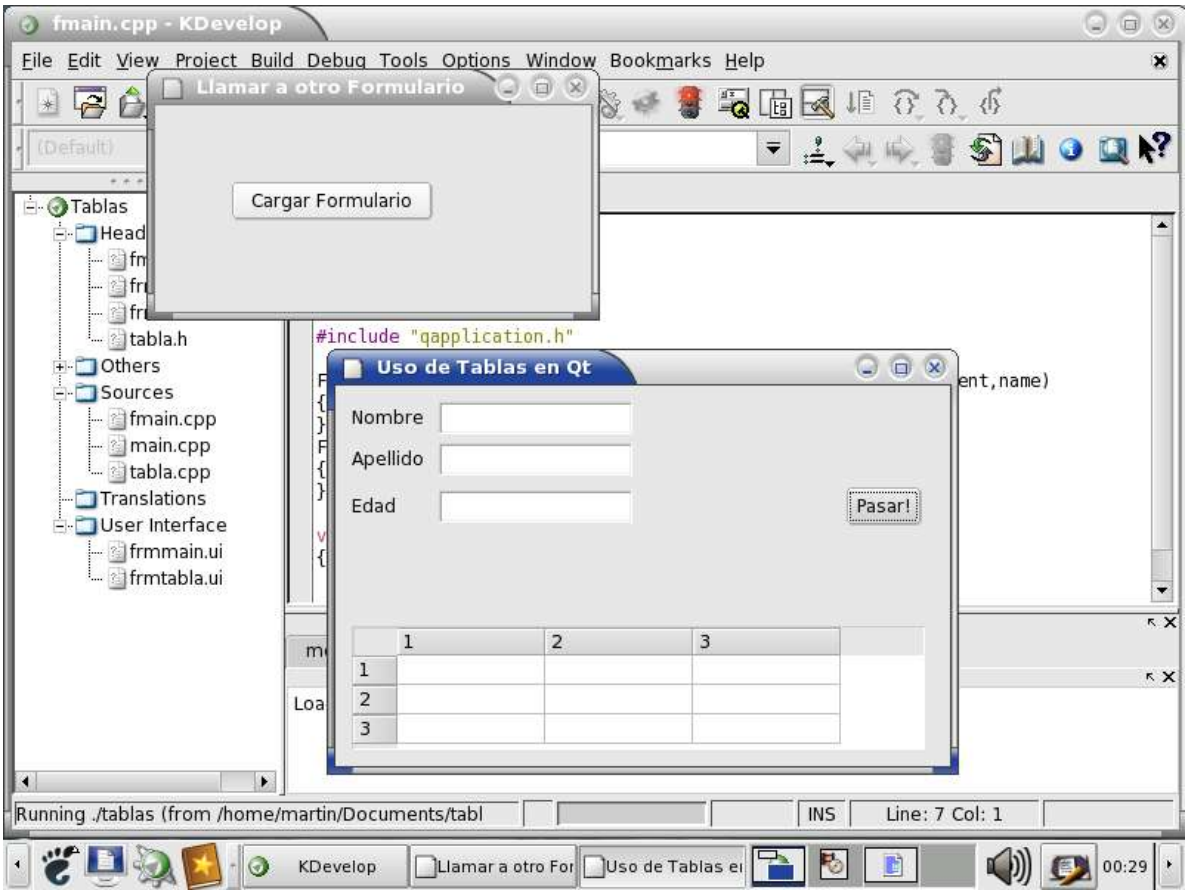
En la declaración de la clase FMain, en public slots : agregamos

```
virtual void mostrarForm();
```

Ahora viene lo importante, nosotros el evento de mostrar otro formulario lo codificaremos dentro del archivo fmain.cpp para en estos ejemplos tener todos los procedimientos de una clase en un mismo archivo, con lo cual en fmain.cpp agregamos las siguientes líneas de código:

```
void FMain::mostrarForm()
{
        Tabla *tabla = new Tabla ();
        tabla->show();
}
```

Compilamos y ejecutamos, logrando tener el programa funcionando de la siguiente manera:



Como se ve en la imágen superior, al presionar el botón Cargar Formulario se carga un nuevo formulario, en este caso Tabla con el caption en Uso de Tablas en Qt.

Ahora agregaremos funcionalidad a este formulario, la cual será la de cargar en la tabla los nombres, apellidos y edad de las personas que escribamos en las cajas de texto, siendo los disparadores para pasar a la tabla en evento *clicked()* del botón Pasar! (*cmdPasar*) y *returnpressed()* de la caja de texto Edad (*txtEdad*).

Antes de pasar a la codificación de los eventos, realizare una breve explicación de la **POO** (**P**rogramación **O**rientada a **O**bjetos), mas específicamente de las clases y del principio de encapsulación, ya que para escribir en la tabla necesitaremos crear una clase para que el dato de la fila en la cual deba escribir no puede estar declarada dentro de la función que escribirá ya que cada vez que termine la función se destruirá la variable, tampoco es aconsejable declarar variables publicas porque si ya que estaríamos violando la encapsulación (ya que una de las ventajas que ofrece es la de proteger los datos del exterior) por lo cual debemos crear una clase, y declarar las variables privadas (accesibles con el operador (.) punto) y las funciones publicas que manejen esos datos (accesibles con el operador de resolución de alcance (::) ó (->)
Claro, que a veces es mejor facilitar el código, por lo tanto nos limitaremos a crear una variable privada dentro de la clase Tabla, ya definida en el archivo tabla.h, en el cual dentro de la declaración de Tabla insertaremos las siguientes líneas de código (debajo de Qobject):

En *tabla.h* :

```
private:
        int fila;
```

La declaración de los títulos de las columnas las debemos declarar en la función constructora de la clase Tabla (Tabla::Tabla()), además, como la idea es cargar la tabla a partir de las cajas de texto, deshabilitaremos la propiedad de poder escribir en la tabla poniendo la propiedad readOnly() en TRUE.
A continuación de muestra el código a incluir en el archivo *tabla.cpp* :

```
Tabla::Tabla(QWidget *parent, const char *name ) : FrmTabla(parent, name)
{
fila=0;
QHeader *Titulos = tblNombres->horizontalHeader();
Titulos->setLabel( 0, ("Nombre"));
Titulos->setLabel( 1, ("Apellido"));
Titulos->setLabel( 2, ("Edad"));

Titulos->setMovingEnabled(TRUE);
tblNombres->setReadOnly(TRUE);
}

void Tabla::pasarPersona()
{
if ( fila >= tblNombres->numRows() )
{
     tblNombres->insertRows ( tblNombres->numRows() );
}
     tblNombres->setText ( fila , 0 , txtNombre->text() );
     tblNombres->setText ( fila , 1 , txtApellido->text() );
     tblNombres->setText ( fila , 2 , txtEdad->text() );

     txtNombre->clear();
     txtApellido->clear();
     txtEdad->clear();

fila++;
}
```

A continuación se incluye una imagen de nuestro programa en acción:

El código completo del programa se encuentra en el apéndice C de este manual.

CREACIÓN DE UNA APLICACIÓN MDI

En este apartado del manual procederemos a la creación de un programa con una interfaz **MDI** (**M**ultiple **D**ocument **I**nterface) en la cual reutilizaremos nuestros dos formularios programados en los ejemplos anteriores (Saludar y Tabla) iniciándonos en la programación de las opciones del menú del formulario MDI y la creación de barras de herramientas y los eventos mas importantes del formulario MDI.

Lo primero que debemos hacer en crear un nuevo proyecto en Kdevelop, para lo cual vamos al menú Project --> New

En esta ventana elegimos la opción QT -> Qt MDI y de la misma manera que en el primer ejemplo, desmarcamos las opciones de :

generate sources and headers
GNU Standart-Files (INSTALL,README,COPYING...)
User-Documentation
VCS Support
headertemplate for .h-files (opcional)
headertemplate for .cpp-files (opcional)

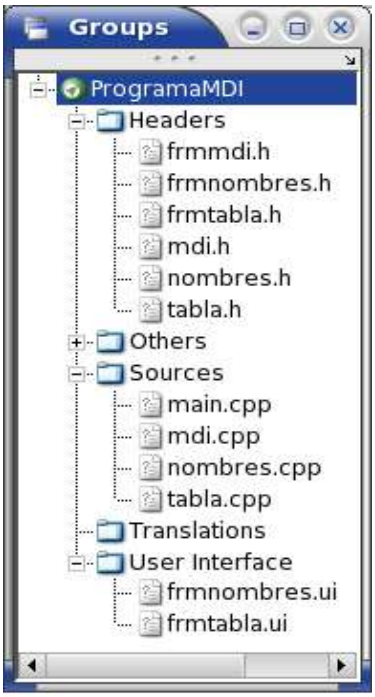
Y hacemos click en el botón *Create*.

Una vez terminada la creación del proyecto, copiamos en la carpeta donde se creo el proyecto los archivos correspondientes a los formularios Saludar y Tablas, los archivos a copiar son:

frmnombres.ui
frmnombres.cpp
frmnombres.h
nombres.cpp
nombres.h

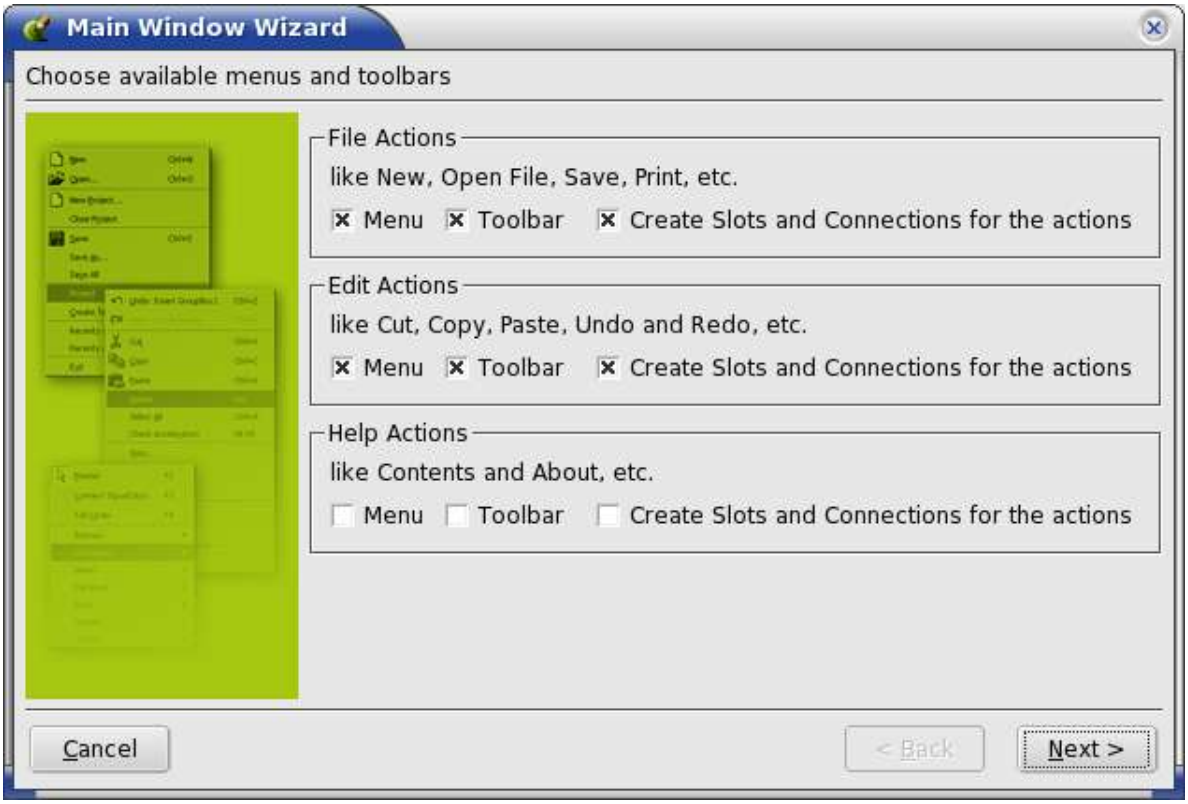
frmtabla.ui
frmtabla.cpp
frmtabla.h
tabla.cpp
tabla.h

main.cpp (plantilla usada en los otros dos programas)

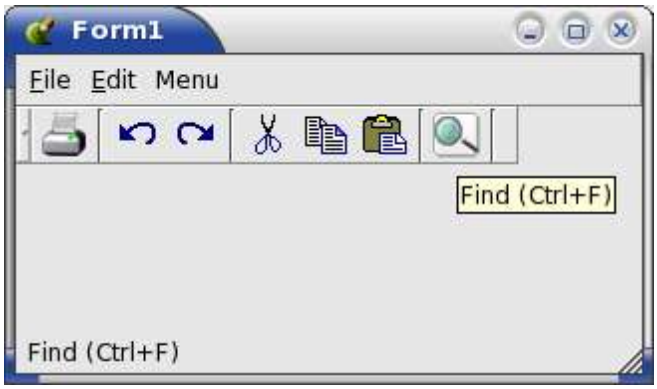


Con lo cual nuestro proyecto MDI debería quedar con los siguientes archivos, tal como se muestra en la figura adyacente, una vez que tenemos todos los archivos en nuestro proyecto, procedemos a crear nuestra interfaz MDI, con lo cual vamos al menú File --> New --> Qt Designer File (*.ui) como nombre ingresamos frmmdi.ui, le decimos que no a la opción de mostrar como texto la interfaz, con lo cual nos abre Qt Designer y elegimos la opción “Main Window”

Nos aparecerá un asistente donde nos preguntara las opciones por defecto que queremos tener en el Menú File, Menú Edit y Menú Help (ayuda) tanto para el menú o para la barra de herramientas (toolbar) y si queremos que nos cree las conexiones, marcamos todas las opciones como se ve a continuación en la siguiente pantalla y hacemos click en “Next >”



En la pantalla posterior nos pregunta que opciones queremos tener del menú en la barra de herramientas, para nuestro ejemplo incluiremos del menú File la opción Print , y del menú Edit todas las opciones, damos Next> y la interfaz nos deberá quedar como muestra la siguiente figura:



Ahora procederemos a modificarlo para adaptarlo a nuestras necesidades, como primera medida es la de cambiar los nombres de los menues a castellano, eliminar algunos ítems que nuestra aplicación no usara y agregar los nuestros a la barra del menú y a la barra de herramientas, para ello nos debemos manejar con las barras de de Qt de *"Action Editor"* y la de *"Property Editor / Signal Handlers"*

Para elegir en la barra de "Property..." un ítem de la barra de menú debemos hacer doble click sobre cualquier ítem, y con la propiedad "Ítem Number" pasar sobre los ítems principales, cambiando el nombre (i*temName*) y el texto que aparece en el menú (*itemText*)
Con el signo Ampersand ("&") seteamos el acelerador para la función, los aceleradores se activan presionando Alt + la letra subrayada de la opción.
Luego de cambiar los nombres nos disponemos a quitar las opciones que no usaremos, para ello nos paramos en la opción que no necesitamos, presionamos el botón secundario del mouse y elegimos la opción "Delete Ítem".
En el menú Archivo dejamos solo la opción Exit, en Editar quitamos Redo y Find y los separadores que hagan falta para que quede bien el menú.
Para cambiar los nombres de los subítems debemos elegirlos de la barra de *"Action Editor"* y cambiarle las propiedades dentro de la barra de herramientas *"Property Editor / Signal Handlers"* , en el caso de Exit su acción se llama *fileExitAction* debemos cambiar las propiedades y asignarle las que aparecen en la tabla siguiente (en este ejemplo se han cambiado solo algunas de las propiedades que tiene el ítem, ya que los demás no son necesarios de poner sí ó sí, como es el caso de los aceleradores de aplicación (por ejemplo activar un ítem con Ctrl + F u otra tecla, solo activamos el acelerador del menú al colocar el signo Ampersand ("&")

Propiedad	*Valor*	*Descripción*
name	archivoSalir	Nombre del ítem
text	Salir	Texto del ítem
menuText	S&alir	Texto que aparece en el menú
tooltip	Salir del Programa	Texto de ayuda con el mouse
statusTip	Salir del Programa	Texto que aparece en la barra de estado
iconset	/emblems.png	Icono del ítem

NOTA: el archivo emblems.png se encuentra en la misma carpeta donde esta guardado el archivo frmmdi.ui, se puede poner cualquier archivo de ejemplo

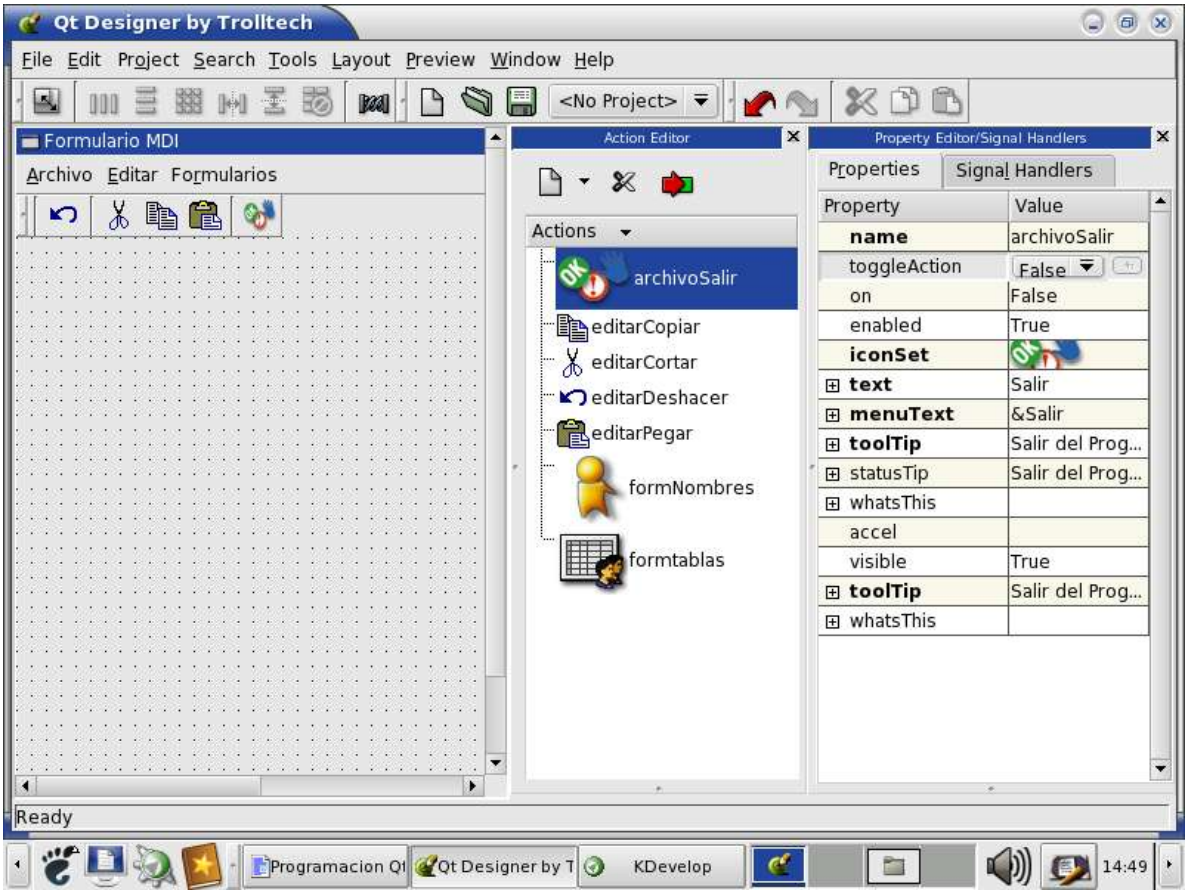
Aplicando la misma metodología, hacemos lo mismo con las opciones del menú Editar.
Una vez que realizamos esto, pasamos a eliminar las acciones que no usamos que se encuentran en la ventana de *"Action Editor"* dejando únicamente las opciones que usamos.

En este momento procedemos a crear nuestras acciones, en la ventana "Action Editor" haciendo click en New action podremos crear las acciones que necesitamos, que son dos y ponerles las siguientes propiedades como se ve en las siguientes tablas:

formNombres	
Propiedad	***Valor***
name	formNombres
text	Nombres
menuText	&Nombres
tooltip	Nombres para Saludar
statusTip	Nombres para Saludar
iconset	/gaim.png

formTablas	
Propiedad	***Valor***
name	FormTablas
text	Tablas
menuText	&Tablas
tooltip	Uso de Tablas en Qt
statusTip	Uso de Tablas en Qt
iconset	/resources.png

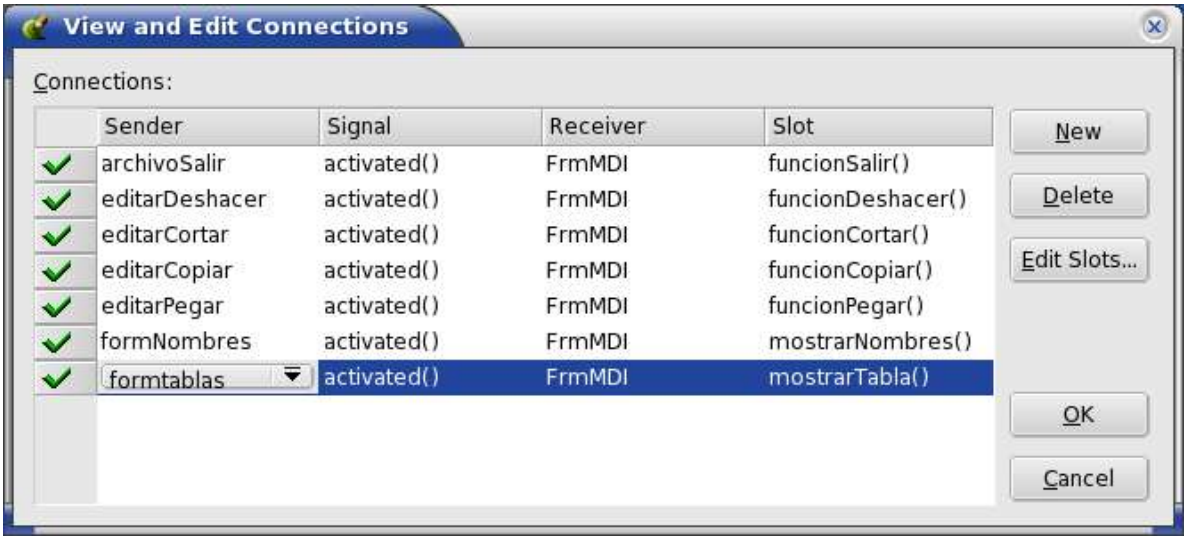
Con lo cual obtenemos la siguiente interfaz:



Ahora procedemos a cambiar los slots de las conexiones creadas anteriormente, con lo cual abrimos la ventana de “View and Edit Connections” y procedemos a realizar los siguientes cambios:

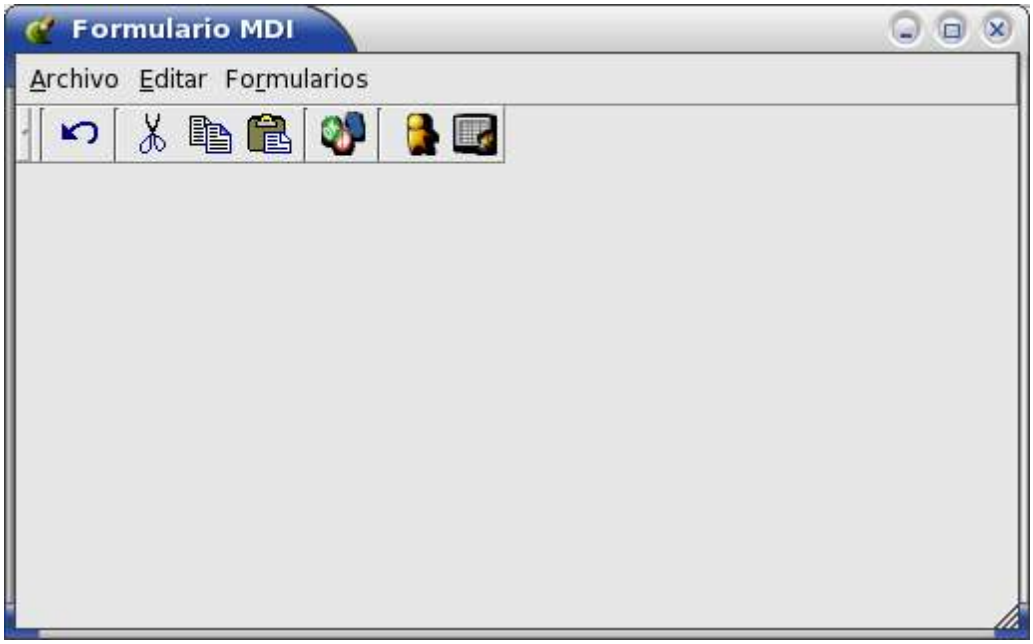
1) Cambiar los nombres de los slots de las conexiones que usamos
2) Agregar nuestros funciones *mostrarNombres()* y *mostrarTabla()*
3) Crear las conexiones nuevas y asignarle las funciones anteriores.
4) Eliminar las funciones que no se usen.

Con lo cual las conexiones deben quedar de esta forma:



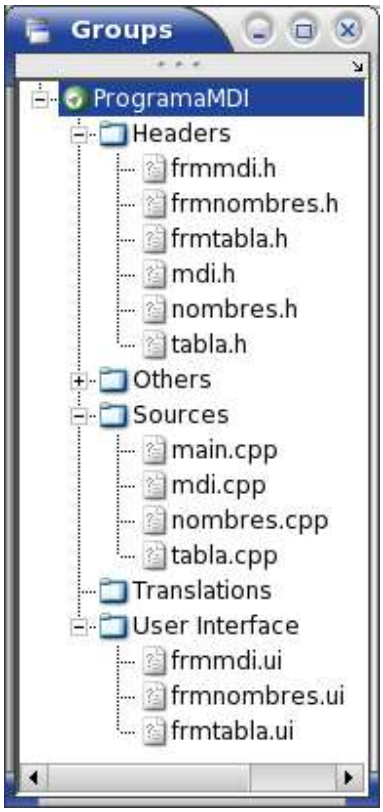
En este momento solo nos queda asignar nuestras dos nuevas acciones al menú *Formularios*, para lo cual utilizamos técnica “Drag & Drop” ya clásica archiutilizada presionando el botón primario del mouse en la acción *formNombres* y sin soltar el botón dejarla dentro del menú *Formularios*, y repetir la misma técnica para formTablas.
Testeamos la interfaz haciendo click en el menú Preview --> Preview Form, chequeamos que haya quedado bien, hacemos los cambios pertinentes en caso contrario, guardamos los cambios y compilamos la interfaz.

En imágen siguiente se puede ver la interfaz terminada, como plus le añadimos los ítems creados por nosotros también a la barra de herramientas.



Volvemos a nuestro proyecto en Kdevelop, y agregamos los archivos generados por el compilador **uic**, de los cuales solo agregamos *frmmdi.h*
En el explorador de archivos de nuestro proyecto quitamos los archivos frmnombres.cpp y frmtablas.cpp, una vez realizadas todas estas tareas, procedemos a crear la clase MDI cuya clase base en FrmMDI, vamos al menú Proyect --> New Class y ponemos los siguientes datos:

Propiedad	*Valor*
classname	MDI
base class	FrmMDI
header file	Mdi.h
implementation file	Mdi.cpp
inheritance	Public
Generated a Qwidget child-class	marcado



Una vez que realizamos esto el proyecto nos debe haber quedado con los siguientes archivos según consta en la figura de la derecha.

Una vez que hicimos esto procedemos a agregar el código en el archivo mdi.h correspondiente a las acciones del menú y de la barra de herramienta, si quisiéramos compilar en este momento el proyecto el compilador nos daría error (además ya tendríamos que haber modificado el archivo main.cpp para que arranque asignándole la clase para el formulario principal MDI.
Para el archivo *main.cpp* el código es el siguiente:

```
#include "mdi.h"            //nombre de la clase creada en Kdevelop
#include "frmmdi.h"         //nombre de la clase creada con uic (ídem form)
#include <qapplication.h>   //si o si base de toda aplicación de Qt

Int main (int argc, char ** argv)  //procedimiento principal
        {
           QApplication Main (argc, argv);          //Creo Aplicación de Qt
           MDI Form;                                      /Creo form a partir de clase
           Form.show();                             //Muestro form
           Form.setCaption("Formulario MDI");       /Asigno caption al form
           return Main.exec ();
         };
```

Como nuestra aplicación será con un formulario MDI que contenga todos los formularios que se ejecuten, debemos crear nuestro espacio de trabajo, el cual pertenece a la clase QWorkspace, con lo cual debemos modificar los archivos mdi.h t frmmdi.h y agregar las siguientes líneas de código (en ambos archivos agregar lo mismo)

```
#include <qworkspace.h>

class QWorkspace;

class MDI : public FrmMDI  //  en frmmdi sera  class FrmMDI : public QMainWindow
{
  Q_OBJECT

private:
        QWorkspace * espacioTrabajo;

//.................SIGUE IGUAL.............................//
```

En *mdi.cpp* empezaremos a agregar las líneas de código para el funcionamiento del entorno MDI, cabe destacar que no modificaremos ninguna línea de código en los archivos *frmnombres.h*, *frmnombres.cpp*, *nombres.h*, *nombres.cpp*, *frmtabla.h*, *frmtabla.cpp*, *tabla.h* y *tabla.cpp*.

Agregamos los siguientes archivos en la sección de includes (archivo *mdi.cpp*)

```
#include "mdi.h"
#include "tabla.h"
#include "nombres.h"
#include <qdockarea.h>
#include <qdockwindow.h>
#include <qvbox.h>
#include <qmessagebox.h>
```

Y procederemos a la codificación de las siguientes funciones:

```
MDI:MDI (QWidget *parent, const char *name ) : FrmMDI(parent,name)
void funcionSalir();
void funcionDeshacer();
void funcionCortar();
void funcionCopiar();
void funcionPegar();
void mostrarNombres();
void mostrarTabla();
```

El primer procedimiento que se ejecutara es la función Constructora MDI::MDI, llamada a partir de nuestra archivo main.cpp, en este procedimiento se creara el espacio de trabajo necesario para mostrar los formularios.

```
MDI::MDI(QWidget *parent, const char *name ) : FrmMDI(parent,name)
{
    QVBox* cajaVertical = new QVBox( this );
```

```
    cajaVertical->setFrameStyle( QFrame::StyledPanel | QFrame::Sunken );
    espacioTrabajo = new QWorkspace( cajaVertical );
    espacioTrabajo->setScrollBarsEnabled( TRUE );
    setCentralWidget( cajaVertical );

    this->showMaximized();
}
```

En la función funcionsalir(), cerraremos la aplicación en caso de no estar abierto ningún formulario, caso contrario mediante un mensaje en pantalla le informaremos que debe cerrar todas las ventanas para cerrar la aplicación.

```
void MDI::funcionSalir()
{
QWidgetList listaVentanas = espacioTrabajo->windowList();
    if ( listaVentanas.count() > 0 )
    {
        QMessageBox::information( 0, "Cerrar Aplicación ",
        "No se puede cerrar la aplicación.\n"
        "Debe cerrar todas las ventanas." );
    }
    else
    {
        this->close();
    }
}
```

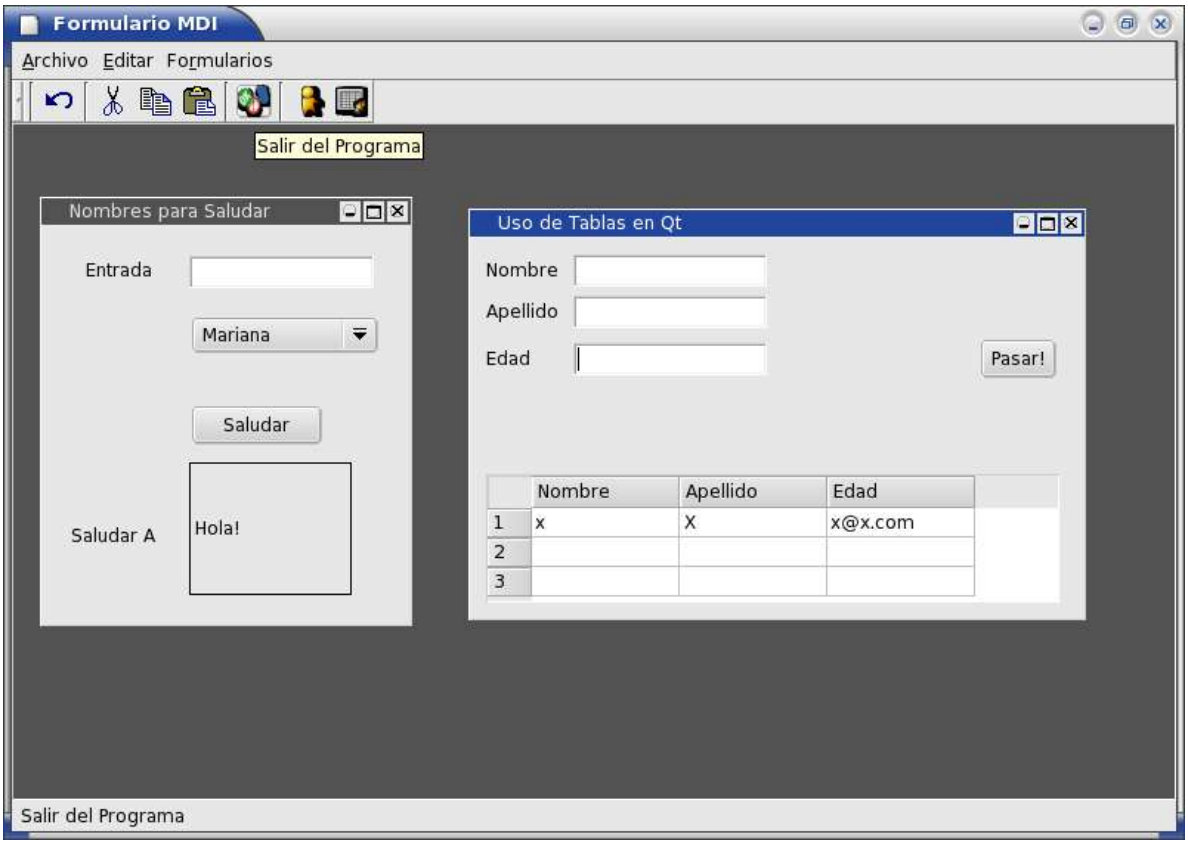
Pasaremos codificar las funciones de los botones Copiar, Pegar, Cortar y Deshacer:

```
void MDI::funcionDeshacer()
{
}

void MDI::funcionCortar()
{
}

void MDI::funcionCopiar()
{
}

void MDI::funcionPegar()
{
}
```

Por ultimo, codificaremos los procedimientos *mostrarNombres()* y *mostrarTabla()*, con los cuales mostraremos los formularios dentro del Espacio de Trabajo de nuestro formulario MDI (*espacioTrabajo*)

```
void MDI::mostrarNombres()
{
        Nombres *nombres = new Nombres(espacioTrabajo);
        nombres->show();
}

void MDI::mostrarTabla()
{
        Tabla *tabla = new Tabla (espacioTrabajo);
        tabla->show();
}
```

Guardamos los cambios, y ejecutamos el programa, quedando nuestra aplicación de esta forma (la pantalla esta maximizada y para el ejemplo abrimos los dos formularios)



Imágen de nuestra aplicación MDI funcionando, el código completo de los archivos *mdi.cpp*, *mdi.h*, *frmmdi.cpp* y *frmmdi.h* y *main.cpp* los encontraran en el apéndice D de este manual.

CREACIÓN DE UNA APLICACIÓN CON ACCESO A BASE DE DATOS mySQL

En este capítulo procederemos a la creación de una aplicación con la capacidad de acceder a base de datos mySQL, para ello antes de la creación de los archivos de la aplicación deberemos asegurarnos de tener en ejecución el servicio de base de datos *mysql* y la forma de utilizar *mysqladmin* para la creación de la base de datos y la creación de los datos.
Para saber si tenemos corriendo el servicio de base de datos, abrimos una terminal y ejecutamos el siguiente comando, si todo anda bien debería retornarnos lo siguiente:

```
[root@cloud martin]# mysqlshow
+---------------+
| Databases     |
+---------------+
|  tmp          |  //la base de datos tmp puede estar o no
+---------------+
```

En nuestro caso utilizaremos una base de datos creada por nosotros llamada compañeros, en mySQL se pueden crear base de datos enviándole al motor de base de datos la consulta SQL usando lenguaje **DDL** (**D**ata **D**efinition **L**anguaje) para la creación de la misma, en el siguiente ejemplo procederemos a la creación de la base de datos usando el modo texto, en caso de no querer lidiar con la consola, esta disponible en Internet un front-end para el manejo de mySQL en formato de páginas en formato PHP llamado *phpMyAdmin*.

La consulta SQL en formato DDL es la siguiente, la cual guardaremos en un archivo llamado por ejemplo companeros.sql

DB_companeros.sql

```
CREATE TABLE `personas`
(
  `IdPersona` int(3) NOT NULL auto_increment,
  `Apellido` varchar(30) NOT NULL default '',
  `Nombre` varchar(30) NOT NULL default '',
  `email` varchar(60) NOT NULL default '',
  `Edad` int(2) NOT NULL default '0',
  PRIMARY KEY  (`IdPersona`)
)
TYPE=MyISAM AUTO_INCREMENT=0 ;
```

En nuestra terminal abierta ejecutamos las siguientes líneas de código, recuerden ejecutar estas líneas de código en la carpeta donde guarden el archivo *DB_companeros.sql*.

[root@cloud programasql]# mysqladmin createdatabasename compañeros

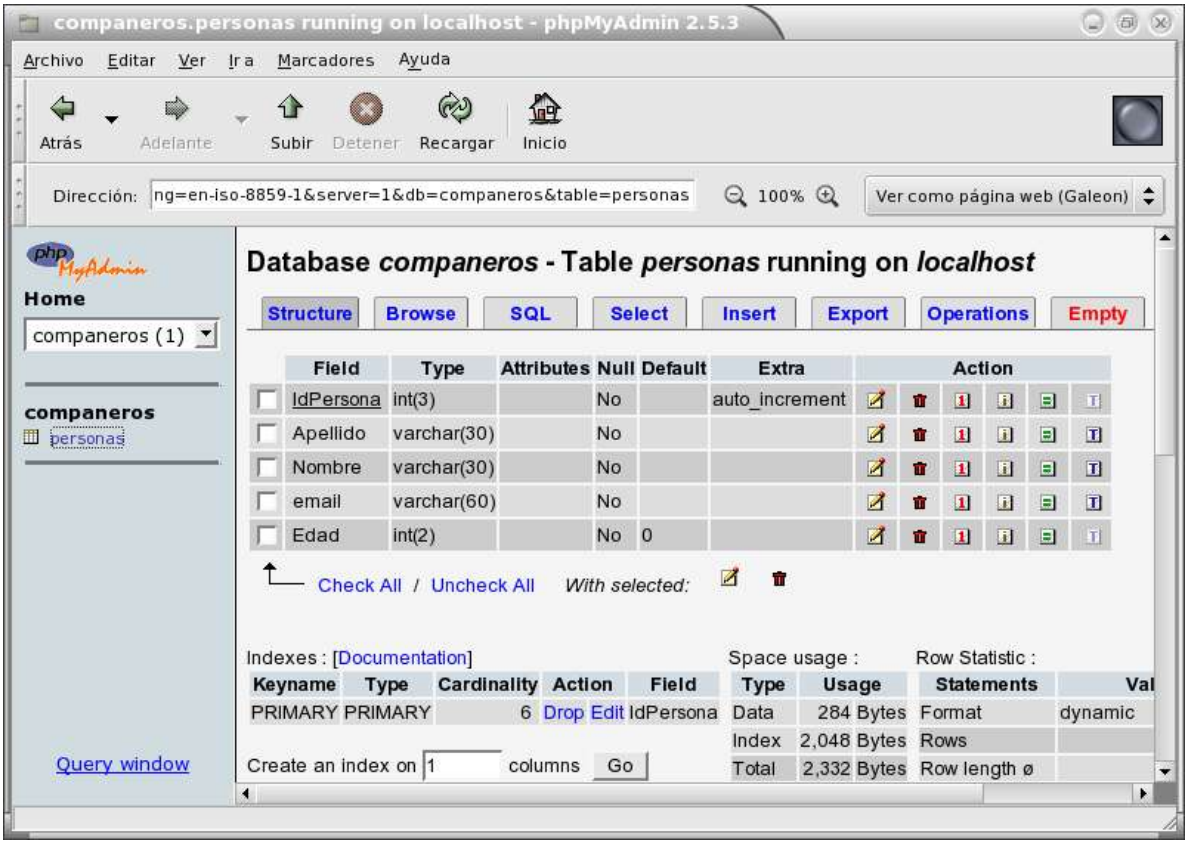
[root@cloud programasql]# mysql compañeros < BD_companeros.sql

Chequeamos que fue creada con:

```
[root@cloud martin]# mysqlshow
+---------------+
| Databases     |
+---------------+
|companeros     |
|tmp            |
+---------------+
```

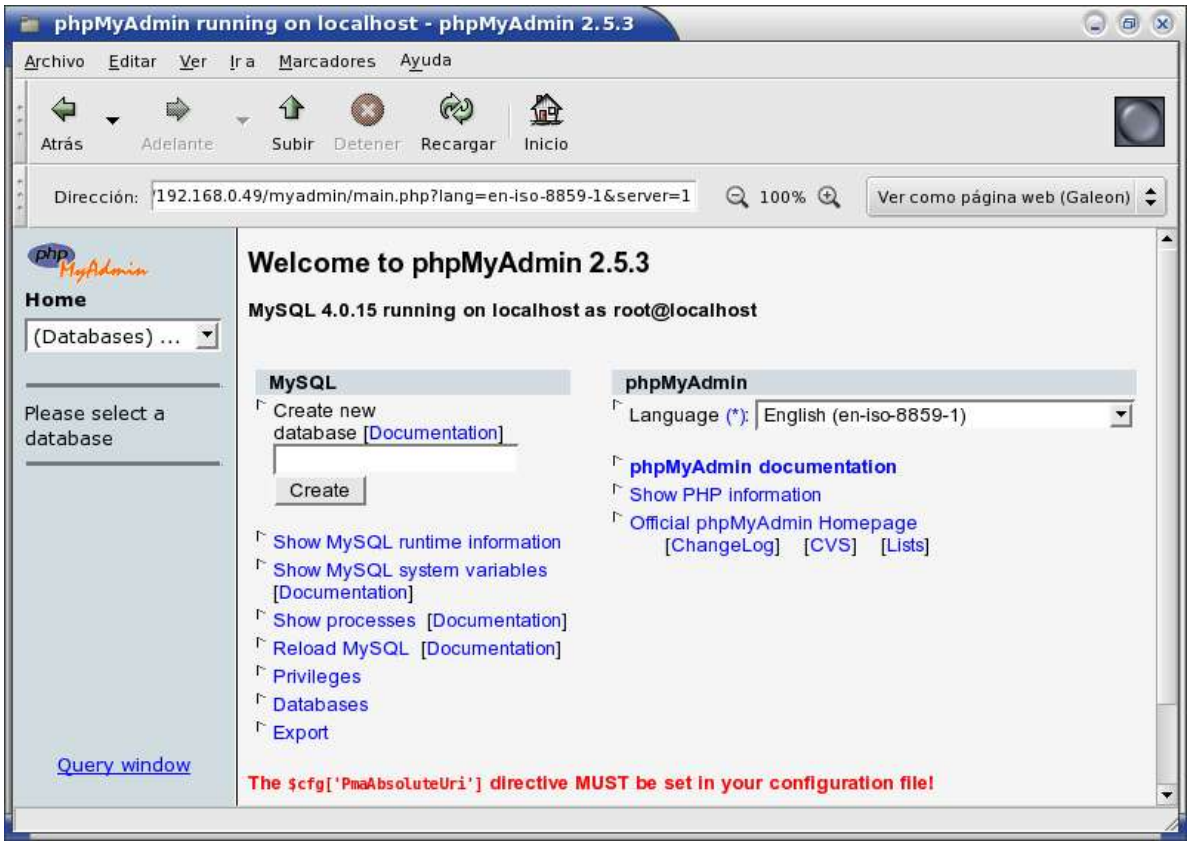
Al ejecutar estas líneas de código ya tendremos nuestra base de datos creada con la estructura que aparece en el archivo DB_companeros.sql

En la imágen inferior se puede ver el entorno de administración de base de datos *phpMyAdmin*.



En nuestra aplicación crearemos un usuario con el cual nos conectaremos a nuestra base de datos, en caso de que no puedan contar con este programa (ya que con el mismo crearemos este usuario) pueden usar el usuario “roo” cuya contraseña es “” (no tiene contraseña) , es por este motivo que crearemos un nombre de usuario y contraseña, ya que de lo contrario nuestra base de datos será muy vulnerable.
Igualmente también es altamente recomendable colocarle contraseña al root de la base de datos.

En la pantalla principal de phpMyAdmin, en la pantalla principal, elegimos la opción “Privileges” como se muestra en la siguiente figura:

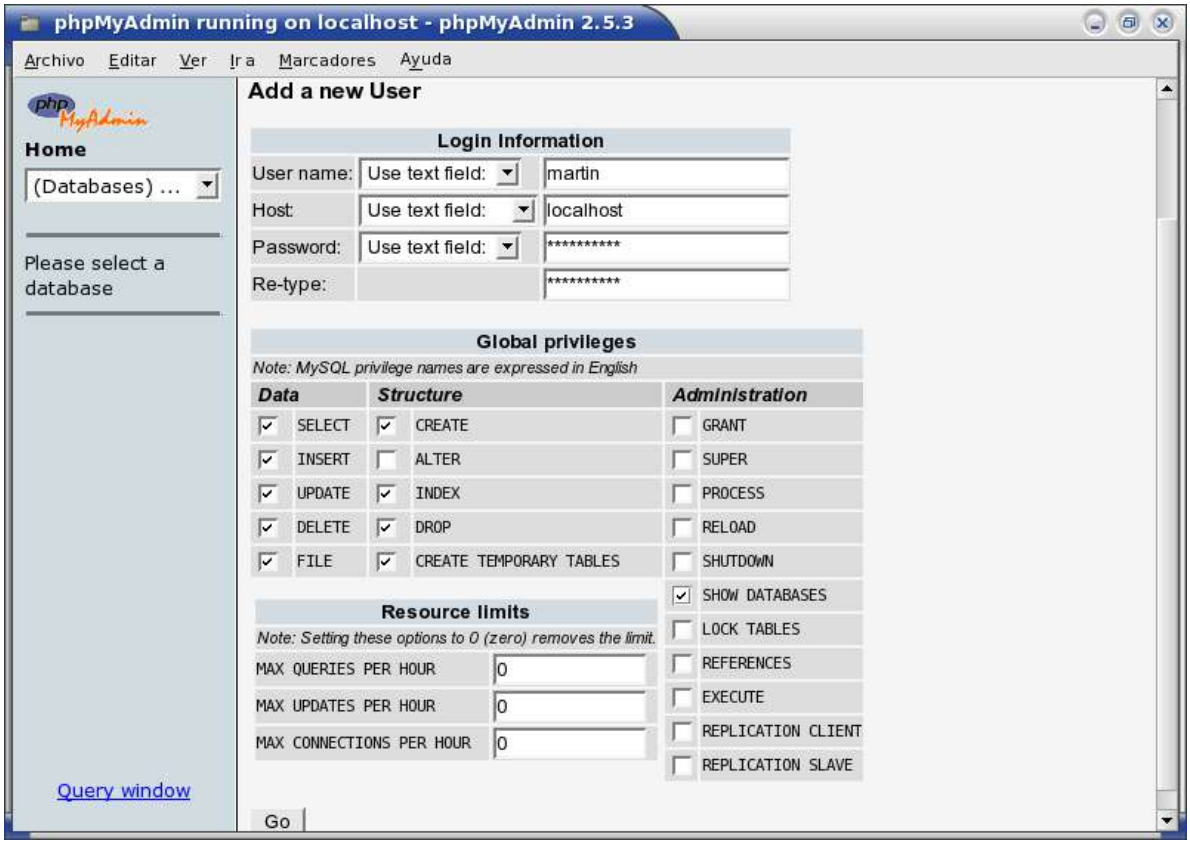


Una vez que ingresamos, hacemos click en la opción “Add new User”, se nos desplegara una nueva pantalla donde ingresaremos los siguientes datos:

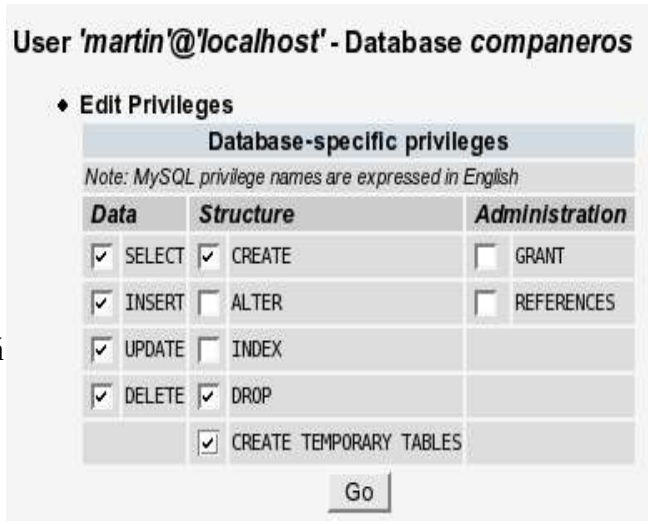
User name: *Nombre de usuario*
Host: *Ámbito de acción para el usuario* (si será en esa maquina poner **localhost**)
Password: *Contraseña*

Para nuestro ejemplo:
UserName: martin
Host: localhost
Password: companeros

y los Privilegios globales del usuario, para nuestro ejemplo daremos privilegios de todos los comandos SQL y de administración de ver las bases de datos de la máquina, todo debería quedar como muestra la siguiente imágen:



Una vez realizado los cambios, presionamos el botón “Go”, donde se nos mostrara una nueva pantalla con la información ya procesada, nos movemos a la sección llamada *“Database-specific privileges”* y en el combo al lado de la leyenda *“Add privileges on the following database”*, seleccionamos la base de datos creada anteriormente compañeros, con lo cual nos cambiara a una nueva pantalla donde ingresaremos los privilegios que tendrá este usuario en esta base de datos específicamente, elegimos todas las opciones de las consultas SQL (sección Data y Structure) tal como se muestra en la siguiente imágen y hacemos click en el botón “Go”.



Una vez realizados todos estos pasos habremos creado un usuario con el cual conectarnos a la base de datos y tener los permisos para agregar, editar, y eliminar registros de la base de datos y la posibilidad de crear o eliminar tablas dentro de la misma.

Es en este momento en el cual procederemos a la creación de nuestra aplicación para manejar la información dentro de nuestra base de datos *companeros*.

Abrimos Kdevelop, vamos al menú Project --> New

Elegimos la opción Qt --> Qt SDI, a este proyecto lo llamaremos mySQLApp, y las demás opciones las desmarcamos:

generate sources and headers
GNU Standart-Files (INSTALL,README,COPYING...)
User-Documentation
LSM (Linux Software Map)
VCS Support
headertemplate for .h-files (opcional)
headertemplate for .cpp-files (optional)

Y hacemos click en el botón *Create*.

Con esto creamos el proyecto, ahora procederemos a copiar de alguno de nuestros anteriores proyectos el archivo *main.cpp* dentro de la carpeta y lo agregamos al proyecto.

Una vez realizado esto pasamos a crear la interfaz de nuestra aplicación, vamos al menú File --> New --> Qt Designer file (*.ui) y los llamamos *frmmysql.ui*, desmarcamos la opción *use Template* y presionamos el botón Ok.
En la ventana que despliega Qt Designer elegimos la opción “Widget” y damos OK. Para la manipulación de los datos procederemos a la creación del formulario con el cual realizaremos las operaciones de Agregar, Editar o Eliminar registros de la Base de Datos mySQL llamada compañeros, mediante el uso de consultas SQL (Structured Query Language ó Lenguaje de Consultas Estructurado) mediante el uso de las sentencias SELECT, UPDATE, INSERT y DELETE y los objetos que Qt incorpora para el manejo de base de datos.

La interfaz estará compuesta por los siguientes elementos:

1) Un Formulario llamado *FrmMySQL*.

2) 4 LineEdit para modificar los datos o agregar nuevos en la BD.
   1. *txtNombre*
   2. *txtApellido*
   3. *txtEmail*
   4. *txtEdad*

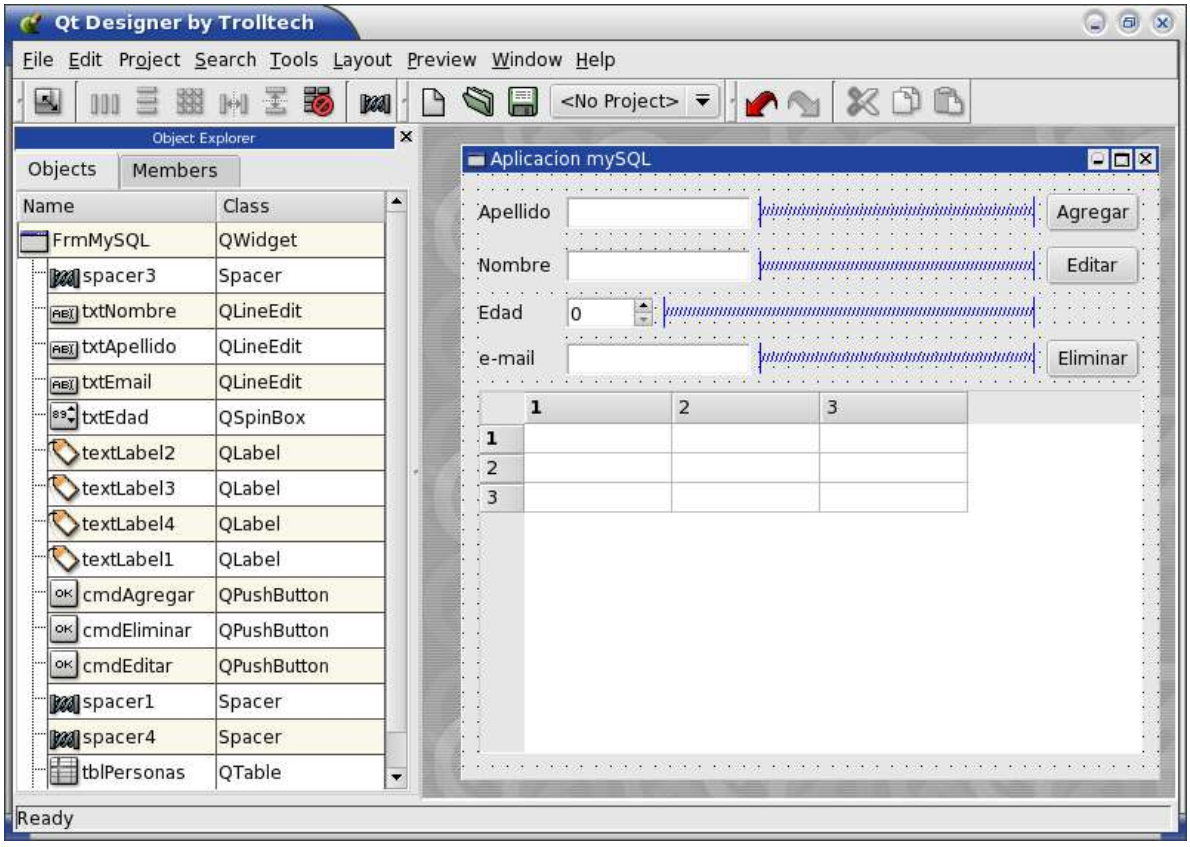
3) 1 Tabla para mostrar los datos de la BD o elegir uno. (*tblPersonas*) del tipo readOnly para que únicamente muestre datos y su procedimiento *cargarTabla()*.

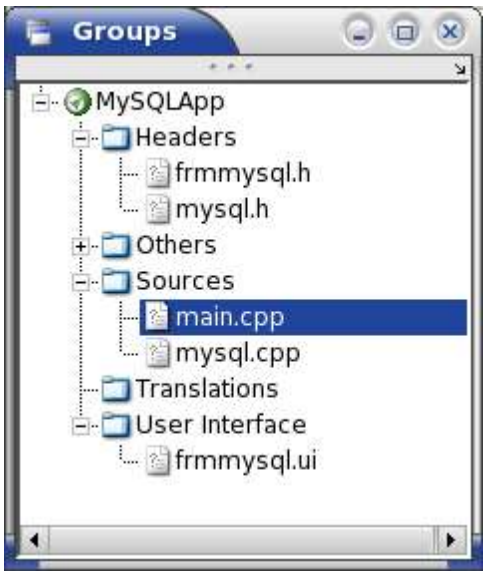
4) 3 PushButton para las operaciones de Agregar, Editar o Eliminar.
   1. *cmdAgregar*
   2. *cmdEditar*
   3. *cmdEliminar*

5) Los Spacer horizontales y verticales para mantener el aspecto del formulario.

6) Las acciones que realizara el programa que serán las siguientes:
   1. botón Agregar (*agregarPersonas()*)
   2. botón Editar (*editarPersonas()*)
   3. botón Eliminar e ítem seleccionado en tabla (*eliminarPersonas()*)
   4. Selección en Tabla /Doble click (*cargarPersonas()*)

Luego de realizar todos estos pasos, la interfaz nos deberá quedar como muestra la siguiente figura:





Una vez que terminamos de confeccionar la interfaz la guardamos compilamos la interfaz con **uic** como hemos realizado en nuestros anteriores programas y agregamos al proyecto en Kdevelop el archivo *frmmysql.h*, una vez que agregamos el archivo de cabecera, procedemos a crear la clase con la cual trabajaremos en Kdevelop (Proyect --> New Class) , la cual llamaremos *MySQL*, cuya base class en *FrmMySQL*.
Una vez que realizamos estos pasos, obtenemos los siguientes archivos en nuestro proyecto de Kdevelop como se puede observar en la figura a la derecha.
En estos momentos lo que queda de nuestra aplicación es la codificación de los eventos, para lo cual debemos conocer un poco mas a fondo como es que manejaremos la base de datos desde nuestra aplicación, para esto, las librerías Qt incorpora un módulo llamado SQL, en el cual se encuentran una serie de objetos con los cuales podremos manejar los datos de la tabla y agregar nuevos, crear nuevas tablas (mediante consultas DDL) y un sinfín de operaciones más, nosotros para nuestra aplicación usaremos dos objetos , los cuales son:

Objeto (clase)	***Archivo *.h***	***Uso***
QSqlDatabase	qsqldatabase.h	Conectarse a Base de Datos
QSqlQuery	qsqlquery.h	Ejecutar Consultas SQL

Con QSqlDatabase crearemos la conexión a la base de datos mientras que con QSqlQuery ejecutaremos las consultas SQL necesarias para agregar (INSERT) , editar (UPDATE), eliminar (DELETE) y mostrar (SELECT).

En la siguiente página se explicaran más en detalle estas dos clases para una compresión acerca de su uso y su aplicación en nuestro programa.

CLASE QSQLDATABASE

Esta clase nos permite abrir una conexión apuntando a una determinada base de datos, se pueden tener abiertas al mismo tiempo varias base de datos y de distintos tipos, ya que al momento de crear la conexión debemos establecer que Driver usara, los cuales pueden ser:

Driver	***Gestor de Base de Datos que soporta***
QODBC3	Open Database Conectivity
QOCI8	Oracle 8 y 9
QTDS7	Sybase Adaptative Server y Microsoft SQL Server
QPSQL7	PostgreSQL 6 y 7
QMYSQL3	MySQL

Lo primero que debemos hacer es activar el driver ejecutando QSqlDatabase::addDatabase(), enviando el nombre del driver que deseamos utilizar para esta conexión.
La conexión es creada convirtiéndose en la conexión a base de datos por defecto y es la utilizada por todas las clases de SQL de Qt si no especificamos el nombre de la conexión.
La forma de codificación de la llamada *QSqlDatabase::addDatabase()* es:

QSqlDatabase **NombreBD* = QSqlDatabase::addDatabase(*Driver* , *identificador*)
// NOTA: Se crea siempre un puntero a la BD.

Una vez que creamos la conexión, llamamos las propiedades *setDatabaseName(), setUserName (), setPassword()* y *setHostName()* para inicializar la información de la conexión. Una vez que seteamos la información de la conexión, la abrimos con el comando *open()* dándonos acceso a los datos. Si esta llamada falla retornara FALSE, la información sobre el error puede ser obtenida desde *QSqlDatabase::lastError()*

Construir una conexión a base de datos es un proceso simple que consta de 3 pasos: activar el driver, configurar la informacion de la conexión, y abrirla.
La función estática *QSqlDatabase::addDatabase()* puede ser llamada desde cualquier lugar para proveer un puntero a una conexión de base de datos. Si la llamamos sin ningún parámetro retornara la conexión por defecto. Si la llamamos con un identificador que hemos usado para la conexión, retornara un puntero a esa conexión.
Para remover una base de datos de la lista de conexiones ejecutándose, primero debemos cerrar la base de datos con *QSqlDatabase::close()* y luego removerlo con la función *QSqlDatabase::removeDatabase()*
Se puede tener varias conexiones a base de datos abiertas al mismo tiempo, las cuales se distinguen a partir del identificador que les hallamos dado al momento de crearlas (segundo parámetro de la función *QSqlDatabase::addDatabase()*)

A continuación se provee un ejemplo para abrir una base de datos mySQL:

En la sección de declaración de los headers (includes)

```
include <qsqldatabase.h>

void abrirConexionBD()
{
QSqlDatabase *DBCompaneros = QSqlDatabase::addDatabase ("QMYSQL3", "companeros");
DBCompaneros->setDatabaseName ("companeros");
DBCompaneros->setUserName ("martin");
DBCompaneros->setPassword ("companeros");
DBCompaneros->setHostName ("localhost"); // 127.0.0.1 (Standart Protocolo TCP/IP)

  if ( DBCompaneros->open() );
  }
        //la consulta devuelve registros, colocar  el código correspondiente
  }
  else
  {
        QMessageBox::information( 0, "Fallo en conexión ",
        "Ocurrió un error al querer conectarse a la Base de Datos companeros \n"
        + DBCompaneros->lastError  );
  }
}

void cerrarConexionBD()
{
DBCompaneros>close();
DBCompaneros->removeDatabase();
}
```

A continuación en la próxima hoja veremos la clase QSqlQuery

CLASE QSQLQUERY

Esta clase nos permite ejecutar consultas SQL en una conexión a base de datos, la cual ya debe estar abierta. Contiene funciones para navegar por los resultados de un conjunto de registros traídos con una consulta SELECT y devuelve registros individuales y los valores de los campos.

Para consultar si la consulta fue ejecutada exitosamente (devuelve registros) la propiedad *isActive()* devuelve TRUE. Para navegar por los registros de la consulta nos provee cuatro funciones, las cuales son: *next(), prev(), first()* y l*ast()* y cada registro puede ser chequeado para comprobar si tiene datos con la propiedad *isValid()*

Se puede ir a un registro en particular usando la función *seek()*. El primer registro de la consulta es cero (0). La cantidad de registros que devuelve la consulta se puede ver con la propiedad *size ()*.

Se puede consultar el texto de la última consulta ejecutada en la base de datos con la propiedad *lastquery()*. Esto es altamente útil para corroborar si la consulta que se esta ejecutando es la que piensas que es y no otra.

La forma de declarar un objeto QSqlQuery y devolver datos a determinados objetos es la siguiente :

QSqlQuery nombre_consulta (*consulta_SQL*, *nombre_Conexion*)

En la sección de declaración de los headers (includes)

```
include <qsqlcursor.h>

void retornarRegistros()
{
//la base de datos ya debe haber sido abierta anteriormente.

QSqlQuery consultaBD (“SELECT IdPersonas, Nombre, Apellido, email , Edad FROM personas WHERE 1” , DBCompaneros )

  if ( consultaBD.isActive() )
  {
    txtNombre->setText ( consultaBD value(1).toString ) ;
    txtApellido->setText ( consultaBD.value(2).toString ) ;
  }

}
```

Una vez ya explicados estas clases, volveremos a nuestra aplicación para dotar a nuestro programa de las capacidades de acceso a datos.

Codificación en kdevelop

Nuestro programa contara con las siguientes funciones, las cuales nos proponemos a codificar:

```
virtual void agregarPersonas();
virtual void editarPersonas();
virtual void eliminarPersonas();
virtual void cargarTabla();
virtual void cargarPersonas();
```

Lo primero que debemos hacer es agregar las siguientes líneas de código en la declaración de la clase MySQL (*mysql.h*)

```
private:
   int  esNuevo; //bandera para saber si el ítem a agregar es nuevo

public slots:

   virtual void agregarPersonas();
   virtual void editarPersonas();
   virtual void eliminarPersonas();
   virtual void cargarTabla();
   virtual void cargarPersonas();
```

La declaración de la conexión a nuestra base de datos junto con la informacion de la conexión la incluiremos en la función constructora de la clase MySQL, entonces quedara codificada de la siguiente manera (en *mysql.cpp*) para luego llamarla con la función en cualquier momento que necesitemos acceder a la base de datos usaremos::

```
 QSqlDatabase *DBCompaneros = QSqlDatabase::database (  “companeros” ) identificador
/*************************************************************************/
//Agregar las siguientes líneas en la sección de includes:
#include <qsqldatabase.h>
#include <qsqlquery.h>

MySQL::MySQL (QWidget *parent, const char *name ) : FrmMySQL(parent,name)
{
QSqlDatabase *DBCompaneros = QSqlDatabase::addDatabase ("QMYSQL3", "companeros");
DBCompaneros->setDatabaseName ("compañeros");
DBCompaneros->setUserName ("martin");
DBCompaneros->setPassword ("compañeros");
DBCompaneros->setHostName ("localhost");
DBCompaneros->open();
tblPersonas->setReadOnly (TRUE);                    //seteamos la tabla a solo lectura
MySQL::cargarTabla();                          //llamamos al procedimiento cargarTabla()
esNuevo=0;
}
```

La función *cargarTabla()* quedará de la siguiente manera:

```
void MySQM::cargarTabla()
{
QSqlDatabase *DBCompaneros = QSqlDatabase::database ("personas");

QSqlQuery Consulta( "SELECT IdPersona, Apellido, Nombre, email, Edad FROM personas
WHERE 1;", DBCompaneros );

tblPersonas->setNumRows (Consulta.size());
tblPersonas->setNumCols (5);
int contador=0;

//construimos un objeto para los encabezados de las columnas
//y le damos los valores correspondientes.

QHeader *titulosTabla = tblPersonas->horizontalHeader();
titulosTabla->setLabel( 0, ("Id") );
titulosTabla->setLabel( 1, ("Apellido") );
titulosTabla->setLabel( 2, ("Nombre") );
titulosTabla->setLabel( 3, ("Email") );
titulosTabla->setLabel( 4, ("Edad") );

//Asignamos los anchos a las columnas de la tabla

tblPersonas->setColumnWidth( 0, 30 );
tblPersonas->setColumnWidth( 1, 80 );
tblPersonas->setColumnWidth( 2, 80 );
tblPersonas->setColumnWidth( 3, 170 );
tblPersonas->setColumnWidth( 4, 40 );

//Llenamos la tabla con los valores de devolvió la consulta.

while ( Consulta.next() )
  {
        tblPersonas->setText( contador , 0, Consulta.value(0).toString() );
        tblPersonas->setText( contador , 1, Consulta.value(1).toString() );
        tblPersonas->setText( contador , 2, Consulta.value(2).toString() );
        tblPersonas->setText( contador , 3, Consulta.value(3).toString() );
        tblPersonas->setText( contador , 4, Consulta.value(4).toString() );
        contador++;
   }
}
```

La función *AgregarPersonas()* limpiara las cajas de los campos Nombre, Apellido, Edad y e-mail, y posicionará el cursos en la primera caja, la cual es *txtApellido* y cambiará el caption del botón Editar por Guardar.

```
void MySQL::agregarPersonas()
{
txtApellido->clear();
txtNombre->clear();
txtEdad->setText(0);
txtEmail->clear();
txtApellido->setFocus();
cmdEditar->setText("Guardar");
esNuevo=1;
}
```

La función editarPersonas() deberá chequear si el caption del botón es guardar o editar, además deberá chequear si es un ítem nuevo o no. Dependiendo estas variables el programa guardara un ítem nuevo o editara uno existente. pasemos al código de este evento:

```
void MySQL::editarPersonas()
{
QSqlDatabase *DBCompaneros = QSqlDatabase::database ("personas");
QSqlQuery agregarRegistro ( "SELECT * FROM personas where 1" , DBCompaneros);

if ( esNuevo==1 )
{
switch ( QMessageBox::information ( 0, "Agregar Registro",
                                    "Esta a punto de insertar el siguiente registro: \n\n"
                                    + txtApellido->text() + ", "
                                    + txtNombre->text() + "\nEmail: "
                                    + txtEmail->text() + ".\nEdad: "
                                    + txtEdad->text()  + ".\n\n\Desea Continuar?",
                                    QMessageBox::Yes,  QMessageBox::No )  )
        {

        case QMessageBox::Yes:
                agregarRegistro.exec ( "INSERT INTO personas ( Apellido, Nombre,
        email, Edad ) VALUES ( '" + txtApellido->text() + "', '" +
        txtNombre->text() + "', '" + txtEmail->text()  + "', " + txtEdad->text() + " );" );
                break;
                }

        }
```

```
else
        {
        switch ( QMessageBox::information ( 0, "Actualizacion de Registro",
                        "Esta a punto de actualizar el siguiente registro: \n\n"
                        + tblPersonas->text( tblPersonas->currentRow(), 1 ) + ", "
                        + tblPersonas->text( tblPersonas->currentRow(), 2 ) + "\nEmail: "
                        + tblPersonas->text( tblPersonas->currentRow(), 3 ) + ".\nEdad: "
                        + tblPersonas->text( tblPersonas->currentRow(), 4 ) + ".\nA:\n"
                        + txtApellido->text() + ", " + txtNombre->text() + "\nEmail: "
                        + txtEmail->text() + ".\nEdad: " + txtEdad->text()  + ".\n\n\Desea
                        Continuar?",
                        QMessageBox::Yes,  QMessageBox::No )  )
        {
        case QMessageBox::Yes:
                        agregarRegistro.exec (" UPDATE personas SET Apellido = '" +
                txtApellido->text() + "', " + " Nombre ='" + txtNombre->text() + "', " +
                        " email ='" + txtEmail->text() + "', " + " Edad =" + txtEdad->text() +
                        " WHERE IdPersona =" +
                tblPersonas->text( tblPersonas->currentRow(), 0 ) +";" );
        break;
        }
}
MySQL::cargarTabla();
cmdEditar->setText("Editar");
txtApellido->clear();
txtNombre->clear();
txtEdad->setText("0");
txtEmail->clear();
txtApellido->setFocus();
esNuevo=0;
}
```

Con esto lograremos que nos pregunte antes de guardar los cambios o insertar una nueva persona con los siguientes mensajes como se puede ver en las imágenes inferiores:

Agregar registro

Actualizar Registro

A continuación procederemos a la codificación de la acción de eliminar un registro de la base de datos, la cual la haremos con una consulta SQL del tipo DELETE, antes de eliminar preguntaremos al usuario si esta seguro de la acción:

```
void MySQL::eliminarPersonas()
{
switch ( QMessageBox::information ( 0, "Eliminación de Registro",
                        "Esta a punto de eliminar el siguiente registro: \n\n"
                        + tblPersonas->text( tblPersonas->currentRow(), 1 ) + ", "
                        + tblPersonas->text( tblPersonas->currentRow(), 2 ) + "\nEmail: "
                        + tblPersonas->text( tblPersonas->currentRow(), 3 ) + ".\nEdad: "
                        + tblPersonas->text( tblPersonas->currentRow(), 4 ) + ".\n",
                        QMessageBox::Yes,  QMessageBox::No )  )
        {
        case QMessageBox::Yes:

                QSqlDatabase *DBCompaneros = QSqlDatabase::database ("personas");
                QSqlQuery borrarRegistro ("DELETE FROM personas WHERE IdPersona ='"
                + tblPersonas->text( tblPersonas->currentRow(), 0 ) + "';", DBCompaneros);
                MySQL::cargarTabla();
                break;
        }
}
```

Otra funcionalidad de nuestro programa es la editar un ítem de la base de datos, para lo cual lo debemos seleccionar de la tabla con un doble click en la tabla, la codificación del procedimiento es el siguiente:

```
void MySQL::cargarPersonas()
{
QSqlDatabase *DBCompaneros = QSqlDatabase::database ("personas");
QSqlQuery cargarRegistro ( "SELECT Apellido, Nombre, email, Edad FROM personas
        where IdPersona ='" + tblPersonas->text( tblPersonas->currentRow(), 0 ) + "';" ,
        DBCompaneros);

cargarRegistro.first();
txtApellido->setText ( cargarRegistro.value(0).toString() );
txtNombre->setText (cargarRegistro.value(1).toString() );
txtEmail->setText (cargarRegistro.value(2).toString() );
txtEdad->setText (cargarRegistro.value(3).toString() );

cmdEditar->setText("Guardar");
}
```

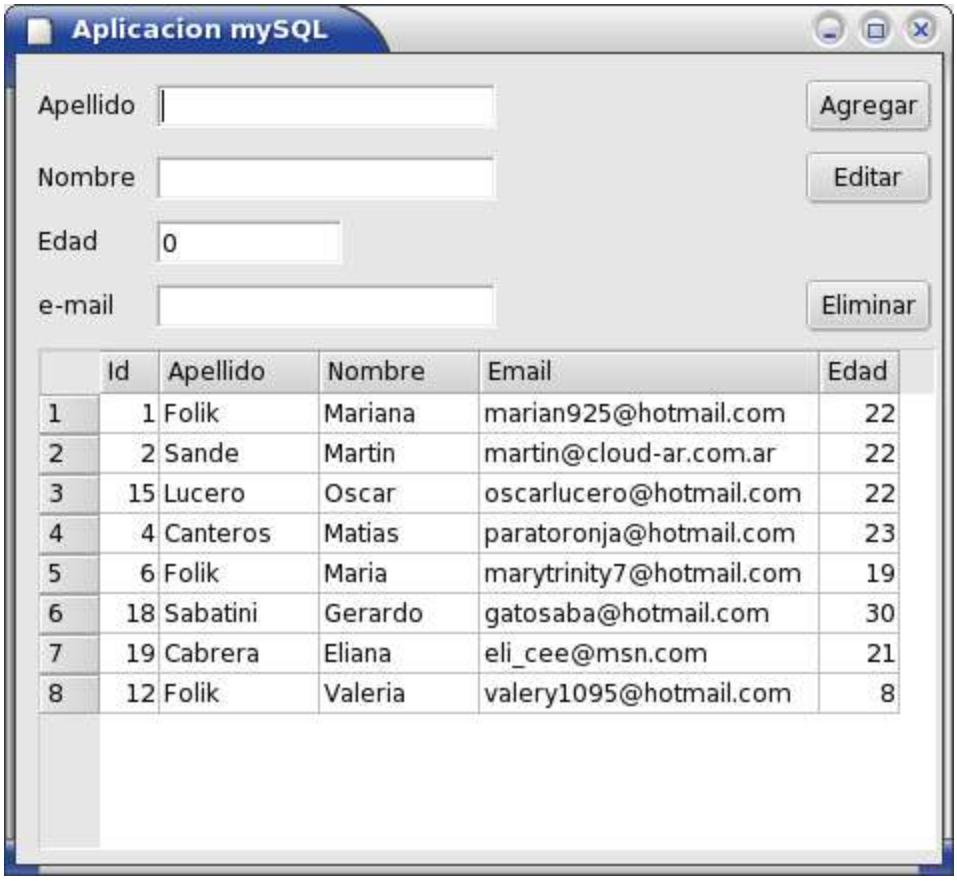
Como broche final a nuestra aplicación le daremos la posibilidad de que si apretara la cruz de la ventana para cerrar la aplicación de le consulte si realmente desea salir de la aplicación , esto se maneja con una función propia de todas las ventanas que es *closeEvent()* (incluyendo la librería qevent.h)
En la definición de la clase MySQL (mysql.h) agregamos las siguientes líneas:

```
//sección include
#include <qevent.h>

//sección definición clase MySQL (después de public:
protected:
        void closeEvent( QCloseEvent* );
```

En el archivo de implementación (mysql.cpp) agregamos las siguientes líneas de código:

```
//section include
#include <qevent.h>

void MySQL::closeEvent ( QCloseEvent* ventana )
{
switch ( QMessageBox::information ( 0, "Salir del Programa",
                "Esta seguro que desea Salir del Programa?\n",
                QMessageBox::Yes , QMessageBox::No ) )
        {
        case QMessageBox::Yes:
                        ventana->accept();
                        break;
        case QMessageBox::No:
                        ventana->ignore();
                        break;
        }
}
```

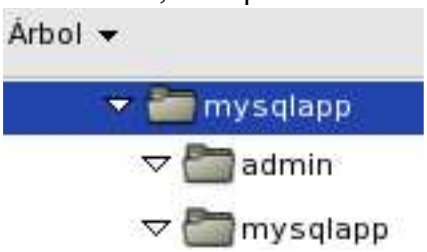
Programa en Funcionamiento:



El código completo de los archivos *mysql.cpp*, *mysql.h*, *main.cpp* los encontraran en el apéndice E de este manual.

DISTRIBUYENDO NUESTRA APLICACIÓN GNU/LINUX BAJO LA LICENCIA GPL

Una vez que realizamos nuestra aplicación sólo nos basta distribuir la misma, si la misma la distribuimos bajo los términos de la licencia GPL debemos agregar dentro del directorio creado anteriormente. En nuestro caso para este ejemplo usaremos la aplicación MySQL (mysqlapp) la cual se encuentra contenida en un directorio con el mismo nombre y dentro del cual se encuentra todos los archivos necesarios para la distribución, el esquema del directorio mysqlapp debería ser el siguiente:



Es usual crear los siguientes archivos y colocarlos en el directorio principal (mysqlapp)

README	Archivo informativo de las funciones del programa y los cambios.
LICENSE	Archivo en el cual se puede leer la licencia GPL
INSTALL	Archivo con las instrucciones para la instalación del programa.
TODO	Informacion adicional no incluida en los 3 archivos anteriores.

Ahora procederemos a crear el archivo con el cual distribuiremos nuestra aplicación, para el cual usaremos dos programas GNU, *tar* y *gzip*, con tar comprimiremos los archivos guardando la estructura de directorios y permisos y con *gzip* la comprimiremos a un 25-30% del tamaño del archivo *tar* mediante el algoritmo de compresión de Lempel-Ziv (LZ77).

La secuencia de creación es la siguiente:

```
[martin@cloud Programación]$ tar -cf mysqlapp-0.1.tar mysqlapp/ //creamos el archivo tar
[martin@cloud Programación]$ gzip -9 mysqlapp-0.1.tar  //comprimimos el archivo tar
```

Con lo cual tendremos un archivo **mysqlapp-0.1.tar.gz**, listo para distribuir!
Para instalarlo en la misma máquina u otra basta realizar (como root) donde guardamos el archivo

```
[root@cloud Programación]$ tar -xvzf mysqlapp-0.1.tar.gz
[root@cloud Programación]$ cd mysqlapp/
[root@cloud mysqlapp]$ ./configure
[root@cloud mysqlapp]$ make
[root@cloud mysqlapp]$ make install
```

Si no recibieron ningún mensaje de error la aplicación fue instalada satisfactoriamente, ahora con sólo escribir el nombre del archivo ejecutable en la línea de comando llamaran a la aplicación. Con esto lograron instalar la aplicación!

Apéndice A

MATERIAL DE CONSULTA LENGUAJE C y C++

Variables e identificadores;

La variable es la unidad básica de almacenamiento de los datos, para usar una variable en C o su derivado C++ debe ser declarada con anterioridad, la llamada variable determina el tipo de dato que contendrá, ya que dependiendo el tipo de datos será el espacio de memoria que el compilador le reservara, pero como nosotros esa dirección de memoria no la conocemos hasta que el compilador no la asigne en tiempo de ejecución, para poder trabajar con ella disponemos de los Identificadores, que son nombres para identificar esas variables, en C++ suele ser habital el uso de punteros, un puntero es una variable que apunta a una dirección de memoria, se suele usar para declarar variables a partir de una estructura propia.
En la siguiente tabla se pueden observar los distintos tipos de variables que admite C.

TIPOS	*RANGO*	*TAMAÑO*	*DESCRIPCIÓN*
Char	-128 a 127	1	Para una letra o un dígito.
unsigned char	0 a 255	1	Letra o número positivo.
Int	-32.768 a 32.767	2	Para números enteros.
unsigned int	0 a 65.535	2	Para números enteros.
long int	±2.147.483.647	4	Para números enteros
unsigned long int	0 a 4.294.967.295	4	Para números enteros
float	3.4E-38 decimales(6)	6	Para números con decimales
double	1.7E-308 decimales(10)	8	Para números con decimales
long double	3.4E-4932 decimales(10)	10	Para números con decimales

Los identificadores deben cumplir las siguientes normas:

- La longitud puede ir de un carácter a 31.
- El primero de ellos debe ser siempre una letra.
- No puede contener espacios en blanco, ni acentos y caracteres gramaticales.
- Hay que tener en cuenta que el compilador distingue entre mayúsculas y minúsculas.

LAS CONSTANTES

Son valores fijos que no pueden ser modificados por el programa. Pueden ser de cualquier tipo de datos básicos. Si quisiéramos cambiar el valor de una constante el compilador nos devuelve un error. Las constantes de carácter van encerradas en comillas simples. Las constantes enteras se especifican con números sin parte decimal y las de coma flotante con su parte entera separada por un punto de su parte decimal.

Ejemplos:

```
const int NUMEROENTERO = 21;
const double NUMERODOBLE=25.62;
const char TEXTO = 'Hola';
```

El comando #define se utiliza para definir un identificador y una secuencia de caracteres que sustituirá cada vez que se encuentre en el archivo fuente (*.cpp ó *.h). También pueden ser utilizados para definir valores numéricos constantes.

Ejemplos:

```
#define DB_NOMBRE “personas”
#define DB_YEAR 1981
#define VALOR_USS 2.92
```

OPERADORES

C es un lenguaje muy rico en operadores incorporados, es decir, implementados al realizarse el compilador. Define cuatro tipos de operadores: aritméticos, relacionales, lógicos y a nivel de bits. También se definen operadores para realizar determinadas tareas, como las asignaciones.

Asignación *(=)*: En C se puede utilizar el operador de asignación en cualquier expresión valida. Solo con utilizar un signo de igualdad se realiza la asignación. El operador destino (parte izquierda) debe ser siempre una variable, mientras que en la parte derecha puede ser cualquier expresión valida. Es posible realizar asignaciones múltiples, igualar variables entre si y a un valor.

En las siguientes tablas se vera a continuación los distintos operadores que se pueden utilizar en C / C++

NOTA: Los operadores a nivel de bits no se incluyen en las tablas de la siguiente pagina ya que no lo usaremos en nuestra aplicaciones de C++ con Qt, también se incluyen tablas con los identificadores de formato en caso de que hagan alguna aplicación mediante consola.

OPERADORES ARITMETICOS		
OPERADOR	***DESCRIPCIÓN***	***ORDEN***
-	Restar.	**3**
+	Sumar	**3**
*****	Multiplicar	**2**
/	Dividir	**2**
%	Resto de una división	**2**
-	Signo (monario).	**2**
--	Decremento en 1.	**1**
++	Incrementa en 1.	**1**

OPERADORES RELACIONALES		
OPERADOR	***DESCRIPCIÓN***	***ORDEN***
<	Menor que.	**5**
>	Mayor que.	**5**
<=	Menor o igual.	**5**
>=	Mayor o igual	**5**
= =	Igual	**6**
! =	Distinto	**6**
+	Concatenación de Texto	**-**

OPERADORES LÓGICOS		
OPERADOR	***DESCRIPCIÓN***	***ORDEN***
&&	Y (AND)	**10**
||	O (OR)	**11**
!	NO (NOT)	**1**

Entrada y Salida de Datos en C++

La entrada y salida de datos en C++ con aplicaciones mediante consola se realiza con los comando *cin* >> y *cout* <<(en C se utiliza *printf()* y *scanf()*), sin embargo, cabe destacar que estos son sólo dos comandos de los tantos que se pueden usar para mostrar datos en pantalla. A continuación se adjuntan las tablas con las constantes de carácter y los identificadores de formato admitidos por C / C++.

IDENTIFICADOR DE FORMATO	
IDENTIFICADOR	***DESCRIPCION***
%c	Carácter
%d	Entero
%e	N. Científica
%E	N. Científica
%f	Coma flotante
%o	Octal
%s	Cadena
%u	Sin signo
%x	Hexadecimal
%X	Hexadecimal
%p	Puntero
%ld	Entero Largo
%h	Short
%%	signo %

CONSTANTES DE CARACTER	
CONSTANTE	***DESCRIPCION***
\n	Salto de línea
\f	Salto de página
\r	Retorno de carro
\t	Tabulación
\b	Retroceso
\’	Comilla simple
\”	Comillas
****	Barra invertida
\?	Interrogación

Sentencias de Control

Son distintas maneras que nos ofrece un lenguaje de programación de hacer que nuestro programa se desempeñe o la ejecución del mismo cambie teniendo en cuenta las acciones que haga el usuario.

Las sentencias de control de C y C++ son las siguientes:

IF – ELSE:

Al llegar la ejecución del programa a este punto la futura ejecución del mismo se vera afectado por las variables del programa y las opciones programadas en este bloque.
Su sintaxis de programación es la siguiente:

```
if ( condición)                    //la expresión debe ser booleana (usando operadores aritméticos
{                          //o de expresión)
condición verdadera
}
else
{
condición falsa
}
```

Ejemplo:

```
if ( txtCodigo->text() == “m”)
{
txtDescripcion->setText(“Mecanico”);
}
else
{
 txtDescripcion->setText(“Electrico”);
}
```

O tomando las condiciones al revés:

```
if ( txtCodigo->text() != “m”)
{  txtDescripcion->setText(“Electrico”);
}
else
{ txtDescripcion->setText(“Mecanico”);
}
```

Además se pueden concatenar if las veces que sea necesario como se ve en este ejemplo:

```
if
{   if
       {condicion 1-1}
    else
       {condicion 1-2 }
}
else
{condición  2}
```

SWITCH:

Esta función tiene la propiedad de realizar distintas funciones en base al valor que adopte determinada variable, es una sentencia muy parecida a un grupo de Ifs anidados pero mucho más fácil de comprender. Se debe poner todos los valores que esperamos ingresen y las acciones que cada opción realizara, si los valores son numéricos se ponen sin agregar ningún otro signo, en cambio, si son valores de texto, los mismos van delimitados por la comilla simple (“ ' ”)

Su forma de aplicación es la siguiente:

```
switch ( variable )
{
        case valor_1
                comandos de la opción
                break   //finalización de los comandos de la opción
        case valor_2
                comandos de la opción
                break   //finalización de los comandos de la opción
        case valor_n
                comandos de la opción
                break   //finalización de los comandos de la opción
        default:
}
```

La forma en la que opera es la siguiente: cuando se ejecuta, compara el valor que contiene la variable con cada opción, en caso de que alguna coincida se ejecuta el código que contiene la opción, es posible que en un mismo ciclo la ejecución del programa entre en una o varias opciones si el criterio es compartido, en caso de no corresponder el valor con ninguna de las opciones disponibles, se ejecuta el código de la opción default, en caso de no estar declarado default no se ejecuta nada.

La sentencia break realiza la salida de la ejecución del programa en determinada opción, en el caso de la sentencia switch, ejecuta el código de las opciones y cuando encuentra la sentencia break prosigue con la ejecución del programa.

WHILE:

Esta sentencia ejecuta repetidamente un bloque de código hasta que se cumpla una determinada condición para la finalización. En la sentencia while se encuentra cuatro partes, las cuales son:

1) Inicialización.
2) Cuerpo.
3) Iteración.
4) Terminación.

```
[ inicialización ]
while ( terminación )
{
cuerpo;
[ iteración; ]
}
```

Ejemplo:

```
int contador  = 0 ;
int total;

while ( contador  <= 10 )
        total = total + contador;
contador++;
```

DO WHILE:

La forma de escritura es la misma con la una diferencia que este tipo de bloque se ejecuta aunque sea una única vez, siendo que en el caso anterior es posible que el bloque no se ejecute ni siquiera una sola vez.

```
int contador  = 0 ;
int total;

do
{
total = total + contador;
contador++;
}
while ( total  <  20 );
```

FOR:

Es una sentencia que cumple las mismas operaciones que las sentencias anteriores pero con la ventaja de ser de fácil declaración y en pocas líneas, la salida a este bucle es casi siempre con un valor numérico. La iteración es cualquier expresión matemática valida, la forma de escribirla es la siguiente:

```
for ( iniciación, finalización, iteración )
{
        sentencia1;
        sentencia2;
}

for ( int contador=0, contador  < 10, contador++ )
{
total = total + contador;
}
```

FUNCIONES:

Las funciones son los bloques de código constructores de C / C++, son los lugares donde se realizan todas las actividades del programa. Este concepto es la cualidad mas importante de C / C++, la de dividir en pequeñas tareas en el programa. Con lo cual se podrán ver como pequeños fragmentos de pocas líneas, cuya confección y depuración será una tarea simple. Las funciones pueden o no devolver valores del programa.

El mecanismo para agregar funciones es la de declarar la función antes de usarla, una vez realizada la declaración, hacemos la llamada a esa función.

El ámbito de trabajo de las funciones es privada, El código de una función es privado a esa función, este código esta oculto al resto del programa a menos que se haga a través de una llamada. Todas las variables que se definen dentro de una función son locales con la excepción de las variables estáticas.

Forma de uso:

```
tipo _ devuelto  nombre _ función  ( [parámetros ] );
{
cuerpo;
}
```

Llamada a una función:

```
nombre _ función  ( [parámetros ] );
```

Al llamar a una función se puede hacer la llamada ***por valor*** o ***por referencia***. En el caso de hacerla *por valor* se copia el contenido del argumento al parámetro de la función, es decir si se producen cambios en el parámetro no afecta a los argumentos. C utiliza esta llamada por defecto. Cuando se utiliza la llamada *por referencia* lo que se pasa a la función es la dirección de memoria del argumento, por tanto si se producen cambios estos afectan también al argumento. La llamada a una función se puede hacer tantas veces como se quiera.

PRIMER TIPO: Las funciones de este tipo ni devuelven valor ni se les pasa parámetros. Ente caso hay que indicarle que el valor que devuelve es de tipo void y para indicar que no recibirá parámetros también utilizamos el tipo void. Cuando realizamos la llamada no hace falta indicarle nada, se abren y cierran los paréntesis.

```
void nombre _ función(void);

nombre _ función();
```

SEGUNDO TIPO: Son funciones que devuelven un valor una vez que han terminado de realizar sus operaciones, sólo se puede devolver uno. La devolución se realiza mediante la sentencia **`return`**, que además de devolver un valor hace que la ejecución del programa vuelva al código que llamo a esa función. Al compilador hay que indicarle el tipo de valor que se va a devolver poniendo delante del nombre de la función el tipo a devolver. En este tipo de casos la función es como si fuera una variable, pues toda ella equivale al valor que devuelve.

```
tipo _ devuelto nombre _ función(void);
variable=nombre_funcion();
```

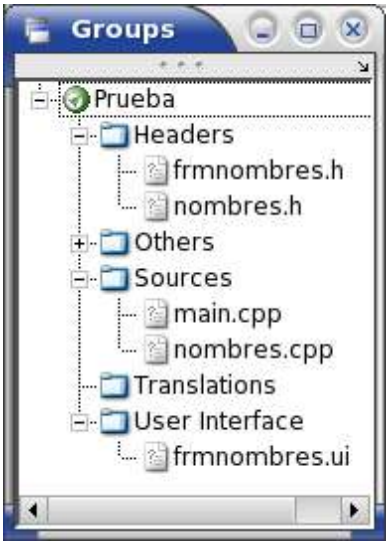
TERCER TIPO: En este tipo las funciones pueden o no devolver valores pero lo importante es que las funciones pueden recibir valores. Hay que indicar al compilador cuantos valores recibe y de que tipo es cada uno de ellos. Se le indica poniéndolo en los paréntesis que tienen las funciones. Deben ser los mismos que en el prototipo.

```
void nombre _ función(tipo1,tipo2…tipoN);
nombre _ función(var1,var2…varN);
```

Apéndice B

CODIGO FUENTE PROGRAMA SALUDAR

Grupo de Archivos:



Interfaz:


```
/***************************************************************************
 main.cpp  -  description
 ----------------
   copyright          : (C) 2004 by Martin Sande
   email              : cloud@argentina.com
***************************************************************************/

#include "nombres.h"              //nombre de la clase creada en Kdevelop
#include "frmnombres.h"           //nombre de la clase creada con uic (ídem form)
#include <qapplication.h>         //si o si base de toda aplicación de Qt

int main( int argc, char ** argv )     //procedimiento principal

 {
   QApplication Main( argc, argv );              //Creo Aplicación de Qt
   Nombres Form;                                 //Creo form a partir de clase
   Form.show();                                  //Muestro form
   Form.setCaption("Nombres para Saludar");      //Asigno caption al form
   return Main.exec();
};
```

```
/***************************************************************************
 nombres.cpp  -  description
 ----------------
   copyright          : (C) 2004 by Martin Sande
   email              :  cloud@argentina.com
***************************************************************************/

#include "nombres.h"
#include "frmnombres.h"
#include "qlineedit.h"
#include "qlabel.h"
#include "qpushbutton.h"
#include "qcombobox.h"

Nombres::Nombres(QWidget *parent, const char *name ) : FrmNombres(parent,name)
{
cmbNombres->insertItem("Mariana");
cmbNombres->insertItem("Martin");
cmbNombres->insertItem("Matias");
cmbNombres->insertItem("Rosa");
cmbNombres->insertItem("Manuel");
}

Nombres::~Nombres()
{}

void Nombres::saludar()
{
lblSaludar->setText(txtSaludar->text());
txtSaludar->clear();
}

void Nombres::saludarcombo()
{
 lblSaludar->setText(cmbNombres->currentText());
}
```

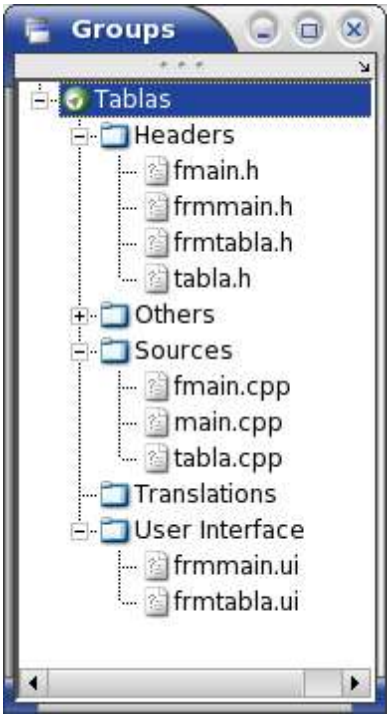
```
/****************************************************************************
 nombres.h  -  description
 -----------------
   copyright          : (C) 2004 by Martín Sande
   email              : cloud@argentina.com
****************************************************************************/

#ifndef NOMBRES_H
#define NOMBRES_H
#include <qwidget.h>
#include <frmnombres.h>

class Nombres : public FrmNombres
{
  Q_OBJECT
public:
        Nombres(QWidget *parent=0, const char *name=0);
        ~Nombres();
public slots:
        virtual void saludar();
        virtual void saludarcombo();
};
#endif



/****************************************************************************
 frmnombres.h  -  description
 -----------------
   copyright        : Form interface generated from reading ui file 'frmnombres.ui'
   by: The User Interface Compiler ($Id: qt/main.cpp   3.1.2
****************************************************************************/

#ifndef FRMNOMBRES_H
#define FRMNOMBRES_H
#include <qvariant.h>
#include <qwidget.h>

class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QLabel;
class QPushButton;
class QLineEdit;
class QComboBox;
```

```
class FrmNombres : public QWidget
{
    Q_OBJECT
public:
    FrmNombres( QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );
    ~FrmNombres();

    QLabel* label1;
    QPushButton* cmdSaludar;
    QLabel* lblSaludar;
    QLabel* label2;
    QLineEdit* txtSaludar;
    QComboBox* cmbNombres;

public slots:
    virtual void saludar();
    virtual void saludarcombo();

protected:

protected slots:

    virtual void languageChange();

};
#endif  // FRMNOMBRES_H
```

Apéndice C

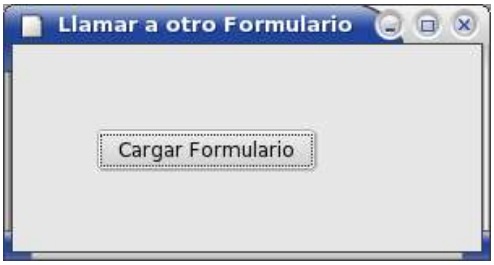
CODIGO FUENTE VARIOS FORMULARIOS (TABLAS)

Grupo de Archivos :



Interface:

FrmMain (fmain.h /fmain.cpp)



FrmTabla (tabla.h / tabla.cpp)


```
/***************************************************************************
 main.cpp  -  description
 ----------------
   copyright          : (C) 2004 by Martin Sande
   email              : cloud@argentina.com
***************************************************************************/

#include "fmain.h"                  //nombre de la clase creada en Kdevelop
#include "frmmain.h"                //nombre de la clase creada con uic (ídem form)
#include <qapplication.h>           //si o si base de toda aplicación de Qt

int main( int argc, char ** argv )  //procedimiento principal
 {
   QApplication Main( argc, argv );          //Creo Aplicación de Qt
   FMain Form;                               //Creo form a partir de clase
```

```
    Form.show();                                        //Muestro form
    Form.setCaption("Llamar a otro Formulario");   //Asigno caption al form
    return Main.exec();
}


/****************************************************************************
 fmain.h  -  description
 -----------------
    copyright          : (C) 2004 by Martin Sande
    email              : cloud@argentina.com
****************************************************************************/

#ifndef FMAIN_H
#define FMAIN_H

#include <qwidget.h>
#include <frmmain.h>

class FMain : public FrmMain  {
   Q_OBJECT
public:
        FMain(QWidget *parent=0, const char *name=0);
        ~FMain();

public slots:
        virtual void mostrarForm( );
};

#endif

/****************************************************************************
 tabla.h  -  description
 -----------------
    copyright          : (C) 2004 by Martin Sande
    email              : cloud@argentina.com
****************************************************************************/

#ifndef TABLA_H
#define TABLA_H

#include <qwidget.h>
#include <frmtabla.h>

class Tabla : public FrmTabla
{
   Q_OBJECT
```

```
private:
int fila;

public:
        Tabla(QWidget *parent=0, const char *name=0);
        ~Tabla();

public slots:
        virtual void pasarPersona();
};

#endif

/***************************************************************************
 fmain.cpp  -  description
 -----------------
   copyright          : (C) 2004 by Martin Sande
   email              : cloud@argentina.com
***************************************************************************/

#include "fmain.h"
#include "qpushbutton.h"
#include "frmtabla.h"
#include "tabla.h"

FMain::FMain(QWidget *parent, const char *name ) : FrmMain(parent,name)
{
}
FMain::~FMain()
{
}

void FMain::mostrarForm()
{
        Tabla *tabla = new Tabla ();
        tabla->show();
}

/***************************************************************************
 tabla.cpp  -  description
 -----------------
   copyright          : (C) 2004 by Martin Sande
   email              : cloud@argentina.com
***************************************************************************/

#include "tabla.h"
#include "qlineedit.h"
```

```
#include "qtable.h"

Tabla::Tabla(QWidget *parent, const char *name ) : FrmTabla(parent,name)
{
fila=0;

QHeader *Titulos = tblNombres->horizontalHeader();

Titulos->setLabel( 0, ("Nombre"));
Titulos->setLabel( 1, ("Apellido"));
Titulos->setLabel( 2, ("Edad"));

Titulos->setMovingEnabled(TRUE);
tblNombres->setReadOnly(TRUE);
}

Tabla::~Tabla()
{
}

void Tabla::pasarPersona()
{
        if ( fila >= tblNombres->numRows() )
        {
                tblNombres->insertRows ( tblNombres->numRows() );
        }

        tblNombres->setText ( fila , 0 , txtNombre->text() );
        tblNombres->setText ( fila , 1 , txtApellido->text() );
        tblNombres->setText ( fila , 2 , txtEdad->text() );

        txtNombre->clear();
        txtApellido->clear();
        txtEdad->clear();

        fila++;
}

/****************************************************************************
 frmmain.cpp  -  description
 ----------------
   copyright         : Form interface generated from reading ui file 'frmmain.ui'
   by: The User Interface Compiler ($Id: qt/main.cpp   3.1.2
****************************************************************************/

#ifndef FRMMAIN_H
#define FRMMAIN_H
```

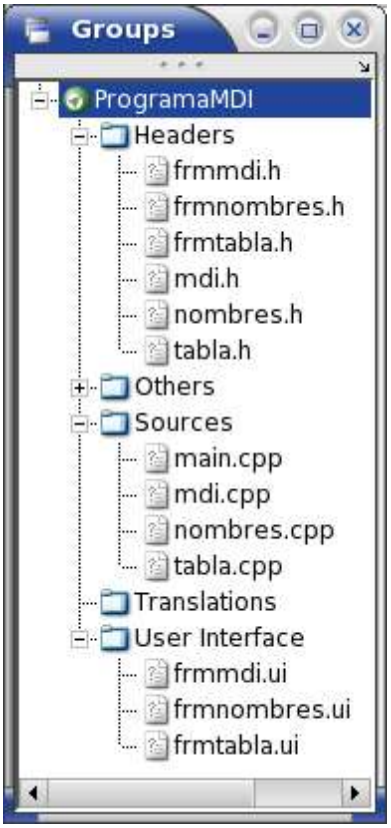
```
#include <qvariant.h>
#include <qwidget.h>

class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QPushButton;

class FrmMain : public QWidget
{
   Q_OBJECT
public:
   FrmMain( QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );
   ~FrmMain();
   QPushButton* cmdMain;

public slots:
   virtual void mostrarForm();
protected:
protected slots:
   virtual void languageChange();

};

#endif // FRMMAIN_H

/****************************************************************************
 frmtabla.h  -  description
 ----------------
   copyright         : Form interface generated from reading ui file 'frmtabla.ui'
   by: The User Interface Compiler ($Id: qt/main.cpp   3.1.2
****************************************************************************/

#ifndef FRMTABLA_H
#define FRMTABLA_H

#include <qvariant.h>
#include <qwidget.h>

class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QLabel;
class QPushButton;
class QLineEdit;
class QTable;
```

```
class FrmTabla : public QWidget
{
    Q_OBJECT
public:
    FrmTabla( QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );
    ~FrmTabla();

    QLabel* textLabel1_3;
    QPushButton* cmdPasar;
    QLabel* textLabel1_2;
    QLabel* textLabel1;
    QLineEdit* txtNombre;
    QLineEdit* txtApellido;
    QLineEdit* txtEdad;
    QTable* tblNombres;

public slots:
    virtual void pasarPersona();

protected:
    QGridLayout* FrmTablaLayout;

protected slots:
    virtual void languageChange();

};

#endif // FRMTABLA_H
```

Apéndice D

CODIGO FUENTE APLICACIÓN MDI

Grupo de Archivos



Interface

Los archivos *frmnombres.h*, *nombres.h*, *nombres.cpp*, *frmtabla.h*, *tabla.h* y *tabla.cpp* se encuentran en el apéndice A y apéndice B respectivamente.

```
/***************************************************************************
 main.cpp  -  description
 ----------------
   copyright          : (C) 2004 by Martin Sande
   email              : cloud@argentina.com
***************************************************************************/

#include "mdi.h"                        // nombre de la clase creada en Kdevelop
#include "frmmdi.h"                     // nombre de la clase creada con uic (idem form)
#include <qapplication.h>               // si o si base de toda aplicación de Qt

int main( int argc, char ** argv )  //procedimiento principal

 {
   QApplication Main( argc, argv );                 // Creo Aplicación de Qt
   MDI *Form = new MDI();                                   // Creo form a partir de clase
   Main.setMainWidget (Form);                       // Asigno Formulario principal
```

```
    Form->show();                                   // Muestro form
    Form->setCaption("Formulario MDI");             // Asigno caption al form

    Main.connect( &Main, SIGNAL(lastWindowClosed()), &Main, SLOT(quit()) );
    int res = Main.exec();
    return res;   // En estas líneas se conecta el formulario para que devuelva un entero.
};

/***************************************************************************
 mdi.h  -  description
 -----------------
    copyright          : (C) 2004 by Martin Sande
    email              : cloud@argentina.com
***************************************************************************/

#ifndef MDI_H
#define MDI_H

#include <qwidget.h>
#include "frmmdi.h"
#include <qworkspace.h>

class QWorkspace;

class MDI : public FrmMDI  {
  Q_OBJECT

private:
        QWorkspace * espacioTrabajo;

public:
        MDI(QWidget *parent=0, const char *name=0);
        ~MDI();

public slots:
    virtual void funcionSalir();
    virtual void funcionDeshacer();
    virtual void funcionCortar();
    virtual void funcionCopiar();
    virtual void funcionPegar();
    virtual void mostrarNombres();
    virtual void mostrarTabla();
};

#endif
```

```
/***************************************************************************
 mdi.cpp  -  description
 ----------------
   copyright          : (C) 2004 by Martin Sande
   email              : cloud@argentina.com
***************************************************************************/

#include "mdi.h"
#include "tabla.h"
#include "nombres.h"
#include <qdockarea.h>
#include <qdockwindow.h>
#include <qvbox.h>
#include <qmessagebox.h>

MDI::MDI(QWidget *parent, const char *name ) : FrmMDI(parent,name)
{
   QVBox* cajaVertical = new QVBox( this );
   cajaVertical->setFrameStyle( QFrame::StyledPanel | QFrame::Sunken );
   espacioTrabajo = new QWorkspace( cajaVertical );
   espacioTrabajo->setScrollBarsEnabled( TRUE );
   setCentralWidget( cajaVertical );

   this->showMaximized();
}

MDI::~MDI()
{
}

void MDI::funcionSalir()
{
        QWidgetList listaVentanas = espacioTrabajo->windowList();
        if ( listaVentanas.count() > 0 )
        {
        QMessageBox::information( 0, "Cerrar Aplicación ",
        "No se puede cerrar la aplicacion.\n"
        "Debe cerrar todas las ventanas." );
        }
        else
        {
        this->close();
        }
}
```

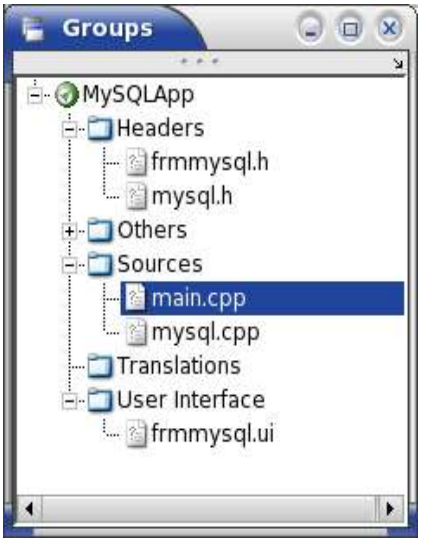
```
void MDI::funcionDeshacer()
{
}

void MDI::funcionCortar()
{
}

void MDI::funcionCopiar()
{
}

void MDI::funcionPegar()
{
}

void MDI::mostrarNombres()
{
        Nombres *nombres = new Nombres (espacioTrabajo);
        nombres->show();
}

void MDI::mostrarTabla()
{
        Tabla *tabla = new Tabla (espacioTrabajo);
        tabla->show();
}

/****************************************************************************
 frmmdi.h  -  description
 -----------------
   copyright         : Form interface generated from reading ui file 'frmmdi.ui'
   by: The User Interface Compiler ($Id: qt/main.cpp   3.1.2
****************************************************************************/

#ifndef FRMMDI_H
#define FRMMDI_H

#include <qvariant.h>
#include <qpixmap.h>
#include <qmainwindow.h>
#include <qworkspace.h>

class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QAction;
```

```
class QActionGroup;
class QToolBar;
class QPopupMenu;
class QWorkspace;

class FrmMDI : public QMainWindow
{
   Q_OBJECT

private:
        QWorkspace * espacioTrabajo;

public:
   FrmMDI( QWidget* parent = 0, const char* name = 0, WFlags fl = WType_TopLevel );
   ~FrmMDI();

   QMenuBar *menu;
   QPopupMenu *mnuArchivo;
   QPopupMenu *mnuEditar;
   QPopupMenu *mnuFormularios;
   QToolBar *toolBar;
   QAction* archivoSalir;
   QAction* editarDeshacer;
   QAction* editarCortar;
   QAction* editarCopiar;
   QAction* editarPegar;
   QAction* formNombres;
   QAction* formtablas;

public slots:
   virtual void funcionSalir();
   virtual void funcionDeshacer();
   virtual void funcionCortar();
   virtual void funcionCopiar();
   virtual void funcionPegar();
   virtual void mostrarNombres();
   virtual void mostrarTabla();

protected:
protected slots:
   virtual void languageChange();

private:
QPixmap image0,  image1,   image2,   image3,   image4,  image5,   image6;
};
#endif // FRMMDI_H
```

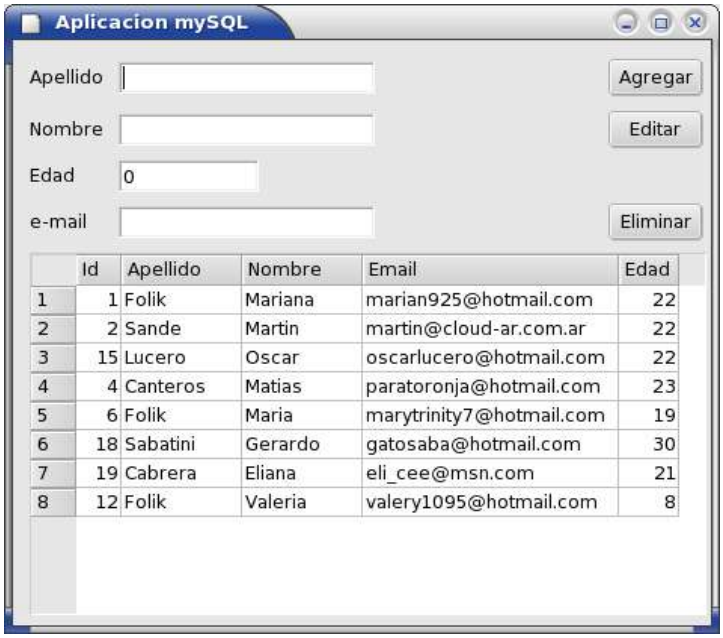
Apéndice E

CODIGO FUENTE APLICACIÓN MySQL

Grupo de Archivos



Interface




```
/***************************************************************************
 main.cpp  -  description
 ----------------
   copyright          : (C) 2004 by Martin Sande
   email              : cloud@argentina.com
***************************************************************************/

#include "mysql.h"
#include "frmmysql.h"
#include <qapplication.h>

int main( int argc, char ** argv )

 {
   QApplication Main( argc, argv );
   MySQL Form;
   Form.show();
   return Main.exec();

};
```

```
/***************************************************************************
 mysql.h  -  description
 -----------------
   copyright          : (C) 2004 by Martin Sande
   email              : cloud@argentina.com
***************************************************************************/

#ifndef MYSQL_H
#define MYSQL_H

#include <qwidget.h>
#include <frmmysql.h>
#include <qevent.h>

class MySQL : public FrmMySQL  {
  Q_OBJECT

private:
        int esNuevo;

public:
        MySQL(QWidget *parent=0, const char *name=0);
        ~MySQL();

protected:
  void closeEvent( QCloseEvent* );

public slots:

   virtual void agregarPersonas();
   virtual void editarPersonas();
   virtual void eliminarPersonas();
   virtual void cargarTabla();
   virtual void cargarPersonas();
};

#endif
```

```
/***************************************************************************
 mysql.cpp  -  description
 -----------------
   copyright          : (C) 2004 by Martin Sande
   email              : cloud@argentina.com
***************************************************************************/

#include "mysql.h"
#include <qsqldatabase.h>
#include <qsqlquery.h>
#include <qheader.h>
#include <qtable.h>
#include <qlineedit.h>
#include <qlabel.h>
#include <qpushbutton.h>
#include <qmessagebox.h>
#include <qevent.h>


MySQL::MySQL(QWidget *parent, const char *name ) : FrmMySQL(parent,name)
{
QSqlDatabase *DBCompaneros = QSqlDatabase::addDatabase ("QMYSQL3");
DBCompaneros->setDatabaseName ("companeros");
DBCompaneros->setUserName ("martin");
DBCompaneros->setPassword ("companeros");
DBCompaneros->setHostName ("localhost");
DBCompaneros->open();
esNuevo=0;
tblPersonas->setReadOnly (TRUE); //seteamos la tabla a solo lectura
MySQL::cargarTabla();
}

MySQL::~MySQL()
{
}

void MySQL::cargarTabla()
{
QSqlDatabase *DBCompaneros = QSqlDatabase::database ("personas");
QSqlQuery Consulta( "SELECT IdPersona, Apellido, Nombre, email, Edad FROM personas
WHERE 1;", DBCompaneros );

tblPersonas->setNumRows (Consulta.size());
tblPersonas->setNumCols (5);
int contador=0;
```

```
//construimos un objeto para los encabezados de las columnas
//y le damos los valores correspondientes.
QHeader *titulosTabla = tblPersonas->horizontalHeader();
titulosTabla->setLabel( 0, ("Id"));
titulosTabla->setLabel( 1, ("Apellido"));
titulosTabla->setLabel( 2, ("Nombre"));
titulosTabla->setLabel( 3, ("Email"));
titulosTabla->setLabel( 4, ("Edad"));

tblPersonas->setColumnWidth( 0, 30 );
tblPersonas->setColumnWidth( 1, 80 );
tblPersonas->setColumnWidth( 2, 80 );
tblPersonas->setColumnWidth( 3, 170 );
tblPersonas->setColumnWidth( 4, 40 );

        while ( Consulta.next() )
        {
                tblPersonas->setText( contador , 0, Consulta.value(0).toString() );
                tblPersonas->setText( contador , 1, Consulta.value(1).toString() );
                tblPersonas->setText( contador , 2, Consulta.value(2).toString() );
                tblPersonas->setText( contador , 3, Consulta.value(3).toString() );
                tblPersonas->setText( contador , 4, Consulta.value(4).toString() );
                contador++;
        }
}

void MySQL::agregarPersonas()
{
txtApellido->clear();
txtNombre->clear();
txtEdad->setText(0);
txtEmail->clear();
txtApellido->setFocus();
cmdEditar->setText("Guardar");
esNuevo=1;
}

void MySQL::editarPersonas()
{
QSqlDatabase *DBCompaneros = QSqlDatabase::database ("personas");
QSqlQuery agregarRegistro ( "SELECT * FROM personas where 1" , DBCompaneros);

if ( esNuevo==1 )
{
switch ( QMessageBox::information ( 0, "Agregar Registro",
```

```
                                "Esta a punto de insertar el siguiente registro: \n\n"
                                + txtApellido->text() + ", "
                                + txtNombre->text() + "\nEmail: "
                                + txtEmail->text() + ".\nEdad: "
                                + txtEdad->text()  + ".\n\n\Desea Continuar?",
                                QMessageBox::Yes,  QMessageBox::No )  )
        {

        case QMessageBox::Yes:
                agregarRegistro.exec ( "INSERT INTO personas ( Apellido, Nombre,
        email, Edad ) VALUES ( '" + txtApellido->text() + "', '" +
        txtNombre->text() + "', '" + txtEmail->text()  + "', " + txtEdad->text() + " );" );
                break;
                }

        }
else
        {
        switch ( QMessageBox::information ( 0, "Actualizacion de Registro",
                        "Esta a punto de actualizar el siguiente registro: \n\n"
                        + tblPersonas->text( tblPersonas->currentRow(), 1 ) + ", "
                        + tblPersonas->text( tblPersonas->currentRow(), 2 ) + "\nEmail: "
                        + tblPersonas->text( tblPersonas->currentRow(), 3 ) + ".\nEdad: "
                        + tblPersonas->text( tblPersonas->currentRow(), 4 ) + ".\nA:\n"
                        + txtApellido->text() + ", " + txtNombre->text() + "\nEmail: "
                        + txtEmail->text() + ".\nEdad: " + txtEdad->text()  + ".\n\n\Desea
                        Continuar?",
                        QMessageBox::Yes,  QMessageBox::No )  )
        {
        case QMessageBox::Yes:
                        agregarRegistro.exec (" UPDATE personas SET Apellido = '" +
                txtApellido->text() + "', " + " Nombre ='" + txtNombre->text() + "', " +
                        " email ='" + txtEmail->text() + "', " + " Edad =" + txtEdad->text() +
                        " WHERE IdPersona =" +
                tblPersonas->text( tblPersonas->currentRow(), 0 ) +";" );
        break;
        }
}
MySQL::cargarTabla();
cmdEditar->setText("Editar");
txtApellido->clear();
txtNombre->clear();
txtEdad->setText("0");
txtEmail->clear();
txtApellido->setFocus();
esNuevo=0;
}
```

```
void MySQL::eliminarPersonas()
{
switch ( QMessageBox::information ( 0, "Eliminación de Registro",
                        "Esta a punto de eliminar el siguiente registro: \n\n"
                        + tblPersonas->text( tblPersonas->currentRow(), 1 ) + ", "
                        + tblPersonas->text( tblPersonas->currentRow(), 2 ) + "\nEmail: "
                        + tblPersonas->text( tblPersonas->currentRow(), 3 ) + ".\nEdad: "
                        + tblPersonas->text( tblPersonas->currentRow(), 4 ) + ".\n",
                        QMessageBox::Yes,  QMessageBox::No )  )
        {
        case QMessageBox::Yes:
                QSqlDatabase *DBCompaneros = QSqlDatabase::database ("personas");
                QSqlQuery borrarRegistro ("DELETE FROM personas WHERE IdPersona ='"
                + tblPersonas->text( tblPersonas->currentRow(), 0 ) + "';", DBCompaneros);
                MySQL::cargarTabla();
                break;
        }
}

void MySQL::cargarPersonas()
{
QSqlDatabase *DBCompaneros = QSqlDatabase::database ("personas");
QSqlQuery cargarRegistro ( "SELECT Apellido, Nombre, email, Edad FROM personas
                        where IdPersona ='" + tblPersonas->text( tblPersonas->currentRow(),
                0 ) + "';" , DBCompaneros);
cargarRegistro.first();
txtApellido->setText ( cargarRegistro.value(0).toString() );
txtNombre->setText (cargarRegistro.value(1).toString() );
txtEmail->setText (cargarRegistro.value(2).toString() );
txtEdad->setText (cargarRegistro.value(3).toString() );
cmdEditar->setText("Guardar");
}

void MySQL::closeEvent ( QCloseEvent* ventana )
{
switch ( QMessageBox::information ( 0, "Salir del Programa",
                "Esta seguro que desea Salir del Programa?\n",
                QMessageBox::Yes , QMessageBox::No ) )
        {
        case QMessageBox::Yes:
                        ventana->accept();
                        break;
        case QMessageBox::No:
                        ventana->ignore();
                        break;
        }
}
```

Apéndice F

LICENCIA PÚBLICA GENERAL (GPL)

Esta es una traducción NO oficial de la "GNU General Public License" al español. No fué publicada por la "FSF Free Software Foundation", y no respalda legalmente los términos de distribución del software que utiliza la "GNU GPL", sólo el texto original en inglés lo hace. Sin embargo esperamos que esta traducción ayude a las personas de habla hispana a entender mejor la "GPL".

PREAMBULO

Los contratos de licencia de la mayor parte del software están diseñados para quitarle su libertad de compartir y modificar dicho software. En contraste, la "GNU General Public License" pretende garantizar su libertad de compartir y modificar el software "libre", esto es para asegurar que el software es libre para todos sus usuarios. Esta licencia pública general se aplica a la mayoría del software de la "FSF Free Software Foundation" (Fundación para el Software Libre) y a cualquier otro programa de software cuyos autores así lo establecen. Algunos otros programas de software de la Free Software Foundation están cubiertos por la "LGPL Library General Public License" (Licencia Pública General para Librerías), la cual puede aplicar a sus programas también.

Cuando hablamos de software libre, nos referimos a libertad, no precio. Nuestras licencias "General Public Licenses" están diseñadas para asegurar que:

1. Usted tiene la libertad de distribuir copias del software libre (y cobrar por ese sencillo servicio si así lo desea)
2. Recibir el código fuente (o tener la posibilidad de obtenerlo si así lo desea)
3. Que usted puede modificar el software o utilizar partes de el en nuevos programas de software libre
4. Que usted esté enterado de que tiene la posibilidad de hacer todas estas cosas.

Para proteger sus derechos, necesitamos hacer restricciones que prohiban a cualquiera denegarle estos derechos o a pedirle que renuncie a ellos. Estas restricciones se traducen en algunas responsabilidades para usted si distribuye copias del software, o si lo modifica.

Por ejemplo, si usted distribuye copias de un programa, ya sea gratuitamente o por algún importe, usted debe dar al que recibe el software todos los derechos que usted tiene sobre el

mismo. Debe asegurarse también que reciban el código fuente o bién que puedan obtenerlo si lo desean. Y por último debe mostrarle a esa persona estos términos para que conozca los derechos de que goza.

Nosotros protegemos sus derechos en 2 pasos: (1) protegiendo los derechos de autor del software y (2) ofreciendole este contrato de licencia que le otorga permiso legal para copiar, distribuir y modificar el software.

Además, para la protección de los autores de software y la nuestra, queremos asegurarnos de que toda persona entienda que no existe ninguna garantía del software libre. Si el software es modificado por alguien y lo distribuye, queremos que quienes lo reciban sepan que la copia que obtuvieron no es la original, por lo que cualquier problema provocado por quien realizó la modificación no afectará la reputación del autor original.

Finalmente, cualquier programa de software libre es constantemente amenazado por las patentes de software. Deseamos evadir el peligro de que los re-distribuidores de un programa de software libre obtengan individualmente los derechos de patente con el fin de volver dicho programa propietario. Para prevenir esto, hemos dejado en claro que cualquier patente deberá ser licenciada para el uso libre de toda persona o que no esté licenciada del todo.

A continuación se describen con precisión los términos y condiciones para copiar, distribuir y modificar el software.

TÉRMINOS Y CONDICIONES PARA COPIA, MODIFICACION Y DISTRIBUCIÓN

0. Esta licencia aplica a cualquier programa o trabajo que contenga una nota puesta por el propietario de los derechos del trabajo estableciendo que su trabajo puede ser distribuido bajo los términos de esta "GPL General Public License". El "Programa", utilizado en lo subsecuente, se refiere a cualquier programa o trabajo original, y el "trabajo basado en el Programa" significa ya sea el Programa o cualquier trabajo derivado del mismo bajo la ley de derechos de autor: es decir, un trabajo que contenga el Programa o alguna porción de el, ya sea íntegra o con modificaciones y/o traducciones a otros idiomas. De aquí en adelante "traducción" estará incluida (pero no limitada a) en el término "modificación", y la persona a la que se aplique esta licencia será llamado "usted".

Otras actividades que no sean copia, distribución o modificación no están cubiertas en esta licencia y están fuera de su alcance. El acto de ejecutar el programa no está restringido, y la salida de información del programa está cubierta sólo si su contenido constituye un trabajo basado en el Programa (es independiente de si fue resultado de ejecutar el programa). Si esto es cierto o no depende de la función del programa.

1. Usted puede copiar y distribuir copias fieles del código fuente del programa tal como lo recibió, en cualquier medio, siempre que proporcione de manera conciente y apropiada una nota de derechos de autor y una declaración de no garantía, además de mantener intactas todas las notas que se refieran a esta licencia y a la ausencia de garantía, y que le proporcione a las demás personas que reciban el programa una copia de esta licencia junto con el Programa.

Usted puede aplicar un cargo por el acto físico de transferir una copia, y ofrecer protección de garantía por una cuota, lo cual no compromete a que el autor original del Programa responda por tal efecto.

2. Usted puede modificar su copia del Programa o de cualquier parte de el, formando así un trabajo basado en el Programa, y copiar y distribuir tales modificaciones o bién trabajar bajo los términos de la sección 1 arriba descrita, siempre que cumpla con las siguientes condiciones:

1. Usted debe incluir en los archivos modificados notas declarando que modificó dichos archivos y la fecha de los cambios.
2. Usted debe notificar que ese trabajo que distribuye contiene totalmente o en partes al Programa, y que debe ser licenciado como un conjunto sin cargo alguno a cualquier otra persona que reciba sus modificaciones bajo los términos de esta Licencia.
3. Si el programa modificado lee normalmente comandos interactivamente cuando es ejecutado, usted debe presentar un aviso, cuando el programa inicie su ejecución en ese modo interactivo de la forma más ordinaria, que contenga una noticia de derechos de autor y un aviso de que no existe garantía alguna (o que sí existe si es que usted la proporciona) y que los usuarios pueden redistribuir el programa bajo esas condiciones, e informando al usuario como puede ver una copia de esta Licencia. (Excepción: si el programa en sí es interactivo pero normalmente no muestra notas, su trabajo basado en el Programa no tiene la obligación de mostrar tales notas)

Estos requerimientos aplican al trabajo modificado como un todo. Si existen secciones identificables de tal trabajo que no son derivadas del Programa original, y pueden ser razonablemente consideradas trabajos separados e independientes como tal, entonces esta Licencia y sus términos no aplican a dichas secciones cuando usted las distribuye como trabajos separados. Pero cuando usted distribuye las mismas secciones como parte de un todo que es un trabajo basado en el Programa, la distribución del conjunto debe ser bajo los términos de esta Licencia, cuyos permisos para otras personas que obtengan el software se extienden para todo el software, así como para cada parte de el, independientemente de quién lo escribió.

No es la intención de esta sección de reclamar derechos o pelear sus derechos sobre trabajos hechos enteramente por usted, en lugar de eso, la intención es ejercer el derecho de controlar la distribución de los trabajos derivados o colectivos basados en el Programa.

Adicionalmente, el simple agregado de otro trabajo NO basado en el Programa al Programa en cuestión (o a un trabajo basado en el Programa) en algún medio de almacenamiento no pone el otro trabajo bajo el alcance de esta Licencia.

3. Usted puede copiar y distribuir el Programa (o un trabajo basado en él, bajo la Sección 2) en código objeto o en forma de ejecutable trajo los términos de las secciones 1 y 2 arriba descritas siempre que cumpla los siguientes requisitos:

1. Acompañarlo con el correspondiente código fuente legible por la máquina, que debe ser distribuido bajo los términos de las secciones 1 y 2 y en un medio comúnmente utilizado para el intercambio de software, o

2. Acompañarlo con una oferta escrita, válida por al menos 3 años y para cualquier persona, por un cargo no mayor al costo que conlleve la distribución física del código fuente correspondiente en un medio comúnmente utilizado para el intercambio de software, o
3. Acompañarlo con la información que usted recibió sobre la oferta de distribución del código fuente correspondiente. (Esta alternativa está permitida sólo para distribución no-comercial y sólo si usted recibió el Programa en código objeto o en forma de ejecutable con tal oferta de acuerdo a la subsección b anterior)

El código fuente de un trabajo significa la forma preferida de hacer modificaciones al mismo. Para un trabajo ejecutable, un código fuente completo significa todo el código fuente de todos los módulos que contiene, mas cualquier archivo de definición de interfases, mas los programas utilizados para controlar la compilación y la instalación del ejecutable.

Sin embargo, como excepción especial, no se requiere que el código fuente distribuído incluya cualquier cosa que no sea normalmente distribuída con las componentes mayores (compilador, kernel, etc.) del sistema operativo en el cual el ejecutable corre, a menos de que una componente en particular acompañe al ejecutable.

Si la distribución del ejecutable o del código objeto se hace ofreciendo acceso a copiar desde un lugar designado, entonces el ofrecer acceso equivalente para copiar el código fuente desde el mismo lugar se considera distribución del código fuente, aunque las demás personas no copien el código fuente junto con el código objeto.

4. Usted no puede copiar, modificar, sub-licenciar ni distribuir el Programa a menos que sea expresamente bajo esta Licencia, de otra forma cualquier intento de copiar, modificar, sub-licenciar o distribuir el programa es nulo, y automáticamente causará la pérdida de sus derechos bajo esta Licencia. Sin embargo, cualquier persona que haya recibido copias o derechos de usted bajo esta Licencia no verán terminadas sus Licencias ni sus derechos perdidos mientras ellas continúen cumpliendo los términos de esta Licencia.

5. Usted no está obligado a aceptar esta Licencia, dado que no la ha firmado. Sin embargo, nada le otorga el permiso de modificar o distribuir el Programa ni sus trabajos derivados. Estas acciones están prohibidas por la ley si usted no acepta esta Licencia. Sin embargo, modificando o distribuyendo el Programa (o cualquier trabajo basado en el Programa) indica su aceptación de esta Licencia y de todos sus términos y condiciones para copiar, distribuir o modificar el Programa y/o trabajos basados en el.

6. Cada vez que usted redistribuye el Programa (o cualquier trabajo basado en el Programa), la persona que lo recibe automáticamente recibe una licencia del autor original para copiar, distribuir o modificar el Programa sujeto a estos términos y condiciones. Usted no puede imponer ninguna restricción adicional a las personas que reciban el Programa sobre los derechos que en esta Licencia se les otorga. Usted no es responsable de forzar a terceras personas en el cumplimiento de esta Licencia.

7. Si como consecuencia de un veredicto de un juzgado o por el alegato de infringir una patente o por cualquier otra razón (no limitado solo a cuestiones de patentes) se imponen condiciones sobre usted que contradigan los términos y condiciones de esta Licencia, éstas no le excusan de

los términos y condiciones aquí descritos. Si usted no puede distribuir el producto cumpliendo totalmente con las obligaciones concernientes a la resolución oficial y al mismo tiempo con las obligaciones que se describen en este contrato de Licencia, entonces no podrá distribuir más este producto. Por ejemplo, si una licencia de patente no permitirá la distribución del Programa de forma libre de regalías (sin pago de regalías) por parte de quienes lo reciban directa o indirectamente, entonces la única forma de cumplir con ambas obligaciones es renunciar a la distribución del mismo.

Si cualquier parte de esta sección resulta inválida, inaplicable o no obligatoria bajo cualquier circunstancia en particular, la tendencia de esta es a aplicarse, y la sección completa se aplicará bajo otras circunstancias.

La intención de esta sección no es la de inducirlo a infringir ninguna ley de patentes, ni tampoco infringir algún reclamo de derechos, ni discutir la validez de tales reclamos; esta sección tiene el único propósito de proteger la integridad del sistema de distribución del software libre, que está implementado por prácticas de licencia pública. Mucha gente ha hecho generosas contribuciones a la amplia gama de software distribuido bajo este sistema favoreciendo así la constante aplicación de este sistema de distribución; es decisión del autor/donador si su Programa será distribuído utilizando este u otro sistema de distribución, y la persona que recibe el software no puede obligarlo a hacer ninguna elección en particular.

Esta sección pretende dejar muy en claro lo que se cree que será una consecuencia del resto de esta Licencia.

8. Si la distribución y/o el uso del Programa se restringe a algunos países ya sea por patentes, interfases protegidas por derechos de autor, el propietario original de los derechos de autor que ubica su Programa bajo esta Licencia puede agregar una restricción geográfica de distribución explícita excluyendo los países que aplique, dando como resultado que su distribución sólo se permita en los países no excluídos. En tal caso, esta Licencia incorpora la limitación como si hubiera sido escrita en el cuerpo de esta misma Licencia.

9. La "FSF Free Software Foundation" puede publicar versiones nuevas o revisadas de la "GPL General Public License" de uno a otro momento. Estas nuevas versiones mantendrán el espíritu de la presente versión, pero pueden diferir en la inclusión de nuevos problemas o en la manera de tocar los problemas o aspectos ya presentes.

Cada versión tendrá un número de versión que la distinga. Si el Programa especifica un número de versión para esta Licencia que aplique a él y "cualquier versión subsecuente", usted tiene la opción de seguir los términos y condiciones de dicha versión o de cualquiera de las posteriores versiones publicadas por la "FSF". Si el programa no especifica una versión en especial de esta Licencia, usted puede elegir entre cualquiera de las versiones que han sido publicadas por la "FSF".

10. Si usted desea incorporar partes del Programa en otros Programas de software libre cuyas condiciones de distribución sean distintas, deberá escribir al autor solicitando su autorización. Para programas de software protegidas por la "FSF Free Software Foundation", deberá escribir a la "FSF" solicitando autorización, en ocasiones hacemos excepciones. Nuestra decisión será

guiada por dos metas principales:

* mantener el estado de libertad de todos los derivados de nuestro software libre
* promover el uso comunitario y compartido del software en general

NO EXISTE GARANTIA ALGUNA

11. DEBIDO A QUE EL PROGRAMA SE OTORGA LIBRE DE CARGOS Y REGALÍAS, NO EXISTE NINGUNA GARANTÍA PARA EL MISMO HASTA DONDE LO PERMITA LA LEY APLICABLE. A EXCEPCIÓN DE QUE SE INDIQUE OTRA COSA, LOS PROPIETARIOS DE LOS DERECHOS DE AUTOR PROPORCIONAN EL PROGRAMA "COMO ES" SIN NINGUNA GARANTÍA DE NINGÚN TIPO, YA SEA EXPLICITA O IMPLÍCITA, INCLUYENDO, PERO NO LIMITADA A, LAS GARANTÍAS QUE IMPLICA EL MERCADEO Y EJERCICIO DE UN PROPÓSITO EN PARTICULAR. CUALQUIER RIESGO DEBIDO A LA CALIDAD Y DESEMPEÑO DEL PROGRAMA ES TOMADO COMPLETAMENTE POR USTED. SI EL SOFTWARE MUESTRA ALGÚN DEFECTO, USTED CUBRIRÁ LOS COSTOS DE CUALQUIER SERVICIO, REPARACIÓN O CORRECCIÓN DE SUS EQUIPOS Y/O SOFTWARE QUE REQUIERA.

12. EN NINGÚN CASO NI BAJO NINGUNA CIRCUNSTANCIA EXCEPTO BAJO SOLICITUD DE LA LEY O DE COMÚN ACUERDO POR ESCRITO, NINGÚN PROPIETARIO DE LOS DERECHOS DE AUTOR NI TERCERAS PERSONAS QUE PUDIERAN MODIFICAR Y/O REDISTRIBUIR EL PROGRAMA COMO SE PERMITE ARRIBA, SERÁN RESPONSABLES DE LOS DAÑOS CORRESPONDIENTES AL USO O IMPOSIBILIDAD DE USAR EL PROGRAMA, SIN IMPORTAR SI SON DAÑOS GENERALES, ESPECIALES, INCIDENTALES O CONSECUENTES CORRESPONDIENTES AL USO O IMPOSIBILIDAD DE USAR EL PROGRAMA (INCLUYENDO PERO NO LIMITADO A LA PERDIDA DE INFORMACIÓN O DETERIORO DE LA MISMA AFECTÁNDOLO A USTED, A TERCERAS PERSONAS QUE SEA POR FALLAS EN LA OPERACIÓN DEL PROGRAMA O SU INTERACCIÓN CON OTROS PROGRAMAS) INCLUSIVE SI TAL PROPIETARIO U OTRAS PERSONAS HAYAN SIDO NOTIFICADAS DE TALES FALLAS Y DE LA POSIBILIDAD DE TALES DAÑOS.

FIN DE TÉRMINOS Y CONDICIONES

Cómo aplicar estos términos a sus nuevos programas?

Si usted desarrolla un nuevo Programa y desea que sea lo más público posible, el mejor modo de hacerlo es haciéndolo Software Libre donde toda persona lo puede redistribuir y cambiar bajo estos términos.

Para hacer esto, agregue las siguientes notas al programa. Es más seguro agregarlas al inicio de cada archivo del código fuente para notificar de manera más efectiva la ausencia de garantía; y cada archivo debe de contener al menos la línea de "Copyright" o derechos de autor y una referencia de donde se puede encontrar la nota completa.

Ejemplo:

Esta línea que contenga el nombre del programa y una idea de lo que hace.
Copyright (C) AÑO nombre del autor

Este programa es Software Libre; usted puede redistribuirlo
y/o modificarlo bajo los términos de la "GNU General Public
License" como lo publica la "FSF Free Software Foundation",
o (a su elección) de cualquier versión posterior.

Este programa es distribuido con la esperanza de que le será
útil, pero SIN NINGUNA GARANTÍA; incluso sin la garantía
implícita por el MERCADEO o EJERCICIO DE ALGÚN PROPOSITO en
particular. Vea la "GNU General Public License" para más
detalles.

Usted debe haber recibido una copia de la "GNU General Public
License" junto con este programa, si no, escriba a la "FSF
Free Software Foundation, Inc.", 59 Temple Place - Suite 330,
Boston, MA 02111-1307, USA.

Adicionalmente agregue información de cómo contactarle por correo electrónico y convencional.

Si el programa es interactivo, ponga en la salida del programa una nota corta al iniciar el modo interactivo:

Gnomovision versión 69, Copyright (C) AÑO nombre del autor
Gnomovision no tiene NINGUNA GARANTIA, para más detalles
escriba 'show w'. Este es Software Libre, y usted está
permitido para redistribuirlo bajo ciertas condiciones;
escriba 'show c' para más detalles.

Estos supuestos comandos 'show w' y 'show c' deberán mostrar las partes apropiadas de la "GPL General Public License". Por supuesto, los comandos que utilice pueden ser distintos, pueden ser incluso "clicks" del ratón, opciones de menús etc., lo más apropiado para su programa.

Usted debería hacer que su jefe de proyecto (si trabaja como programador) o su escuela, si aplica, firme una "declaración de derechos de autor" para el programa, si se necesita. Aquí hay un ejemplo, modifique los nombres:

Yoyodyne, Inc., aquí vienen las declaraciones de derechos de autor
interés en el programa 'Gnomovision'
(lo que make pasa al compilador)
escrito por James Hacker.
firma de Ty Coon, 1 de Abril 1989
Ty Coon, Presidente

BIBLIOGRAFIA:

- Manual de Lenguaje C
  Ignacion Sanchez Gines, 2000
  www.planetiso.cjb.net

- Aprenda C++ como si estuviera en primero
  Escuela Superior de Ingenieros Industriales
  Universidad de Navarra

- Qt Designer and Kdevelop 3 for Begginers
  Anne-Marie Mahouf (annma@kde.org)
  www.kdevelop.org

- SQL Module
  Documentación Qt 3.1.2

- Documentación Qt 3.1.2