



Universidad Nacional Autónoma de Nicaragua



*Universidad Nacional Autónoma de Nicaragua*  
*UNAN – LEON*  
*Facultad de Ciencias*  
*Ingeniería en Sistemas de Información*



*Trabajo Monográfico para optar al título de Ingeniero en Sistema de Información.*

*Tema: Desarrollo de aplicaciones gráficas de usuario utilizando Qt y C++ con acceso a bases de datos MySql.*

*Integrantes:*

*Darling María Sevilla Jiménez*  
*Armando René Rodríguez López*  
*Mauricio Antonio Pérez Alonso*

*Tutor: Msc. Máximo Guido*

*Leon, 16 de Noviembre de 2007.*

*“A LA LIBERTAD POR LA UNIVERSIDAD”*

---

## Índice

Contenido	Página(s)
Agradecimiento	
Dedicatorias	
Tema	
I Introducción.....	1
II Antecedentes.....	2
III Justificación.....	5
IV Objetivos.....	6
4.1 Generales	
4.2 Específicos	
V Marco teórico	
5.1 Descripción general de Qt.....	7
5.2 ¿Porqué MySQL? .....	16
5.2.1 Definiciones.....	17
5.2.2 Cliente Mysql.....	30
5.2.3 Lenguaje SQL.....	33
5.2.3.1 Creación de bases de datos y tablas.....	33
5.2.3.2 Inserción y modificación de datos.....	37
5.2.3.3 Selección de datos.....	39
VI Metodología.....	41
6.1 Recopilación de la información.....	41
6.2 Especificación de Requisitos.....	41
6.3 Plataforma a Utilizar.....	41
6.4 Instalación del sistema OpenSuSe 10.2.....	42
6.5 Instalación y configuración de Mysql.....	42
6.6 Instalación y configuración del creador de interfaces gráficas Qt..	42
6.7 Personalización.....	42
6.8 Explotación y Mantenimiento.....	42
6.9 Resultados.....	43
VII Diagrama E-R.....	44
VIII Conclusiones.....	45

---



IX Recomendaciones.....	46
X Recursos disponibles.....	47
XI Bibliografía.....	48
XII Anexos.....	49



## Agradecimientos

Al finalizar nuestro trabajo de Investigación, como forma de culminar nuestros estudios profesionales, damos nuestro agradecimiento a:

Dios nuestro padre celestial por proveernos la vida y de todo lo necesario para cumplir esta meta.

Nuestros padres por habernos apoyado moral, emocional y económicamente a lo largo de nuestro desarrollo académico.

Los profesores por brindarnos los conocimientos básicos para emprender nuestro futuro profesional.

Nuestro tutor quien siempre estuvo presente en la evolución de este trabajo monográfico.

Todos aquellos que colaboraron, de una u otra forma, para la realización de nuestro trabajo.

Gracias.

---



## Dedicatoria

El presente trabajo Monográfico lo dedico a:

*Dios nuestro* señor por haberme permitido concluir mis estudios e iluminarme en todo momento y por ser fuente de inspiración y sabiduría.

*La Virgen Maria*, por su intercepción y cubrirme con su inmenso amor.

*Mi madre Marta Delia Alonso Flores y Abuela Margarita Flores Urbina* por el apoyo incondicional y el sacrificio que me han brindado en todo momento para seguir adelante y lograr alcanzar mis metas y ser un hombre de bien en servicio de la sociedad.

Dedicatoria realizada por: Br. Mauricio Antonio Pérez Alonso

---



## Dedicatoria

El presente trabajo Monográfico lo dedico a:

*Dios nuestro señor* por brindarme la vida, la inteligencia, el discernimiento y todo lo necesario para lograr culminar satisfactoriamente mis metas.

*A mis padres Noel Armando Rodríguez Gámez y Martha Eugenia López Calderón* por proveerme su apoyo incondicional para poder desarrollarme y crecer como persona y como profesional.

*A mi novia Meyling Yunieth Silva Rivera* por estar siempre a mi lado apoyándome en todo.

*A mis hermanos Cesar Vladimir Rodríguez López y Gabriela Eugenia Rodríguez López* por ayudarme y cuidarme cuando lo he necesitado.

Dedicatoria realizada por: Br. Armando René Rodríguez López

---



## Dedicatoria

El presente trabajo Monográfico lo dedico a:

*Dios nuestro padre Celestial* por brindarme la vida y la fuerza necesaria para alcanzar esta meta.

*Mis padres Juan Sevilla y Maura Cecilia Jiménez* por haberme dado todo su amor y apoyo incondicional.

*Mis hermanas Meyling Sevilla y Rebeca Sevilla* por estar a mi lado siempre.

*Mi esposo Abelardo José Martínez* por el apoyo que me ha brindado.

*Mi futuro baby* por ser el motivo de inspiración para finalizar esta etapa de mi vida.

Dedicatoria realizada por: Bra. Darling Maria Sevilla Jimenez

---



## TEMA

Desarrollo de aplicaciones gráficas de usuario utilizando Qt y C++ con acceso a bases de datos MySQL bajo plataforma Linux.

---



## I Introducción

En este trabajo monográfico presentamos el uso del diseñador de interfaces gráficas Qt, por ser de fácil manejo, posee disponibilidad del código fuente y por estar escrito en lenguaje C++, es altamente portable.

En Función de este entorno de desarrollo QT proponemos una aplicación cliente – servidor haciendo uso del gestor de bases de datos MySQL que sirve de intérprete entre la aplicación y el usuario con este damos un aporte a los programadores y diseñadores de sistemas un ejemplo claro y útil para la creación de interfaces gráficas con acceso a bases de datos que muestra el funcionamiento de la potente herramienta Qt.

Además, demostramos las cualidades que posee el creador de interfaces gráficas Qt así como sus facilidades de uso e interacción con MySQL mediante la creación de la aplicación *“Control del Stock de vehículos en almacén”*. En esta aplicación se implementa el uso de bases de datos que tendrá acceso sólo el cliente (usuario autorizado) mediante formularios generados por los elementos de interfaz gráfica Qt (botones, etiquetas, etc) que le permitirá acceder de manera segura, ya que el usuario debe identificarse con su contraseña correspondiente.

Esta aplicación le facilita al usuario la manipulación de los datos y a su vez se encarga de actualizar la base datos que le mantiene un sistema seguro, consistente y evitando la redundancia de los datos.

## **II Antecedentes**

El entorno de desarrollo Qt es muy reciente, pues, la primera versión del juego de herramientas Qt estuvo disponible en mayo de 1995. Fue desarrollado por Haavard Nord (CEO de Trolltech) y Eirik Chambe-Eng (presidente de Trolltech). Haavard y Eirik quienes se encontraron en el Instituto noruego de la Tecnología en Trondheim, Noruega, donde ambos se graduaron con los grados de master en ciencias de computación.

En 1991, Haavard comenzó a escribir las clases que finalmente se convirtieron en Qt, con la colaboración de Eirik Chambe-Eng en el diseño. Quien en el año siguiente, tuvo la idea de “señales y ranuras (slots)”, Un paradigma de programación GUI simple pero poderoso. Haavard Nord tomó la idea y realizó la codificación manualmente. Hacia 1993, Haavard y Eirik habían desarrollado el primer kernel gráfico de Qt y eran capaces de poner en práctica sus propios widgets (elementos gráficos como botones, etiquetas, etc.). Al final del año, Haavard sugirió que entraran en un negocio para construir “El mejor juego de herramientas C++ GUI del mundo”.

Ellos eligieron ‘Q’ como el prefijo de clase porque la letra les pareció hermosa en la fuente Haavard de Emacs. El ‘t’ fue añadido para significar “el juego de herramientas”, inspirado por “Xt”, el Juego de herramientas X. La compañía fue incorporada el 4 de marzo de 1994, al principio como “Quasar Technologies”, luego como “Troll Tech”, y hoy como “Trolltech”. En abril de 1995, gracias a un contacto hecho por un profesor de la Universidad de Haavard, la compañía noruega Metis les dio un contrato para desarrollar software basado en Qt. Alrededor de este tiempo, Trolltech contrató Arnt Gulbrandsen, quien ideó y puso en práctica un sistema de documentación ingenioso así como contribución al código de Qt.

El 20 de mayo de 1995, Qt 0,90 fue publicado en *sunsite.unc.edu*. Seis días después, la liberación fue anunciada en *comp.os.linux.announce*. Esta fue la primera liberación pública de Qt. Podía ser usado por ambos desarrolladores, tanto Windows como Unix, ofreciendo la misma API (Application Program Interface, o programa de aplicación de interfaz) en ambas plataformas, Qt estaba disponible desde el primer día bajo dos licencias: Una licencia comercial requerida para desarrollo comercial y una edición de software libre para desarrolladores de código abierto, El contrato con la empresa Metis mantuvo Trolltech a flote, mientras por 10 largos meses nadie más compró una licencia comercial.

En marzo de 1996, la Agencia Espacial Europea se convirtió en el segundo comprador, con una compra de 10 licencias comerciales, Eirik y Haavard adquirieron otro desarrollador. Qt 0,97 que fue liberado al final de mayo y el 24 de septiembre de 1996 salió Qt 1.0 al final de año Qt alcanzó la versión 1.1; ocho compradores, cada uno en diferentes países, compraron 18 licencias entre ellos. Ese mismo año también se vio la fundación del proyecto KDE dirigido por Matthias Ettrich.

Qt 1.2 fue liberada en abril de 1997. Matthias Ettrich tomó la decisión de usar Qt para construir KDE, ayudó a Qt a convertirse inmediatamente el estándar de desarrollo C++ GUI en Linux. Qt 1.3 fue liberado en septiembre de 1997.

Matthias se afilió a Trolltech en 1998, y la última versión mayor de Qt 1 fue publicado en septiembre de ese mismo año. Qt 2.0 fue publicado en junio de 1999. Qt 2 tuvo mayores cambios arquitectónicos y fue un producto mucho más fuerte y más maduro que su predecesor. También presentó 40 nuevas clases y soporte Unicode. Qt 2 tuvo una nueva licencia de código abierto, la Licencia Pública Q (QPL), que se basó en la definición de código abierto. En agosto de 1999, Qt ganó el premio LinuxWorld a la mejor librería de herramienta. Cerca de este tiempo Trolltech Pty Ltd (Australia) fue establecida.



Trolltech publicó Qt/Embedded en el 2000. Fue diseñado para correr en dispositivos Linux empujados y proveía su propio sistema de ventanas GPL como un reemplazo ligero para X11. Ambos Qt/Embedded y Qt/X11 fueron entonces ofertadas bajo la ampliamente usada GNU Licencia Pública General (GPL) y también bajo la licencia comercial. A fines del 2000 Trolltech estableció Trolltech Inc. (USA) y fue liberada la primera versión de Qtopia, un entorno para dispositivos portátiles. Qt/Embedded ganó premio LinuxWorld “La Mejor Solución Empotrada Linux” dos años consecutivos 2001 y 2002, Qt 3,0 fue publicado en el 2001. Qt estaba disponible ahora en Windows, Unix, Linux, Linux Empotrado y Mac OS X. Qt 3.0 proporcionó 42 nuevas clases y el código sobrepasaba las 500,000 líneas. Qt 3.0 ganó el “Premio de Productividad Jolt” de Tiempos de Desarrollo de Software en el 2002.

Las ventas de Trolltech se han doblado año con año desde el nacimiento de la compañía. Este éxito es un reflejo de la calidad de Qt y de lo agradable que es su uso. La mayor parte de la existencia de la compañía, ventas y mercadotecnia fueron manejadas por solo un par de personas. Aún, en menos de una década, Qt pasó de ser un producto “secreto”, conocido solo por un grupo selecto de profesionales, a tener cientos de compradores y decenas de cientos de desarrolladores de código abierto todos alrededor de mundo.

Anteriormente en esta Universidad se utilizó la herramienta Qt para la elaboración de un Chat, la cual fue elaborada por los actuales Ing. Valeria Medina y el Ing. Elder.

### **III Justificación**

El motivo que nos impulsó a realizar este trabajo monográfico fue la necesidad de conocer un entorno de desarrollo gráfico que nos permitiese realizar aplicaciones en poco tiempo manteniendo siempre un nivel profesional utilizando la plataforma Linux ya que es software libre, debido a esto elegimos Qt como herramienta para la creación de nuestra interfaz, ya que el [API](#) de la biblioteca cuenta con [métodos](#) para acceder a bases de datos mediante [SQL](#), así como uso de [XML](#) y una multitud de otros lenguajes para el manejo de ficheros, además de estructuras de datos tradicionales.

Consideramos este trabajo será de mucha utilidad a futuros programadores de C++ y a nosotros mismos, ya que Qt posee un entorno gráfico muy fácil de manipular y utiliza lenguaje de programación C++, uno de los lenguajes más potentes y portables, que permite ser implementado en cualquier plataforma.

Además, estamos muy familiarizados con este lenguaje ya que fue impartido en nuestra carrera como uno de los lenguajes más importantes de programación.

## **IV OBJETIVOS**

### **4.1 Objetivos Generales:**

- Implementar Qt como alternativa para el desarrollo de aplicaciones gráficas de usuario bajo plataforma Linux.
- Desarrollar una aplicación como ejemplo usando el gestor de base de datos MySQL en conjunto con Qt bajo plataforma Linux.

### **4.2 Objetivos Específicos:**

- Aportar conocimientos generales acerca del creador de interfaces gráficas Qt el cual utiliza lenguaje de programación C++ y otros elementos importantes en el desarrollo de aplicaciones que manipulan bases de datos.
- Mostrar un ejemplo útil y claro por medio de una aplicación cliente – servidor utilizando el creador de interfaces gráficas Qt y el gestor de bases de datos MySQL.
- Utilizar las cualidades que posee el creador de interfaces gráficas Qt, así como sus facilidades de uso.

## V MARCO TEORICO

### 5.1 Descripción general de Qt

Qt es un entorno de desarrollo y paquetes de herramientas C++ para la programación de interfaces gráficas, desarrollado por Trolltech, y utilizado como base para el desarrollo del entorno integrado de escritorio [KDE](#). Utiliza el lenguaje de programación [C++](#) de forma nativa y además existen bindings para [C](#), [Python \(PyQt\)](#), [Java \(Qt Jambi\)](#), [Perl \(PerlQt\)](#), [Ruby \(QtRuby\)](#), [PHP \(PHP-Qt\)](#) y [Mono \(Qyoto\)](#) entre otros. Está disponible para plataformas Linux/Unix, Windows y MacOSX.

#### QWidget

Un widget es todo lo que aparece en la pantalla, un botón, una ventana, una barra de desplazamiento, un menú, etc.

Algunos widgets contienen otros widgets dentro. Desde el punto de vista de un desarrollador se puede pensar que cada uno de los widgets especializados heredan de una clase base widget que es la clase QWidget. La clase de QWidget es la clase baja de todos los objetos de interfaz de usuario.

El widget es el átomo de la interfaz del usuario: recibe eventos de ratón, teclado y otros del sistema, y pinturas, una representación de sí mismo en la pantalla.

Cada widget es por lo general rectangular, y ellos se ordenan en un Z-orden. Un widget se sujeta por su padre y por el widgets delante de él.

QWidget tiene muchas funciones miembros, pero algunos de ellos tienen funcionalidad pequeña: por ejemplo, QWidget tiene una propiedad del conjunto de caracteres, pero nunca lo usa. Hay muchas subclases que proporcionan la funcionalidad real, como QPushButton, QListBox y QTabDialog, etc.

Un widget que no es incluido en un widget del padre se llama un widget cima-nivelado. Normalmente, los widgets cima-nivelados son las ventanas con un marco y un título obstrúyete (aunque también es posible crear el widgets cima-nivelado sin la tal decoración si se usan las banderas del widget convenientes).

En el entorno de desarrollo Qt, están QMainWindow y varias subclasses de QDialog que son las ventanas de cima-nivel más comunes.

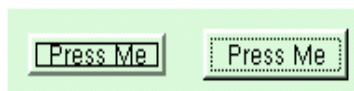
Además, se presentan funciones estáticas para los casos más comunes entre ellos están:

**Widgets básicos:** Estos Widgets básicos se diseñan para el uso directo. Hay, también, algunos widgets de clasificación abstracta diseñados para derivarse y crear widgets más complejos.

**QPushButton:** El botón de orden o comando, es probablemente el widget más usado en cualquier interfaz gráfica de usuario. El botón se utiliza para ordenar a la computadora realizar alguna acción o para contestar una pregunta. Los botones típicos son OK, Apply, Cancel, Close, Yes, No y Help. Un QPushButton es rectangular y típicamente la etiqueta del texto describe su acción. Un carácter subrayado en la etiqueta (precediéndolo con un ampersand en el texto) indica una llave del acelerador, por ejemplo:

```
QPushButton *pb = new QPushButton("&Download", this);
```

En este ejemplo el acelerador es Alt+D, y el texto de la etiqueta se desplegará, según como transmita. Estos pueden desplegar una etiqueta de texto o una imagen, y opcionalmente un icono pequeño. También proporcionan señales menos usada por ejemplo pressed () and released ().

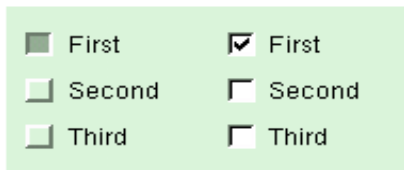




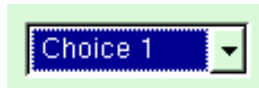
**QRadioButton:** Botón de radio con una etiqueta de texto, es un botón de opción, es decir ellos pueden activarse o desactivarse, este proporciona una de muchas opciones. En un grupo de botones (QButtonGroup), en un momento puede verificarse, si el usuario selecciona otro botón, el botón previamente seleccionado se apaga, ósea se desactiva, es decir, sólo un botón de radio en un momento puede verificarse.



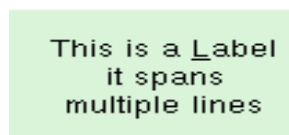
**QCheckBox:** Es un Checkbox con una etiqueta de texto. QCheckBox es un botón de opción, es decir ellos pueden activarse o desactivarse, las cajas de CheckBox definen muchas de muchas opciones. En un grupo de botones (QButtonGroup), en un momento puede verificarse, si el usuario selecciona otro botón, el botón previamente seleccionado queda activo, ya que este se puede seleccionar varios a la misma vez.



**QComboBox:** Es la combinación de un botón y una lista popup. Muestra el elemento actual seleccionado de la lista y puede desplegar la lista de los elementos con un menú pop up,



**QLabel:** Permite el despliegue de texto o de una imagen. Ninguna funcionalidad de interacción de usuario se proporciona por defecto, pero sí se puede derivar para crear un nuevo widget que interactúe con el usuario. La apariencia visual de la etiqueta puede configurarse de varias maneras, y puede usarse para especificar una llave de acelerador de enfoque para otro widget.

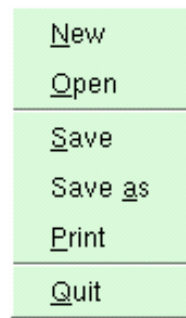




**QLineEdit:** Es un editor de texto de una línea. Permite al usuario modificar su contenido que es texto plano, además brinda las opciones de deshacer, rehacer, cortar, copiar y pegar.



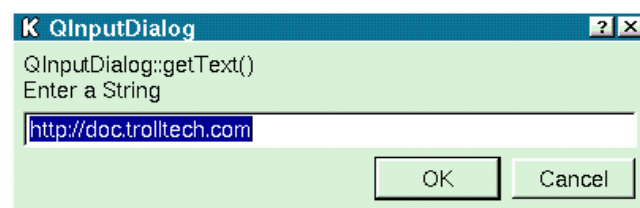
**QPopupMenu:** Es un widget considerado para generar todo tipo de menús por ejemplo menús contextuales o menús desplegables.

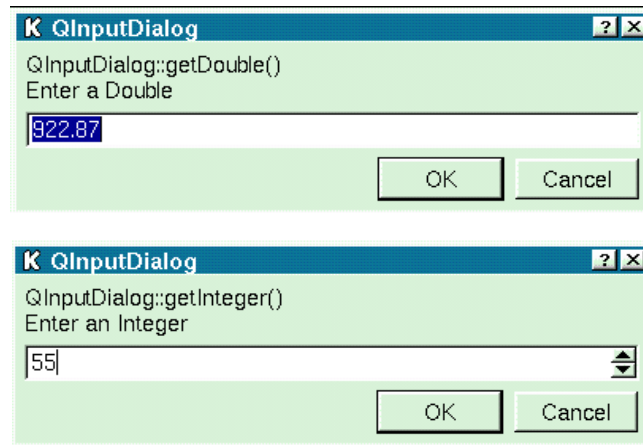


**QTextEdit:** Poderoso editor de texto (Varias líneas). Que es prácticamente igual al QLineEdit pero este puede mostrar imágenes y texto con formato.

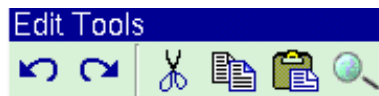
**QLayout:** La disposición (layout) de los widgets es la forma en la que se posicionan los widgets en la pantalla, generalmente se posicionan de unos respecto a otros, de tal forma que uno estará bajo el otro y/o al lado del otro, etc. Existen layout horizontal, vertical y en forma de grid.

**QInputDialog:** La clase de QInputDialog proporciona una caja de diálogo para recibir un solo valor del usuario. El valor de la entrada puede ser una cadena, un número o un artículo de una lista. Una etiqueta debe ponerse para decirle al usuario lo en que ellos deben insertar.





**ToolButton:** La clase `QToolButton` proporciona un botón de acceso rápido a órdenes u opciones, normalmente usadas dentro de un `QToolBar`. Como opuesto a un botón del orden normal, un botón de herramienta no muestra normalmente una etiqueta del texto, pero, en cambio muestras un icono. Su uso clásico es seleccionar las herramientas, por ejemplo la " herramienta de la pluma " en un programa de dibujo. Esto se llevaría a cabo con un `QToolButton` como el botón de la barra traviesa.



**QsqlDatabase:** La clase `QSqlDatabase` se usa para crear las conexiones y proporcionar el manejo de transacciones a bases de datos SQL (el manejo de transacciones depende del SGBD que utilizemos).

**QSqlquery:** Esta clase proporciona un medio de ejecutar y manipular las consultas SQL. Puede usarse para ejecutar DML (el idioma de manipulación de datos) las declaraciones, por ejemplo, `SELECT`, `INSERT`, `UPDATE` y `DELETE`, y también DDL (el idioma de definición de datos) las declaraciones, por ejemplo `CREATE TABLE`. También puede usarse para ejecutar órdenes base de datos específicos que no son SQL normales (por ejemplo `SET DATESTYLE=ISO` para PostgreSQL).

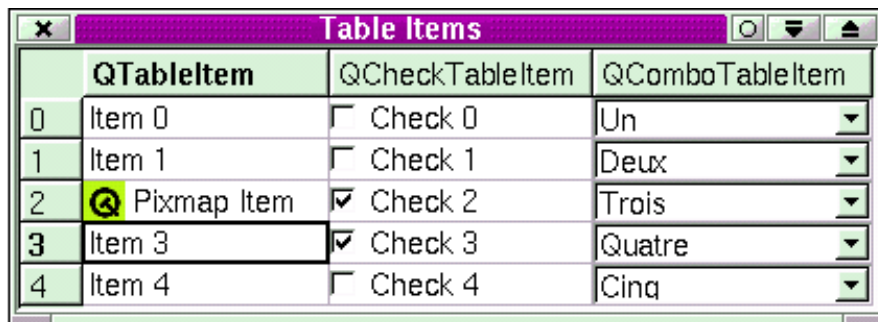
Esta clase tiene las funciones siguientes para recorrer la base de datos: next (), prev (), first (), last (), seek (int).

**Por ejemplo:**

```

QSqlquery query ("SELECT nombre FROM usuarios");
While (query. next ())
{
    QString nombre = query.value (0).toString ();
    doSomething (nombre);
}
    
```

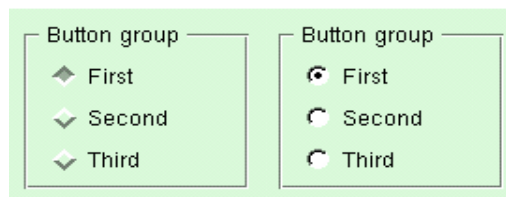
**QTable:** Proporciona un widget de tabla editable flexible, es fácil de usar, aunque tiene un API grande debido a la funcionalidad comprensiva que proporciona. QTable incluye las funciones para manipular títulos (cabeceras), filas y columnas, celdas y selecciones.



**QGroupBox:** El widget QGroupBox proporciona un título a un marco de la caja de grupo. Una caja de grupo proporciona un marco, un título y un atajo del teclado, y despliega varios widgets dentro de sí mismo. El título está en la cima, el atajo del teclado mueve el enfoque del teclado a uno del widgets del hijo de la caja de grupo, y los widgets del hijo, normalmente, se ponen horizontalmente (o verticalmente) dentro del marco.

**QPushButtonGroup:** Un QPushButtonGroup puede usarse para agrupar botones CheckBox y PushButton visualmente. Podemos crear grupos exclusivos, es decir, que un solo botón puede activarse a la vez, por defecto los grupos son no-exclusivo. Todos los botones de radio que se insertan en un grupo de botones son mutuamente exclusivos aún cuando el grupo del botón es no - exclusivo. A los botones se asignan identificadores 0, 1, 2, etc., en el orden ellos se crean. Un QPushButtonGroup puede desplegar un marco y un título porque hereda QGroupBox.

Un botón puede quitarse del grupo con remove (). El id de un botón se obtiene usando id (). Un botón puede ponerse adelante con el setButton (). El número de botones en el grupo es retornado por la función count ().



**QMessageBox:** Proporciona un mensaje corto, un icono, y algunos botones a un diálogo modal. Se usan las cajas de mensaje para proporcionar los mensajes informativos y hacer las preguntas simples.

QMessageBox proporciona un rango de mensajes diferentes:

- **La pregunta (Question)**

Son cajas de mensaje que hacen una pregunta como la parte de funcionamiento normal. Algunas guías de estilo recomiendan usar la información para este propósito.

- **La información (Information)**

Son cajas de mensaje que son parte de funcionamiento normal.





- **Advertencia (Warning)**

Son cajas de mensaje que le dicen al usuario sobre los errores extraños.

- **Crítico (Critical)**

Son cajas de mensaje que, le dicen al usuario sobre los errores críticos.

Las cajas de mensajes tienen un icono diferente para cada uno de los tipos

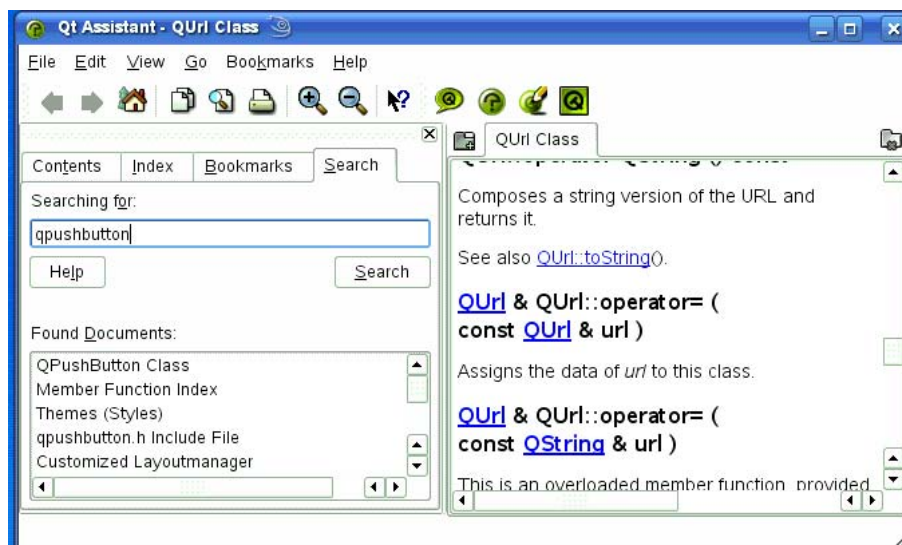
- Question 
- Information 
- Warning 
- Critical 

La complejidad es un botón (OK) para los mensajes simples, o dos o incluso tres botones para las preguntas.

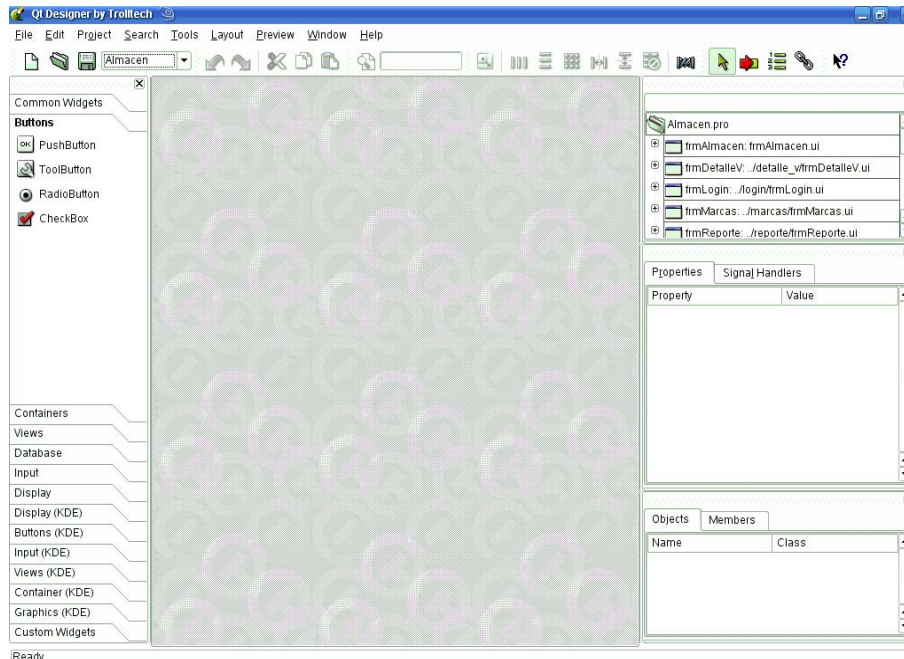
Qt cuenta actualmente con un sistema de doble licencia: una [GPL](#) para el desarrollo de software de código abierto ([open source](#)) y [software libre](#), y otra de pago para el desarrollo de aplicaciones comerciales.

Qt introduce un asistente (QtAssistant), una herramienta para presentar la documentación en línea. Se puede acceder a esta documentación mediante QtAssistant, o con un navegador Web.

La siguiente figura nos muestra la ventana que se abre cuando accedemos a QtAssistant:



Un elemento muy importante para el desarrollo de interfaces gráficas con Qt es: QtDesigner que nos permite crear con facilidad la interfaz gráfica así como elaborar el código necesario para que funcione nuestra aplicación. A continuación presentamos la imagen del QtDesigner utilizado por nosotros.



## Plataformas

Qt se encuentra disponible para las siguientes plataformas:

- [X11](#) - Para [X Window System](#) con licencia [GPL](#).
- [Mac](#) - Para [Mac OS X](#) bajo la licencia [GPL](#).
- [Windows](#) - Para sistemas Windows con licencia [GPL](#) (las antiguas versiones, anteriores a la 4.X no eran libres para este sistema operativo).
- [PDA](#) - Para dispositivos **empotrados**, también con licencia [GPL](#) y generalmente distribuido junto con [Qtopia](#), un entorno completo para [PDAs](#)

Actualmente también está disponible QSA (Qt Scripts for Applications), que, basándose en [ECMAScript/JavaScript](#), permite introducir y crear [scripts](#) en las aplicaciones creadas con Qt.





## 5.2 ¿Por qué MySQL?

Se maneja que para acceder a bases de datos es mucho más útil usar un motor o servidor que hace las funciones de intérprete entre las aplicaciones y usuarios con las bases de datos.

Esta utilidad se traduce en ventajas, entre las que podemos mencionar las siguientes:

- **Acceso a las bases de datos** de forma simultánea por varios usuarios y/o aplicaciones.
- **Seguridad**, en forma de permisos y privilegios, determinados usuarios tendrán permiso para consulta o modificación de determinadas tablas. Esto permite compartir datos sin que peligre la integridad de la base de datos o proteja determinados contenidos.
- **Potencia:** SQL es un lenguaje muy potente para consulta de bases de datos, usar un motor nos ahorra una enorme cantidad de trabajo.
- **Portabilidad:** SQL es también un lenguaje estandarizado, de modo que las consultas hechas usando SQL son fácilmente portables a otros sistemas y plataformas. Esto, unido al uso de C/C++ proporciona una portabilidad enorme.

En concreto, usar MySQL tiene ventajas adicionales:

- **Escalabilidad:** es posible manipular bases de datos enormes, del orden de seis mil tablas y alrededor de cincuenta millones de registros, y hasta 32 índices por tabla.
- **MySQL está escrito en C y C++** y probado con multitud de compiladores y dispone de APIs para muchas plataformas diferentes.
- **Conectividad:** es decir, permite conexiones entre diferentes máquinas con distintos sistemas operativos. Es corriente que servidores Linux o Unix, usando MySQL, sirvan datos para ordenadores con Windows, Linux, Solaris, etc. Para ello se usa TCP/IP, tuberías, o sockets Unix.





- **Es multihilo**, con lo que puede beneficiarse de sistemas multiprocesador.
- Permite manejar multitud de tipos para columnas.
- Permite manejar registros de longitud fija o variable.

### 5.2.1 Definiciones

*¿Qué son las bases de datos?* Vamos a explicar unas bases generales, aunque sólidas, suficientes para desarrollar la mayor parte de los pequeños y medianos proyectos. Hay que señalar que existen varios modelos lógicos de bases de datos, aunque sólo veremos el modelo de **bases de datos relacionales**.

**Dato:** Podemos decir que un dato es una información que refleja el valor de una característica de un objeto real, sea concreto o abstracto, o imaginario. Debe cumplir algunas condiciones, por ejemplo, debe permanecer en el tiempo. En ese sentido, estrictamente hablando, una edad no es un dato, ya que varía con el tiempo. El dato sería la fecha de nacimiento, y la edad se calcula a partir de ese dato y de la fecha actual. Además, debe tener un significado, y debe ser manipulable mediante operadores: comparaciones, sumas, restas, etc. (por supuesto, no todos los datos admiten todos los operadores).

**Base de datos:** Podemos considerar que es un conjunto de datos de varios tipos, organizados e interrelacionados. Estos datos deben estar libres de redundancias innecesarias y ser independientes de los programas que los usan.

**SGBD (DBMS):** Son las siglas que significan Sistema de Gestión de Bases de Datos, en inglés **DBMS**, Data Base Manager System. En este caso, **MySql** es un **SGBD**, o mejor dicho: nuestro **SGBD**.



**Consulta:** Es una petición al SGBD para que procese un determinado comando SQL. Esto incluye tanto peticiones de datos como creación de bases de datos, tablas, modificaciones, inserciones, etc.

**Redundancia de datos:** Hay redundancia de datos cuando la misma información es almacenada varias veces en la misma base de datos. Esto es siempre algo a evitar, la redundancia dificulta la tarea de modificación de datos, y es el motivo más frecuente de inconsistencia de datos. Además, requiere un mayor espacio de almacenamiento, que influye en mayor coste y mayor tiempo de acceso a los datos.

**Inconsistencia de datos:** Sólo se produce cuando existe redundancia de datos. La inconsistencia se refiere a que no todas las copias redundantes contienen la misma información. Así, si existen diferentes modos de obtener la misma información, y esas formas pueden conducir a datos almacenados en distintos sitios. El problema surge al modificar esa información, si sólo cambiamos esos valores en algunos de los lugares en que se guardan, las consultas que hagamos más tarde podrán dar como resultado respuestas inconsistentes (es decir, diferentes). Puede darse el caso de que dos aplicaciones diferentes proporcionen resultados distintos para el mismo dato.

**Integridad de datos:** Cuando se trabaja con bases de datos, generalmente los datos se reparten entre varios ficheros. Así como pasa con MySQL, la base de datos está disponible para varios usuarios de forma simultánea, deben existir mecanismos que aseguren que las interrelaciones entre registros se mantengan coherentes, es decir, que se respetan las dependencias de existencia y que las claves únicas no se repitan.



### ***Diseño de bases de datos: El Modelo conceptual y El modelo Entidad- Relación.***

Siempre que emprendamos un nuevo proyecto, grande o pequeño, antes de lanzarnos a escribir código, crear tablas o bases de datos, hay que analizar el problema sobre el papel. En el caso de las bases de datos, pensaremos sobre qué tipo de información necesitamos guardar, o lo que es más importante: qué tipo de información necesitaremos obtener de la base de datos. En esto consiste el modelado de bases de datos.

#### **Modelado de bases de datos**

El proceso de trasladar un problema del mundo real a un ordenador, usando bases de datos, se denomina **modelado**. Para el modelado de bases de datos es necesario seguir un procedimiento determinado, seguir todo el proceso nos facilitará una documentación necesaria para revisar o mantener la aplicación, ya sea por nosotros mismos o por otros administradores o programadores.

La primera fase del diseño de una aplicación (la base de datos, generalmente, es parte de una aplicación), consiste en hablar con el cliente para saber qué quiere, y qué necesita realmente. Los modelos conceptuales ayudan en esta fase del proyecto, ya que facilitan una forma clara de ver el proceso en su totalidad, puesto que se trata de una representación gráfica.

Los modelos conceptuales no están orientados a ningún sistema físico concreto: tipo de ordenador, sistema operativo, SGBD, etc. Además de consultar con el cliente, una buena técnica consiste en observar el funcionamiento del proceso que se quiere informatizar o modelar. Con las bases de datos lo más importante es observar qué tipo de información se necesita, y que parte de ella se necesita con mayor frecuencia. Una vez recogidos los datos, el siguiente paso es crear un modelo conceptual. El modelo más usado en bases de datos es el **Modelo Entidad-Relación**.



La siguiente fase es convertir el modelo conceptual en un modelo lógico. Existen varios modelos lógicos, pero el más usado, el que mejor se adapta a **MySql** y el que por lo tanto el que utilizaremos, es el **modelo Relacional**.

La conversión entre el modelo conceptual y el lógico es algo bastante mecánico, aunque no por ello será siempre sencillo.

El modelo lógico relacional, consta de un proceso que sirve para verificar que hemos aplicado bien el modelo, y en caso contrario, corregirlo para que sea así. Este proceso se llama normalización, y también es bastante mecánico.

La última fase consiste en codificar el modelo lógico en un modelo físico. Este proceso está ligado al DBMS elegido, en el caso del DBMS que ocuparemos es MySQL, que vamos a manejar, para esto se requiere el conocimiento del lenguaje de consulta SQL.

## **Modelo Entidad-Relación**

En esencia, el modelo entidad-relación (en adelante E-R), consiste en buscar las entidades que describan los objetos que intervienen en el problema y las relaciones entre esas entidades. Todo esto se plasma en un esquema gráfico que tiene por objeto, por una parte, ayudar al programador durante la codificación, y por otra, al usuario a comprender el problema y el funcionamiento del programa.

**Entidad:** Estamos hablando del modelo Entidad-Relación, por lo tanto este es un concepto que no podemos dejar sin definir. Es una representación de un objeto individual concreto del mundo real.



**Conjunto de entidades:** Es la clase o tipo al que pertenecen entidades con características comunes.

**Atributo:** Es cada una de las características que posee una entidad, y que agrupadas permiten distinguirla de otras entidades del mismo conjunto.

**Dominio:** Es conjunto de valores posibles para un atributo.

**Relación:** El otro concepto que no podemos dejar de definir es el de relación. Aunque en realidad, salvo para nombrar el modelo, usaremos el término interrelación, ya que relación tiene un significado radicalmente diferente dentro del modelo relacional, y esto nos puede llevar a error.

**Interrelación:** Es la asociación o conexión entre conjuntos de entidades.

**Grado:** Es el número de conjuntos de entidades que intervienen en una interrelación, existen interrelaciones de grado 2, 3, 4, etc. Pero las más frecuentes son las interrelaciones binarias.

**Clave:** Estaremos de acuerdo en que es muy importante poder identificar claramente cada entidad y cada interrelación. Esto es necesario para poder referirnos a cada elemento de un conjunto de entidades o interrelaciones, ya sea para consultarlo, modificarlo o borrarlo. No deben existir ambigüedades en ese sentido. Es un conjunto de atributos que identifican de forma unívoca una entidad.

**Claves candidatas:** Una característica que debemos buscar siempre en las claves es que contengan el número mínimo de atributos, siempre que mantengan su función. Diremos que una clave es mínima cuando si se elimina cualquiera de los atributos que la componen, deja de ser clave.



Si en una entidad existe más de una de estas claves mínimas, cada una de ellas es una clave candidata. Es cada una de las claves mínimas existente en un conjunto de entidades.

**Clave principal:** Si disponemos de varias claves candidatas no usaremos cualquiera de ellas según la ocasión. Esto sería fuente de errores, de modo que siempre usaremos la misma clave candidata para identificar la entidad. Es una clave candidata elegida de forma arbitraria, que usaremos siempre para identificar una entidad.

**Claves de interrelaciones:** Para identificar interrelaciones el proceso es similar, aunque más simple. Tengamos en cuenta que para definir una interrelación usaremos las claves primarias de las entidades interrelacionadas.

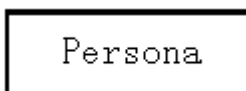
**Entidades fuertes y débiles:** Las entidades débiles no necesitan una clave primaria, sus claves siempre están formadas como la combinación de una clave primaria de una entidad fuerte y otros atributos y las entidades fuertes siempre tienen su clave primaria.

## Representación de Entidades y Relaciones:

### Diagramas

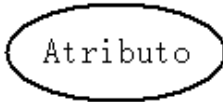
Un diagrama E-R consiste en representar mediante figuras un modelo completo del problema, proceso o realidad a describir, de forma que se definan tanto las entidades que lo componen, como las interrelaciones que existen entre ellas.

**Entidad:** Las entidades se representan con un rectángulo, y en su interior el nombre de la entidad:



### **Atributo**

Los atributos se representan mediante elipses, y en su interior el nombre del atributo:



Algunas variantes de diagramas E-R usan algunas marcas para indicar que cierto atributo es una clave primaria, como subrayar el nombre del atributo.



También es frecuente usar una doble elipse para indicar atributos multivaluados:

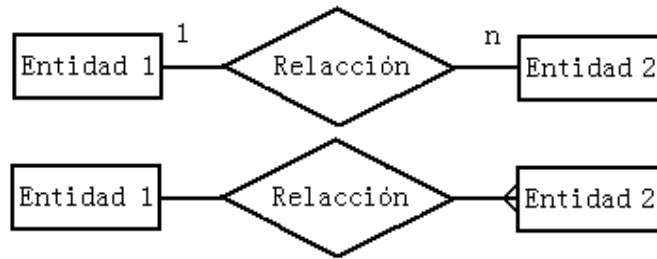


**Atributo multivaluado:** (o multivalorado) se dice del atributo tal que para una misma entidad puede tomar varios valores diferentes, es decir, varios valores del mismo dominio.

**Interrelación:** Las interrelaciones se representan mediante rombos, y en su interior el nombre de la interrelación:



En los extremos de las líneas que parten del rombo se añaden unos números que indican la cantidad de entidades que intervienen en la interrelación: **1, n**. Esto también se suele hacer modificando el extremo de las líneas. Si terminan con un extremo involucran a una entidad, si terminan en varios extremos, (generalmente tres), involucrarán a varias entidades:



Sobre las líneas a veces se añade el rol que representa cada entidad.

**Dominio:** A veces es conveniente añadir información sobre el dominio de un atributo, los dominios se representan mediante hexágonos, con la descripción del dominio en su interior.



### Diseño de bases de datos: El modelo lógico y El modelo relacional

**Modelo relacional:** Entre los modelos lógicos, el modelo relacional está considerado como el más simple. El paso del modelo E-R al relacional es muy simple, y por otra, MySQL, como implementación de SQL, está orientado principalmente a bases de datos relacionales.

El modelo se compone de tres partes:

- **Estructura de datos:** básicamente se compone de relaciones.
- **Manipulación de datos:** un conjunto de operadores para recuperar, derivar o modificar los datos almacenados.
- **Integridad de datos:** una colección de reglas que definen la consistencia de la base de datos.





**Relación:** Es un conjunto de datos referentes a un conjunto de entidades y organizados en forma tabular, que se compone de filas y columnas, (tuplas y atributos), en la que cada intersección de fila y columna contiene un valor.

**Tupla:** Cada una de las filas de una relación. Contiene la información relativa a una única entidad. De esta definición se deduce que no pueden existir dos tuplas iguales en la misma relación.

**Nulo (NULL):** Valor asignado a un atributo que indica que no contiene ninguno de los valores del dominio de dicho atributo.

**Modelo relacional:** Es un modelo basado en relaciones, en la que cada una de ellas cumple determinadas condiciones mínimas de diseño:

- 1) No deben existir dos tuplas iguales.
- 2) Cada atributo sólo puede tomar un único valor del dominio, es decir, no pueden contener listas de valores.
- 3) El orden de las tuplas dentro de la relación y el de los atributos, dentro de cada tupla, no es importante.

**Cardinalidad:** Número de tuplas que contiene una relación.

**Esquema:** Es la parte constante de una relación, es decir, su estructura. Esto es, el esquema es una lista de los atributos que definen una relación y sus dominios.

**Instancia:** Es el conjunto de las tuplas que contiene una relación en un momento determinado. Es como una fotografía de la relación, que sólo es válida durante un periodo de tiempo concreto.



**Clave:** Es un conjunto de atributos que identifica de forma unívoca a una tupla. Puede estar compuesto por un único atributo o una combinación de varios.

Podemos clasificar las claves en distintos tipos:

- **Candidata:** cada una de las posibles claves de una relación, en toda relación existirá al menos una clave candidata. Esto implica que ninguna relación puede contener tuplas repetidas.
- **Primaria:** (o principal) es la clave candidata elegida por el usuario para identificar las tuplas. No existe la necesidad, desde el punto de vista de la teoría de bases de datos relacionales, de elegir una clave primaria. Además, las claves primarias no pueden tomar valores nulos. Es preferible, por motivos de optimización de MySQL, que estos valores sean enteros, aunque no es obligatorio. MySQL sólo admite una clave primaria por tabla, que es lógico, ya que la definición implica que sólo puede existir una.
- **Alternativa:** cada una de las claves candidatas que no son clave primaria, si es que existen.
- **Foránea:** (o externa) es el atributo (o conjunto de atributos) dentro de una relación que contienen claves primarias de otra relación. No hay nada que impida que ambas relaciones sean la misma.

**Interrelación:** Dos relaciones están interrelacionadas cuando una posee una clave foránea de la otra.

Según esto, existen dos tipos de interrelación:

- La interrelación entre entidades fuertes y débiles.
- La interrelación pura, entre entidades fuertes.

Al igual que en el modelo E-R, existen varios tipos de interrelación:

- **Uno a uno:** a cada tupla de una relación le corresponde una y sólo una tupla de otra.
- **Uno a varios:** a cada tupla una relación le corresponden varias en otra.



- **Varios a varios:** cuando varias tuplas de una relación se pueden corresponder con varias tuplas en otra.

### Paso del modelo E-R al modelo relacional

Existen varias reglas para convertir cada uno de los elementos de los diagramas E-R en tablas:

- ✓ Para cada conjunto de entidades fuertes se crea una relación con una columna para cada atributo.
- ✓ Para cada conjunto de entidades débiles se crea una relación que contiene una columna para los atributos que forman la clave primaria de la entidad fuerte a la que se encuentra subordinada y una columna para cada atributo de la entidad.
- ✓ Para cada interrelación se crea una relación que contiene una columna para cada atributo correspondiente a las claves principales de las entidades interrelacionadas.
- ✓ Lo mismo para entidades compuestas, añadiendo las columnas necesarias para los atributos añadidos a la interrelación.

Las relaciones se representan mediante sus esquemas, la sintaxis es simple: <nombre\_relación> (<nombre\_atributo\_i>,...) La clave principal se suele indicar mediante un subrayado.

**Integridad de datos:** El modelo relacional también provee mecanismos para mantener la integridad. Podemos dividir estos mecanismos en dos categorías:

- **Restricciones estáticas**, que se refieren a los estados válidos de datos almacenados.
- **Restricciones dinámicas**, que definen las acciones a realizar para evitar ciertos efectos secundarios no deseados cuando se realizan operaciones de modificación o borrado de datos.



## Restricciones sobre claves primarias

En cuanto a las restricciones estáticas, las más importantes son las que afectan a las claves primarias. Ninguna de las partes que componen una clave primaria puede ser *NULL*. Las modificaciones de claves primarias deben estar muy bien controladas. Dado que una clave primaria identifica de forma unívoca a una tupla en una relación, parece poco lógico que exista necesidad de modificarla, ya que eso implicaría que no estamos definiendo la misma entidad.

**Integridad referencial:** La integridad referencial consiste en que si un atributo o conjunto de atributos se define como una clave foránea, sus valores deben existir en la tabla en que ese atributo es clave principal.

Existen varias formas de asegurarse de que se conserva la integridad referencial:

- **Restringir operaciones:** borrar o modificar tuplas cuya clave primaria es clave foránea en otras tuplas, sólo estará permitido si no existe ninguna tupla con ese valor de clave en ninguna otra relación. Es decir, si el valor de una clave primaria en una tupla es "clave1", sólo podremos eliminar esa tupla si el valor "clave1" no se usa en ninguna otra tupla, de la misma relación o de otra, como valor de clave foránea.
- **Transmisión en cascada:** borrar o modificar tuplas cuya clave primaria es clave foránea en otras implica borrar o modificar las tuplas con los mismos valores de clave foránea. Si en el caso anterior, modificamos el valor de clave primaria "clave1" por "clave2", todas las apariciones del valor "clave1" en donde sea clave foránea deben ser sustituidos por "clave2".
- **Poner a nulo:** cuando se elimine una tupla cuyo valor de clave primaria aparece en otras relaciones como clave foránea, se asigna el valor *NULL* a dichas claves foráneas. De nuevo, siguiendo el ejemplo anterior, si eliminamos la tupla con el valor de clave primaria "clave1", en todas las tuplas donde aparezca ese valor como clave foránea se sustituirá por *NULL*.



**Tipos de columnas:** En MySQL existen bastantes tipos diferentes disponibles, de modo que será mejor que los agrupemos por categorías: de caracteres, enteros, de coma flotante, tiempos, bloques, enumerados y conjuntos.

### Tipos de datos de cadenas de caracteres utilizados

**VARCHAR (M)** [NATIONAL] VARCHAR (M) [BINARY] Contiene una cadena de longitud variable. Los valores válidos para M son de 0 a 255, y de 1 a 255 en versiones de MySQL anteriores a 4.0.2. Los espacios al final se eliminan. Si no se especifica la palabra clave BINARY estos valores se ordenan y comparan sin distinguir mayúsculas y minúsculas. VARCHAR es un alias para CHARACTER VARYING.

### Tipos de dato entero utilizado

#### **INT**

INT [(M)] [UNSIGNED] [ZEROFILL]

Contiene un entero de tamaño normal. El rango con signo está entre -2147483648 y 2147483647. El rango sin signo, entre 0 y 4294967295.

INTEGER

INTEGER [(M)] [UNSIGNED] [ZEROFILL]

Es sinónimo de INT.

DATE

TINYINT

### Tipos de datos para datos sin tipo o grandes bloques de datos

**MEDIUMBLOB**

**MEDIUMTEXT**

Contiene una columna BLOB o TEXT con una longitud máxima de 16777215 caracteres ( $2^{24}$ )

**LOBLOB**

**LONGTEXT**

**LOBLOB**

## LONGTEXT

Contiene una columna BLOB o TEXT con una longitud máxima de 4294967298 caracteres ( $2^{32}$ )

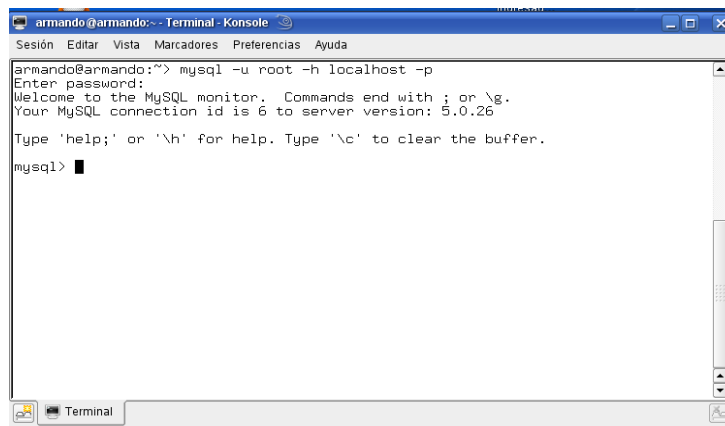
### 5.2.2 Cliente MySQL

Existen muchas formas de establecer una comunicación con el servidor de MySQL. En nuestros programas, generalmente, usaremos un API para realizar las consultas con el servidor. Usando una interfaz grafica como Qt, por ejemplo, este API está integrado con el lenguaje, en C++ se trata de librerías de enlace dinámico, etc.

Para este trabajo monográfico usaremos MySQL de forma directa, mediante un cliente ejecutándose en una consola (una ventana DOS en Windows, o un Shell en Linux u otros sistemas).

Veamos un ejemplo sencillo. Para ello abrimos una consola y tecleamos "MySQL". (Si estamos en Windows y no está definido el camino para MySQL tendremos que hacerlo desde "C:\mysql\bin").

Para entrar en la consola de MySQL se requieren ciertos parámetros. Se debe tener presente que el servidor es multiusuario, y que cada usuario puede tener distintos privilegios, tanto de acceso a tablas como de comandos que puede utilizar. La forma general de iniciar una sesión MySQL es: **mysql -h host -u usuario -p**



```
armando@armando:~$ mysql -u root -h localhost -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6 to server version: 5.0.26

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Podemos especificar el ordenador donde está el servidor de bases de datos (host) y nuestro nombre de usuario. Los parámetros "-h" y "-u" indican que los parámetros a continuación son, respectivamente, el nombre del host y el usuario. El parámetro "-p" indica que se debe solicitar una clave de acceso.

En versiones de MySQL anteriores a la 4.1.9 es posible abrir un cliente de forma anónima sin especificar una contraseña. Pero no es correcto, y de hecho, las últimas versiones de MySQL no lo permiten. Durante la instalación de MySQL se nos pedirá que elijamos una clave de acceso para el usuario 'root', deberemos usar esa clave para iniciar una sesión con el cliente MySQL.

```
mysql -h localhost -u root -p
```

```
Enter password: *****
```

```
Welcome to the MySQL monitor. Commands end with ; or \g.
```

```
Your MySQL connection id is 76 to server version: 4.1.9-nt
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql>
```

Para salir de una sesión del cliente de MySQL se usa el comando "QUIT".

```
mysql> QUIT
```

```
Bye y retornamos a la shell.
```

### Algunas consultas

Para hacer consultas se usa la sentencia SQL SELECT, por ejemplo:

```
mysql> SELECT VERSION (), CURRENT_DATE;
```

```
+-----+-----+
| VERSION () | CURRENT_DATE |
+-----+-----+
| 5.0.26      | 2007-08-28   |
+-----+-----+
```

```
1 row in set (0.53 sec)
```

```
mysql>
```



SELECT es la sentencia SQL para seleccionar datos de bases de datos, pero también se puede usar, como en este caso, para consultar variables del sistema o resultados de funciones. En este caso hemos consultado el resultado de la función VERSION y de la variable CURRENT\_DATE.

Esto es sólo un aperitivo, hay muchísimas más opciones, y mucho más interesantes. Usaremos SELECT para mostrar todas las filas de una tabla. Para ello se usa la sentencia:

```
mysql> SELECT * FROM <tabla>;
```

## **Usuarios y privilegios**

Cuando se trabaja con bases de datos reales y con aplicaciones de gestión de bases de datos, es muy importante definir otros usuarios, además del root, que es el administrador. Esto nos permitirá asignar distintos privilegios a cada usuario, y nos ayudará a proteger las bases de datos.

Podremos por ejemplo, crear un usuario que sólo tenga posibilidad de consultar datos de determinadas tablas o bases de datos, pero que no tenga permiso para añadir o modificar datos, o modificar la estructura de la base de datos.

Otros usuarios podrán insertar datos, y sólo algunos (o mejor, sólo uno) podrán modificar la estructura de las bases de datos: los administradores.



## 5.2.3 Lenguaje SQL

### 5.2.3.1 Creación de bases de datos y tablas

A nivel teórico, existen dos lenguajes para el manejo de bases de datos:

**DDL (Data Definition Language)** Lenguaje de definición de datos. Es el lenguaje que se usa para crear bases de datos y tablas, y para modificar sus estructuras, así como los permisos y privilegios. Este lenguaje trabaja sobre unas tablas especiales llamadas *diccionario de datos*.

**DML (Data Manipulation Language)** lenguaje de manipulación de datos. Es el que se usa para modificar y obtener datos desde las bases de datos.

SQL engloba ambos lenguajes DDL+DML, ambos forman parte del conjunto de sentencias de SQL.

### Crear una base de datos

Cada conjunto de relaciones que componen un modelo completo forma una base de datos. Desde el punto de vista de SQL, una base de datos es sólo un conjunto de relaciones (o tablas), y para organizarlas o distinguirlas se accede a ellas mediante su nombre. A nivel de sistema operativo, cada base de datos se guarda en un directorio diferente.

Crear una base de datos es una tarea muy simple. Claro que, en el momento de crearla, la base de datos estará vacía, es decir, no contendrá ninguna tabla.

Para empezar, crearemos una base de datos solo para nosotros, y la llamaremos "Almacén". Para crear una base de datos se usa una sentencia.



## CREATE DATABASE

```
mysql> CREATE DATABASE almacen;
```

```
Query OK, 1 row affected (0.03 sec)
```

Podemos averiguar cuántas bases de datos existen en nuestro sistema usando la sentencia SHOW DATABASES:

```
mysql> SHOW DATABASES;
```

```
+-----+
| Database          |
+-----+
| information_schema |
| almacen           |
| mysql             |
| test              |
+-----+
```

```
4 rows in set (0.30 sec)
```

Seleccionaremos Database como la base de datos por defecto. Esto nos permitirá obviar el nombre de la base de datos en consultas. Para seleccionar una base de datos se usa el comando [USE](#), que no es exactamente una sentencia SQL, sino más bien de una opción de **MySQL**:

```
mysql> USE almacen;
```

```
Database changed
```

```
mysql>
```

## Crear una tabla

La sentencia CREATE TABLE que sirve para crear tablas. La sintaxis de esta sentencia es muy compleja, ya que existen muchas opciones y tenemos muchas posibilidades diferentes a la hora de crear una tabla.



La sentencia CREATE TABLE creará una tabla con las columnas que indiquemos. Crearemos una tabla que nos permitirá almacenar nombres de usuario y su contraseña. Debemos indicar el nombre de la tabla y los nombres y tipos de las columnas:

```
mysql> USE almacen
```

```
Database changed
```

```
mysql> CREATE TABLE usuarios (nombre VARCHAR (50), pass VARCHAR(50),  
tipo tinyint(1), foto LONGBLOB);
```

```
Query OK, 0 rows affected (0.53 sec)
```

```
mysql>
```

Hemos creado una tabla llamada "usuarios" con cuatro columnas: "nombre" que puede contener cadenas de hasta 50 caracteres, "pass" de hasta 50 caracteres de tipo varchar, "tipo" especifica si es usuario o administrador y "foto" de tipo longblob.

Podemos consultar cuántas tablas y qué nombres tienen en una base de datos, usando la sentencia SHOW TABLES:

```
mysql> SHOW TABLES;
```

```
+-----+
```

```
| Tables_in_almacen |
```

```
+-----+
```

```
| usuarios          |
```

```
+-----+
```

```
1 row in set (0.01 sec)
```

```
mysql>
```

**La sintaxis para definir columnas es:**

```
nombre_col tipo [NOT NULL | NULL] [DEFAULT valor_por_defecto]
[AUTO_INCREMENT] [[PRIMARY] KEY] [COMMENT 'string']
[definición_referencia]
```

**Opciones de columna**

**Valores nulos:** Al definir cada columna podemos decidir si podrá o no contener valores nulos. Aquellas columnas que son o forman parte de una clave primaria no pueden contener valores nulos.

**Columna auto incrementadas**

En MySQL tenemos la posibilidad de crear una columna auto incrementada, aunque esta columna sólo puede ser de tipo entero. Si al insertar una fila se omite el valor de la columna auto incrementado o si se inserta un valor nulo para esa columna, su valor se calcula automáticamente, tomando el valor más alto de esa columna y sumándole una unidad. Esto permite crear, de una forma sencilla, una columna con un valor único para cada fila de la tabla.

**Comentarios**

Adicionalmente, al crear la tabla, podemos añadir un comentario a cada columna. Este comentario sirve como información adicional sobre alguna característica especial de la columna, y entra en el apartado de documentación de la base de datos: usamos

## Comillas simples

Como creamos las definición de las tablas

```
CREATE TABLE usuarios (nombre VARCHAR (50), pass VARCHAR (50),  
foto LONGBLOB);
```

### 5.2.3.2 Inserción y modificación de datos

Una base de datos sin datos no sirve para mucho, de modo que veremos cómo agregar, modificar o eliminar los datos que contienen nuestras bases de datos.

#### *Inserción de nuevas filas*

La forma más directa de insertar una fila nueva en una tabla es mediante una sentencia INSERT. En la forma más simple de esta sentencia debemos indicar la tabla a la que queremos añadir filas, y los valores de cada columna. Las columnas de tipo cadena o fechas deben estar entre comillas sencillas o dobles, para las columnas numéricas esto no es imprescindible, aunque también pueden estar entrecomilladas.

#### *Reemplazar filas*

Existe una sentencia REPLACE, que es una alternativa para INSERT, que sólo se diferencia en que si existe algún registro anterior con el mismo valor para una clave primaria o única, se elimina el viejo y se inserta el nuevo en su lugar.

```
REPLACE [LOW_PRIORITY | DELAYED]  
[INTO] tbl_name [(col_name...)]  
VALUES ({expr | DEFAULT}...), (...)
```

```
REPLACE [LOW_PRIORITY | DELAYED]
  [INTO] tbl_name
  SET col_name= {expr | DEFAULT}...
```

### **Actualizar filas**

Podemos modificar valores de las filas de una tabla usando la sentencia [UPDATE](#). En su forma más simple, los cambios se aplican a todas las filas, y a las columnas que especifiquemos.

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
  SET col_name1=expr1 [, col_name2=expr2 ...]
  [WHERE where_definition]
  [ORDER BY ...]
  [LIMIT row_count]
```

### **Eliminar filas**

Para eliminar filas se usa la sentencia [DELETE](#). La sintaxis es muy parecida a la de [UPDATE](#):

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM table_name
  [WHERE where_definition]
  [ORDER BY ...]
  [LIMIT row_count]
```

La forma más simple es no usar ninguna de las cláusulas opcionales.

### **Vaciar una tabla**

Cuando queremos eliminar todas las filas de una tabla, vimos en el punto anterior que podíamos usar una sentencia DELETE sin condiciones. Sin embargo, existe una sentencia alternativa, TRUNCATE, que realiza la misma tarea de una forma mucho más rápida.

La diferencia es que DELETE hace un borrado secuencial de la tabla, fila a fila. Pero TRUNCATE borra la tabla y la vuelve a crear vacía, lo que es mucho más eficiente

### 5.2.3.3 Selección de datos

Usaremos las Sentencias **SELECT**

Sintaxis:

```
SELECT [ALL | DISTINCT | DISTINCTROW]
  expresion_select...
FROM referencias_de_tablas
WHERE condiciones
[GROUP BY {nombre_col | expresion | position}
  [ASC | DESC]... [WITH ROLLUP]]
[HAVING condiciones]
[ORDER BY {nombre_col | expresion | posicion}
  [ASC | DESC] ,...]
[LIMIT {[desplazamiento,] contador | contador OFFSET desplazamiento}]
```

#### ***Forma incondicional***

La forma más sencilla es la que hemos usado hasta ahora, consiste en pedir todas las columnas y no especificar condiciones.

```
mysql> SELECT * FROM usuarios;
```

#### **Limitar las columnas: proyección**

Una de las operaciones del álgebra relacional era la proyección, que consistía en seleccionar determinados atributos de una relación, mediante la sentencia SELECT es posible hacer una proyección de una tabla, seleccionando las columnas de las que queremos obtener datos.



## Limitar las filas: selección

Otra de las operaciones del álgebra relacional era la selección, que consistía en seleccionar filas de una relación que cumplieran determinadas condiciones.

## Agrupar filas

Es posible agrupar filas en la salida de una sentencia `SELECT` según los distintos valores de una columna, usando la cláusula `GROUP BY`. Esto, en principio, puede parecer redundante, ya que podíamos hacer lo mismo usando la opción `DISTINCT`. Sin embargo, la cláusula `GROUP BY` es más potente:

## Ordenar resultados

Podemos añadir una cláusula de orden `ORDER BY` para obtener resultados ordenados por la columna que queramos. Existe una opción para esta cláusula para elegir el orden, ascendente o descendente. Se puede añadir a continuación `ASC` o `DESC`, respectivamente. Por defecto se usa el orden ascendente, de modo que el modificador `ASC` es opcional.



## **VI METODOLOGIA**

### **6.1 Recopilación de la información:**

Toda la información que se necesitó para la realización de este trabajo monográfico, se obtuvo mediante búsquedas en Internet, libros y manuales.

### **6.2 Especificación de requisitos:**

Los requisitos estarán enfocados a nivel de hardware y software: Los Requisitos mínimos son los siguientes:

- **Estándares:** La aplicación de los estándares en la creación de la aplicación, nos garantiza la reusabilidad y migración entre instalaciones o plataformas diferentes.
- **Tipo de licencia de uso:** Utilizamos licencia GPL (Global Public License). En éste punto se ha tenido en cuenta las ventajas de adaptación que ofrece el software libre, ya que la plataforma debe permitir la integración con nuestros sistemas de información, así como su personalización para una correcta integración.
- **Seguridad:** El gestor MySQL nos provee una excelente seguridad de los datos.
- **Interactividad:** El usuario interactúa con el sistema introduciendo datos en el y obteniendo a su vez resultados.

### **6.3 Plataforma a utilizar:**

Hemos elegido Linux openSuSe debido a su facilidad de uso, además, es un software libre respaldado por Novell.



#### **6.4 Instalación del sistema Linux openSuSe 10.2:**

Teniendo la plataforma sobre la cual trabajaremos, será necesaria la instalación, configuración del sistema operativo Linux openSuSe 10.2, porque es un sistema fiable, seguro, y principalmente de distribución libre.

#### **6.5 Instalación y configuración de MySQL**

Una vez instalada la plataforma seleccionada procedemos a la instalación del servidor y cliente MySQL, para ello haremos uso de la herramienta gráfica YaST.

#### **6.6 Instalación y configuración del creador de interfaces gráficas Qt:**

Luego de haber instalado MySQL procedemos a instalar el creador de interfaces gráficas Qt utilizando también la herramienta Yast.

#### **6.7 Personalización:**

Se procedió a personalizar la interfaz gráfica de la plataforma para que facilite el acceso a la información, presentando al usuario final una serie de herramientas básicas y útiles para una mejor interacción.

#### **6.8 Explotación y Mantenimiento:**

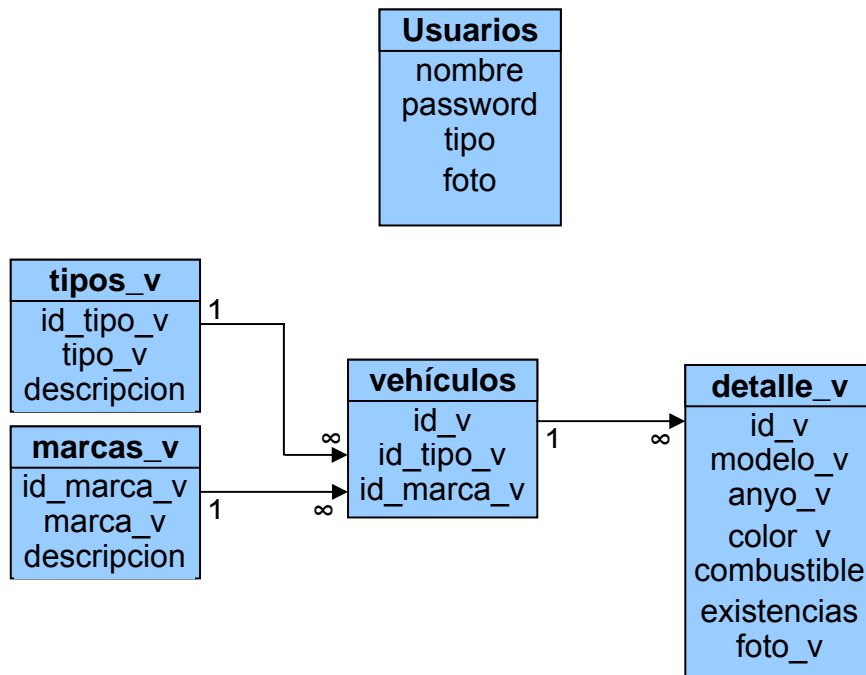
Se hará una prueba experimental con la Aplicación “*Control del Stock de vehículos en bodega*”, para demostrar la utilidad y facilidad de uso del creador de interfaces gráficas Qt y la interoperatividad con MySQL, y resolver las dificultades que se presenten durante el periodo de prueba.



## **6.9 Resultados:**

Con la información recopilada durante los procesos antes mencionados se procedió a la redacción del documento final, donde están plasmados los resultados obtenidos en este trabajo monográfico (Ver anexos).

## VII DIAGRAMA E-R



## **VIII Conclusiones**

En el transcurso de nuestro trabajo monográfico hemos podido comprobar mediante un ejemplo útil que Qt es una herramienta muy potente y fácil de utilizar, además, ha llenado nuestras expectativas propuestas como desarrolladores de sistemas, tanto en la velocidad de creación de interfaces gráficas como en su portabilidad.

Al utilizar MySQL como gestor de bases de datos obtuvimos grandes facilidades, ya que posee herramientas gráficas que nos permitieron la rápida creación de nuestra base de datos (almacén) como son MySQL Administrador y MySQL Query Browser.

Cuando utilizamos la herramienta QtDesigner para crear nuestra aplicación obtuvimos muchas facilidades y al compararlo con otros diseñadores de interfaz gráfica podemos afirmar que poseen muchas características similares pero las aplicaciones desarrolladas con Qt son portables y pueden utilizar licencia GPL.

Los inconvenientes que se nos presentaron en la elaboración de este trabajo fueron referentes al desconocimiento de las librerías que posee Qt y al obtener la información necesaria de las mismas los pudimos superar fácilmente, para ello nos apoyamos en la herramienta QtAssistant que provee mucha información de manera rápida y la amplia comunidad Qt de Internet.

La trilogía Qt, MySQL y openSuSe nos ha demostrado que se pueden crear aplicaciones grandes y complejas en poco tiempo y con pocos recursos en comparación de otros entornos de desarrollo.

## **IX Recomendaciones**

- Instalar las versiones de cada programa descrita en nuestro trabajo monográfico para el correcto funcionamiento de los mismos.
- Leer la documentación adjunta a la instalación de Qt3 (Qt Assistant).
- Poseer conocimientos generales de C++ y Sql.
- Utilizar los tutoriales que trae integrado Qt Assistant para adquirir un aprendizaje gradual.
- Hacer uso de Internet y de la comunidad Qt, para obtener información extra, la cual nos servirá en la solución de posibles problemas en el uso de la herramienta Qt.
- Tener conocimientos técnicos de ingles.
- Utilizar software libre para evitar problemas legales.
- Tomar este trabajo monográfico como referencia para crear una aplicación más compleja.
- Seguir las recomendaciones descrita anteriormente para obtener los resultados deseados.



## **X RECURSOS DISPONIBLES**

Los recursos mínimos necesarios para el desarrollo de nuestra monografía son:

1. Computadora Pentium IV, 2.4Ghz, 256 MB de DDRAM, 30 GB de disco duro,
2. Linux openSuSe versión 10.2
3. Software utilizado: MySql 5.0.26, Qt 3.3.7, Qt Designer 3,3,7, Qt Assistant 3,3,7, MySql Administrator 1.2.12, MySql Query Browser 1.2.12, Open Office 2.0.4.
4. Conectividad a Internet.
5. Otros materiales (Impresora, memorias flash etc.).
6. Traductor en Linea.

## XI Bibliografía

Documentación de Internet:

- <http://www.trolltech.com/qt>
- <http://www.mysql.com/>
- <http://www.traductoronline.com/>

Libros:

- C++ GUI Programming with Qt 3. Prentice Hall en asociación con Trolltech Press, Jasmin Blanchette, Mark Summerfield.
- Manual de referencia de MySQL 5.0,
- Páginas del manual de Linux (man).

Otros:

- Ayuda proporcionada por Qt mediante la aplicación QtAssistant,



## XII ANEXOS

### Paquetes que deberá instalar

- MySql
- Mysql-Client
- Qt3-mysql
- Qt3-devel
- Qt3-devel-doc
- Qt3-examples
- Automake
- Gcc-c++

Después reiniciamos la maquina.

### Post instalación de mysql

- Iniciar el servidor por primera vez  
`$/etc/init.d/mysql start`
- Luego de instalar MySQL probaremos el servidor  
`$mysql -u root -h localhost`
- Terminamos con ctrl+D ó escribiendo quit
- Para crear por primera vez la contraseña de root  
`$mysqladmin -u root -h localhost password 'password_nueva'`
- Para cambiar la contraseña del root de MySQL  
`$mysqladmin -u root -h localhost -p`



- Luego de presionar entrar se nos pedirá la contraseña nueva
- Para cambiar una contraseña existente  
`$mysqladmin -u root -h localhost -p password 'password_nueva'`
- Luego se nos pedirá la password\_antigua
- Entrar en la consola de cliente mysql  
`$mysql -u root -h localhost -p`
- Para conectarnos remotamente al servidor debemos crear el usuario con en comando grant desde el cliente mysql  
`mysql> grant all on *.* to nombre_usuario@dominio identified by 'password';`
- Creamos un nuevo usuario para mysql que esta representado por nombre \_ usuario con un dominio o dirección IP representada por dominio y una contraseña representada por password se le han asignado todos los permisos sobre todas las bases de datos que maneja mysql esto debe modificarse según sea conveniente. Podemos obtener ayuda tecleando desde la consola de mysql \h grant;
- Para mayor información de este comando revise el manual de mysql.
- Para auto inicializar el servidor cada vez que encendamos la computadora
  - 1) Entrar como root  
`$su`
  - 2) Indicar al sistema que debe iniciar el demonio de mysql  
`$chkconfig mysql on`
- Crear el usuario “almacen” con contraseña “almacen” desde MySQLAdministrator

***Presentación de formularios de la aplicación “Stock de vehículos en almacén” y archivos utilizados para desarrollarlos***

```
//Archivo globales.h
#ifdef ALLOC
#define GLOBAL
#else
#define GLOBAL extern
#endif
#include <qsqlquery.h>
#include <qstring.h>
GLOBAL QSqlQuery *query;//Consulta que realizamos sobre la base de datos
GLOBAL bool editando;//Indicar si estamos editando un registro existente
GLOBAL QString tipo_editando;//Almacenamos el tipo del registro que editamos
GLOBAL QString marca_editando;//Almacenamos el marca del registro que editamos
GLOBAL QString nombre_editando;//Almacenamos el nombre del registro que editamos
GLOBAL int id_editando; //Almacenamos el nombre del id del registro que editamos
GLOBAL int pos_editando;//Posicion del registro que se esta editando
GLOBAL QString usuario; //Almacena el nombre del usuario de sesion
GLOBAL bool tipo_usuario; //Tipo de usuario que se conecta al servidor
(true=administrador,false=usuario)
GLOBAL QString servidor; //Almacena el nombre del servidor al que nos conectamos
//Fin archivo globales.h

//Archivo conexion.h
#ifndef CONEXION_H
#define CONEXION_H
#include <qstring.h>
bool createConnection(QString server="localhost");
#endif
//Fin archivo conexion.h

//Archivo conexion.cpp
#include<qsqldatabase.h>
#include<qsqldriver.h>
#include <qmessagebox.h>
#include <qsqldatabase.h>
#include <qsqlquery.h>
#include "conexion.h"
QSqlDatabase *db;//Base de datos a la cual nos conectamos

bool createConnection(QString server)
{
    //Establecer el driver del motor de base de datos a utilizar asi como los
datos necesarios para la conexion
    db = QSqlDatabase::addDatabase("QMYSQL3");
    db->setHostName(server);
```

```

db->setDatabaseName("almacen");
db->setUserName("almacen");
db->setPassword("almacen");
db->setConnectOptions( "CLIENT_SSL;CLIENT_IGNORE_SPACE" ); //Usar una
conexion SSL con el servidor

if (!db->open())//Comprobar si se establecio la conexion
{
    db->setConnectOptions(); //Limpiar las opciones de la cadena de
conexion
    db->lastError().showMessage();//Mostrar el error producido
    return false;
}
return true;
}
//Fin archivo conexion.cpp

```

Este formulario es para la verificación de usuario en el cual se especificará el nombre del servidor al cual se desea conectar, el nombre de usuario y contraseña.



### Formulario frmLogin

```

//Archivo frmLogin.ui.h
#include <qmessagebox.h>
#include <qlineedit.h>
#include <sqldatabase.h>
#include <sqlquery.h>
#include <qmessagebox.h>
#include <sqlrecord.h>
#define ALLOC 0
#include "globales.h"

```



```
#include "conexion.h"

void frmLogin::ok()
{
    bool fin=true;
    if(txtServer->text() == "")
    {
        QMessageBox::information(this, tr("Nombre del servidor vacio"),
                                tr("Nombre del servidor incorrecto"),
                                QMessageBox::Yes | QMessageBox::Default);

        fin=false;
        txtServer->setFocus();
        return;
    }

    if(txtNombre->text() == "")
    {
        QMessageBox::information(this, tr("Nombre vacio"),
                                tr("Nombre de usuario incorrecto"),
                                QMessageBox::Yes | QMessageBox::Default);

        fin=false;
        txtNombre->setFocus();
        return;
    }

    if(txtPassword->text() == "")
    {
        QMessageBox::information(this, tr("Clave vacia"),
                                tr("Clave de usuario incorrecta"),
                                QMessageBox::Yes | QMessageBox::Default);

        fin=false;
        txtPassword->setFocus();
        return;
    }

    QMessageBox mb(tr("Login"),tr("Conectando con el servidor..."),
    QMessageBox::Information,
                QMessageBox::NoIcon ,QMessageBox::NoIcon ,
    QMessageBox::NoIcon,this,false);
    mb.setIconPixmap(QPixmap(QPixmap::fromMimeSource( "conectando.png" ) ));
    mb.show();

    if (!createConnection(txtServer->text()))
        return;

    QSqlQuery query;
    query.exec("SELECT * FROM usuarios WHERE nombre='"+txtNombre->text()+"'
and password='"+txtPassword->text()+"'");

    if (query.next()==false)
    {
        QMessageBox::information(this, tr("Incorrecto"),
                                tr("Nombre o Clave de usuario incorrectos"),
                                QMessageBox::Yes | QMessageBox::Default);

        fin=false;
    }
}
```



```
        txtNombre->setFocus();
        txtNombre->selectAll();
        return;
    }

    if(fin==true)
    {
        extern QSqlDatabase *db;//Declaramos el puntero db como extern para
poder utilizarlo debido a que no existe en este ambito
        servidor=txtServer->text();
        usuario=txtNombre->text();
        tipo_usuario=query.value(db->record(query).position("tipo")).toBool();
        accept();
    }
}
//Fin del archivo frmLogin.ui.h
```

```
//Archivo MaskLogin.h
#ifndef MASKLOGIN_H
#define MASKLOGIN_H

#include "frmLogin.h"
class MaskLogin : public frmLogin
{
public:
    MaskLogin (QWidget *parent, const char *name, WFlags f);
    ~MaskLogin() {}

protected:
    void mousePressEvent( QMouseEvent *);
    void mouseMoveEvent( QMouseEvent *);

private:
    QPoint clickPos;
};

#endif
//Fin del archivo MaskLogin.h
```

```
//Archivo MaskLogin.cpp
#include <qimage.h>
#include <qpixmap.h>
#include <qbitmap.h>
#include "MaskLogin.h"

MaskLogin::MaskLogin (QWidget *parent, const char *name, WFlags
f):frmLogin(parent,name,false,f)
{
    QPixmap p(QPixmap::fromMimeSource("tux.png")); //Obtener la imagen
    setBackgroundPixmap( p );//Establecer la imagen como fondo del formulario
    setFixedSize( p.size() );//Cambiar el tamaño del formulario al tamaño
del pixmap
    setMask( *p.mask() );//Establecer la imagen como mascara para el
formulario
```



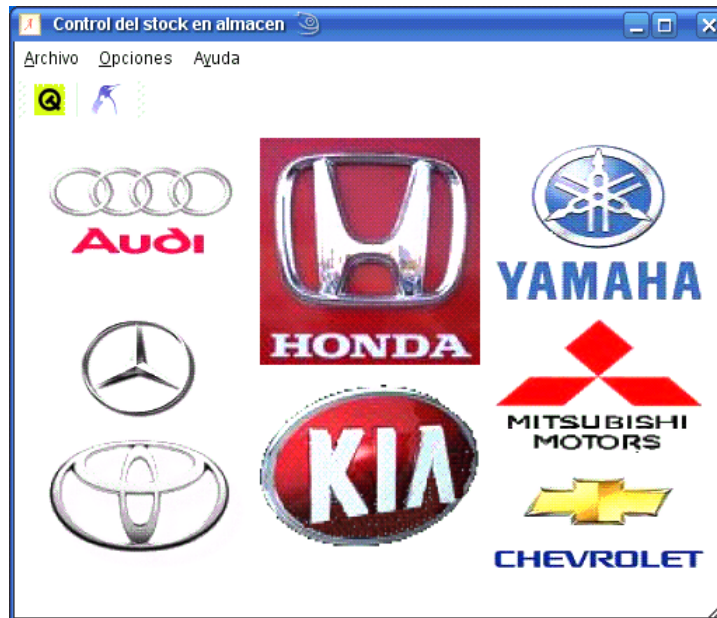
```
}
```

```
void MaskLogin::mousePressEvent( QMouseEvent *e )  
{  
    clickPos = e->pos();//Obtener la ubicacion del mouse al dar click en el  
    formulario  
}
```

```
void MaskLogin::mouseMoveEvent( QMouseEvent *e )  
{  
    move( e->globalPos() - clickPos );//Mover el formulario con respecto al  
    movimiento del mouse  
}
```

```
//Fin del archivo MaskLogin.cpp
```

Luego de comprobar el usuario es correcto nos llevará al formulario principal de la aplicación.



### Formulario: frmAlmacen

```
//Archivo frmAlmacen.ui.h
#include <qmessagebox.h>
#include <qstatusbar.h>
#include <qpixmap.h>
#define ALLOC 0
#include "globales.h"
#include "MaskLogin.h"
#include "MyPixmapLabel.h"
#include "frmUsuarios.h"
#include "frmTipos.h"
#include "frmMarcas.h"
#include "frmVehiculos.h"
#include "frmDetalleV.h"
#include "frmReporte.h"

int foto_actual;

void frmAlmacen::init()
{
    setWindowOpacity(0.8); //Soloamente par S.O. Mac y Windows 2000 en adelante
    MaskLogin login(0,0,Qt::WStyle_Customize|Qt::WStyle_NoBorder); //Invocar al
    constructor indicando que crearemos un widget estilo personalizado y sin borde

    if(login.exec() == QDialog::Rejected) //Esperamos la verificaciÃ³n del
    usuario
    {
        QMessageBox::critical( 0, tr("Almacen"),tr("Error de identificacion de
        usuario \n Abortando aplicacion"));
    }
}
```





```
        exit(-1);
    }

    statusBar()->message("Conectado a "+servidor,2000);//Indicamos que nos
hemos conectado satisfactoriamente al servidor

    query=new QSqlQuery();//Asignamos memoria a la variable query
    foto_actual=0;
    move(0,0);
}

void frmAlmacen::Exit()
{
    close(); //Cerrar la ventana
}

void frmAlmacen::AcercaDeAlmacen()
{
    //Crear y mostrar un mensaje indicando quien construyo esta aplicacion :->
    QMessageBox::about(this,"Acerca de \"Almacen\"...", "<b>Elaborado bajo
licencia GPL por estudiantes de la Universidad Nacional Autonoma de Nicaragua
UNAN - Leon</b><center>Armando Rodriguez Lopez<br>Darling Maria Sevilla
<br>Mauricio Antonio Perez<br><br><b>Almacen 2007</b></center>");
}

void frmAlmacen::AboutQt()
{
    //Mostramos el mensaje que indica la version de Qt
    QMessageBox::aboutQt(this);
}

void frmAlmacen::VerUsuarios()
{
    if(!tipo_usuario)//Si el usuario no es administrador no se permite esta
opcion
    {
        QMessageBox::critical( 0, tr("Almacen"),tr("Error: Usted no es
administrador \n No puede utilizar esta opcion"));
        return;
    }
    //Crear un objeto de tipo frmUsuarios para la manipulacion de los usuarios
del sistema
    frmUsuarios users;
    users.exec(); //Crear un objeto de tipo frmDetalleV para la
manipulacion de los vehiculos de manera individual del sistema
}

void frmAlmacen::VerTipos()
{
    //Crear un objeto de tipo frmTipos para la manipulacion de los tipos de
vehiculos del sistema
    frmTipos tipos;
    tipos.exec(); //Mostrar la ventana de manera modal
}

```



```
void frmAlmacen::VerMarcas()
{
    //Crear un objeto de tipo frmMarcas para la manipulacion de las marcas de
    los vehiculos del sistema
    frmMarcas marcas;
    marcas.exec(); //Mostrar la ventana de manera modal
}

void frmAlmacen::VerVehiculosGeneral()
{
    query->exec("SELECT tipo_v FROM tipos_v");//Seleccionar los tipos
    existentes
    if(!query->size())//Si no existen tipos de vehiculos...
    {
        QMessageBox::information(this, tr("Almacen -- No hay tipos"),tr("No
    existen tipos de vehiculos, primero debe crear al menos uno"));
        return;
    }

    query->exec("SELECT marca_v FROM marcas_v");//Seleccionar las marcas
    existentes
    if(!query->size())//Si no existen marcas de vehiculos...
    {
        QMessageBox::information(this, tr("Almacen -- No hay marcas"),tr("No
    existen marcas de vehiculos, primero debe crear al menos una"));
        return;
    }

    //Crear un objeto de tipo frmVehiculos para la manipulacion de los
    vehiculos de manera general del sistema
    frmVehiculos v_general;
    v_general.exec(); //Mostrar la ventana de manera modal
}

void frmAlmacen::VerDetalleVehiculos()
{
    query->exec("SELECT id_v FROM vehiculos");//Seleccionar las marcas
    existentes
    if(!query->size())//Si no existen marcas de vehiculos...
    {
        QMessageBox::information(this, tr("Almacen -- No hay vehiculos"),tr("No
    existen vehiculos, primero debe crear al menos uno"));
        return;
    }

    //Crear un objeto de tipo frmDetalleV para la manipulacion de los
    vehiculos de manera individual del sistema
    frmDetalleV detalle_v;
    detalle_v.exec(); //Mostrar la ventana de manera modal y esperar resultado
}

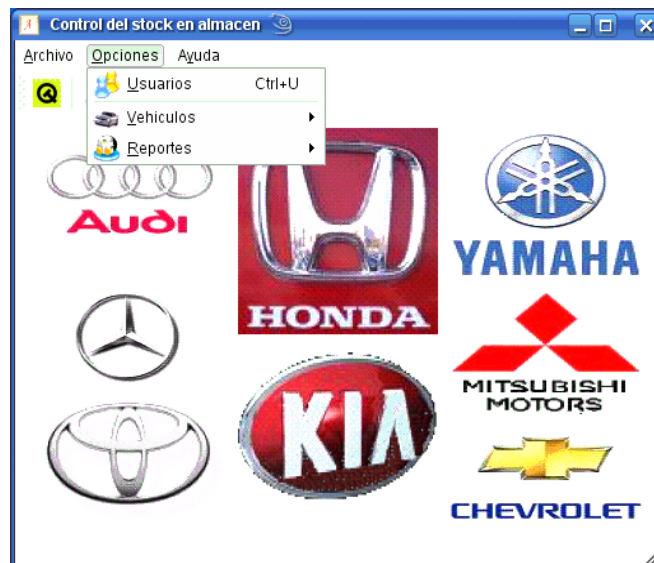
void frmAlmacen::ReportePersonalizado()
{
    //Crear un objeto de tipo frmReporte para la presentacion del reporte
    personalizado
    frmReporte *report_v=new frmReporte();
}
```

```
report_v->show();  
report_v->EditarReporte();  
}  
//Fin del archivo frmAlmacen.ui.h
```

Desde este formulario podemos acceder a toda la información del sistema mediante el menú.



Desde el menú Opciones accedemos a los distintos formularios para la manipulación del sistema.



La opción Usuarios nos permite manipular los datos de los usuarios del sistema. Desde este formulario podemos agregar, editar y borrar los usuarios de la aplicación de manera fácil y segura además podemos desplazarnos por los usuarios existentes así como buscar un usuario específico.



### Formulario frmUsuarios

```
//Archivo MyPixmapLabel.h
#ifndef MYPIXMAPLABEL_H
#define MYPIXMAPLABEL_H

#include <qlabel.h>
#include <qpixmap.h>
#include <qmessagebox.h>
#include <qdragobject.h>

class QDragEnterEvent;
class QDragDropEvent;

class MyPixmapLabel : public QLabel
{
    Q_OBJECT
public:
    MyPixmapLabel ( ) ;
    MyPixmapLabel ( QWidget * parent, const char * name=0, WFlags f=0) ;
    MyPixmapLabel ( const QString & text, QWidget * parent, const char *
name=0, WFlags f=0) ;
```



```
MyPixmapLabel ( QWidget * buddy, const QString & text, QWidget * parent,
const char * name=0, WFlags f=0) ;

~MyPixmapLabel() {}

bool acceptDrop( const QMimeSource *mime ) const;
void dragEnterEvent( QDragEnterEvent *evt );
void dropEvent( QDropEvent *evt );
int dragging;
void mousePressEvent( QMouseEvent *evt );
void mouseMoveEvent( QMouseEvent * evt);
QDragObject *dragObject();
};

#endif
//Fin del archivo MyPixmapLabel.h

//Archivo MyPixmapLabel.cpp
#include "MyPixmapLabel.h"
#include <qurl.h>

MyPixmapLabel::MyPixmapLabel ( ):QLabel(0,0,0)
{
    setFrameStyle( QFrame::Panel | QFrame::Sunken );//Establecer el estilo
del marco de la etiqueta
    setScaledContents ( TRUE) ;//Escalar la imagen para que se ajuste al
tamaño de la etiqueta
    setAcceptDrops( TRUE );//Permitit soltar algo sobre la etiqueta
    setPixmap(QPixmap());//Establece una imagen vacia
    dragging = FALSE;//Indicar que no se esta arrastrando nada en ese momento
}

MyPixmapLabel::MyPixmapLabel ( QWidget * parent, const char * name, WFlags
f):QLabel(parent,name,f)
{
    setFrameStyle( QFrame::Panel | QFrame::Sunken );
    setScaledContents ( TRUE) ;
    setAcceptDrops( TRUE );
    setPixmap(QPixmap());
    dragging = FALSE;
}

MyPixmapLabel::MyPixmapLabel ( const QString & text, QWidget * parent, const
char * name, WFlags f) :QLabel(text,parent,name,f)
{
    setFrameStyle( QFrame::Panel | QFrame::Sunken );
    setScaledContents ( TRUE) ;
    setAcceptDrops( TRUE );
    setPixmap(QPixmap());
    dragging = FALSE;
}

MyPixmapLabel::MyPixmapLabel ( QWidget * buddy, const QString & text, QWidget
* parent, const char * name, WFlags f):QLabel(buddy,text,parent,name,f)
{
```

```
        setFrameStyle( QFrame::Panel | QFrame::Sunken );
        setScaledContents ( TRUE ) ;
        setAcceptDrops( TRUE );
        setPixmap(QPixmap());
        dragging = FALSE;
    }

void MyPixmapLabel::dragEnterEvent( QDragEnterEvent *evt )
{
    if ( QTextDrag::canDecode( evt ) )//Verifica si lo que se esta arrastrando
    es texto, si es asi acepta el evento
        evt->accept();
}

void MyPixmapLabel::dropEvent( QDropEvent *evt )
{
    QString rutafoto;
    QUrl *urlfoto=new QUrl();

    if ( QTextDrag::decode( evt,rutafoto) )
    {
        urlfoto=new QUrl(rutafoto);
        setPixmap(urlfoto->path());
    }
    delete urlfoto;
}

void MyPixmapLabel::mousePressEvent( QMouseEvent *evt )
{
    QLabel::mousePressEvent( evt );
    dragging = TRUE;
}

void MyPixmapLabel::mouseMoveEvent( QMouseEvent * )
{
    if ( dragging )
    {
        if(!pixmap()->isNull())
        {
            QDragObject *d = new QImageDrag(pixmap()->convertToImage(), this );
            d->dragCopy(); // do NOT delete d.
        }
        dragging = FALSE;
    }
}

bool MyPixmapLabel::acceptDrop( const QMimeSource *mime ) const
{
    if ( mime->provides( "text/plain" ) )
        return TRUE;
    return FALSE;
}

QDragObject *MyPixmapLabel::dragObject()
{

```

---



```
    if(pixmap()->isNull())
        return NULL;
    else
        return new QImageDrag( pixmap()->convertToImage(), this );
}
//Fin archivo MyPixmapLabel.cpp

//Archivo MyFilePreview.h
#ifndef MYFILEPREVIEW_H
#define MYFILEPREVIEW_H
#include <qlabel.h>
#include <qfiledialog.h>
//Derivamos una nueva clase de QLabel y QFilePreview para poder previsualizar
la imagen
class MyFilePreview : public QLabel, public QFilePreview
{
    Q_OBJECT
public:
    MyFilePreview( QWidget *parent=0 );//Constructor de la clase

    void previewUrl( const QUrl &u );//Funcion que nos permite utilizar el
fichero seleccionado
};
#endif

//Fin archivo MyFilePreview.h

//Archivo MyFilePreview.cpp
#include <qwmatrix.h>
#include <qpixmap.h>
#include <qurl.h>
#include "MyFilePreview.h"

MyFilePreview::MyFilePreview( QWidget *parent ) : QLabel(tr("This is not a
pixmap" ),parent,"foto")
{
    setFrameStyle( QFrame::Panel | QFrame::Sunken );
    setPixmap(QPixmap());
}

void MyFilePreview::previewUrl( const QUrl &u )//Funcion que nos permite
utilizar el fichero seleccionado
{
    QString path = u.path();//Convertimos la URL a string para tratar de
cargar la imagen
    QPixmap pix(path);//Creamos un pixmap si se selecciono una imagen
    while(pix.width(>300 ||pix.height(>300)//Si la imagen es muy grande, la
reducimos
    {
        QWMatrix m; //Creamos una nueva matriz para escalar el tamaño de la
imagen
        m.scale( 0.1, 0.1 );//Crear la escala para la imagen de 0.1% horizontal
y vertical
        pix=pix.xForm(m);//Guardamos la imagen escalada
    }
}
```



```
//Si no se selecciono una imagen, lo indicamos con un mensaje
if ( pix.isNull() )
    setText( tr("This is not a pixmap" ));
//De lo contrario establecemos la nueva imagen en la etiqueta
else
    setPixmap(pix);
}
//Fin archivo MyFilePreview.cpp

//Archivo frmUsuarios.ui.h
#include <qpixmap.h>
#include <qdialog.h>
#include <qfiledialog.h>
#include <qlineedit.h>
#include <qsqlquery.h>
#include <qinputdialog.h>
#include <qbuffer.h>
#include <qcstring.h>
#include <qcombobox.h>
#include "MyPixmapLabel.h"
#include "MyFilePreview.h"
#define ALLOC 0
#include "globales.h"

void frmUsuarios::BloquearControles(bool activado)
{
    //Bloquear/desbloquear controles
    lineEditNombre->setReadOnly(!activado);
    lineEditContrasenya->setReadOnly(!activado);
    myPixmapLabelFoto->setAcceptDrops(activado);
    checkBoxTipo->setEnabled(activado);
    //Bloquear/desbloquear el boton para cambiar la foto
    pushButtonCambiarImagen->setEnabled(activado);
}

void frmUsuarios::BloquearBotones( bool activado )
{
    //Bloquear/desbloquear los botones de desplazamiento
    pushButtonPrimero->setEnabled(activado);
    pushButtonAnterior->setEnabled(activado);
    pushButtonSiguiete->setEnabled(activado);
    pushButtonUltimo->setEnabled(activado);

    //Bloquear/desbloquear los botones de modificacion de registros
    pushButtonNuevo->setEnabled(activado);
    pushButtonEditar->setEnabled(activado);
    pushButtonBorrar->setEnabled(activado);
    pushButtonGuardar->setEnabled(activado);
    pushButtonCancelar->setEnabled(activado);
    pushButtonBuscar->setEnabled(activado);
}

bool frmUsuarios::BaseDatosVacía()
{

```



```
//Verificar si existen registros en la base de datos
if(query->size(>0)
{
    BloquearControles(FALSE);//Bloquear los controles
    BloquearBotones(TRUE);//Desbloquear los botones
        //Desactivar los botones guardar y cancelar
    pushButtonGuardar->setEnabled(FALSE);
    pushButtonCancelar->setEnabled(FALSE);
    return FALSE;
}
else
{
    BloquearControles(FALSE);
    BloquearBotones(FALSE);
    pushButtonNuevo->setEnabled(TRUE);//Activar solo boton nuevo
    QMessageBox::information( this, tr("Usuarios"),tr("Base de datos
vacía"));
    return TRUE;
}
}

void frmUsuarios::CargarDatos()
{
    //Cargar los datos de la base de datos a la aplicacion
    lineEditNombre->setText(query->value(0).toString());
    lineEditContrasenya->setText(query->value(1).toString());
    checkBoxTipo->setChecked(query->value(2).toBool());
    myPixmapLabelFoto->setPixmap(query->value(3).toByteArray());
}

void frmUsuarios::LimpiarControles()
{
    //Limpiar los datos de los controles
    lineEditNombre->setText("");
    lineEditContrasenya->setText("");
    checkBoxTipo->setChecked(false);
    myPixmapLabelFoto->setPixmap(QPixmap());
}

void frmUsuarios::CambiarImagen()
{
    //Tipos de imagenes soportadas
    QString types("Image files (*.jpeg *.pbm *.pgm *.png *.xbm *.ppm *.jpg
*.bmp);");
    QFileDialog *fd=new QFileDialog();//Crear un nuevo dialogo de navegacion
de ficheros
    fd->setFilters( types );//Establecer los filtros deseados
    MyFilePreview *p = new MyFilePreview;//Crear la vista previa a utilizar
    fd->setContentsPreviewEnabled( TRUE );//Activar el uso de vista previa
    fd->setContentsPreview( p,p );//Indicamos que la vista previa la
mostraremos en p y que p es la vista previa ya que esta derivado de QLabel y
QFilePreview
    fd->setPreviewMode( QFileDialog::Contents );//Indicamos que deseamos usar
la vista previa del fichero actual seleccionado en el dialogo de navegacion de
ficheros
```

```
//Mostrar el dialogo de navegacion de ficheros y verificar si se
selecciono alguna imagen
if(fd->exec()==QDialog::Accepted)
    myPixmapLabelFoto->setPixmap(QPixmap(fd->selectedFile()));
}

//Funcion que se llama al iniciar el formulario
void frmUsuarios::init()
{

    query=new QSqlQuery();
    query->exec("SELECT * FROM usuarios");//Seleccionar todos los registros

    /*Si la base de datos no esta vacia nos posicionamos en el siguiente
registro valido ya que luego de realizar la consulta nos pone en la posicion -
1. Y cargamos los campos del mismo*/
    if(!BaseDatosVacía())
    {
        query->next();
        CargarDatos();
    }

    //Establecer el modo de entrada contraseña para el control
correspondiente
    lineEditContrasenya->setEchoMode(QLineEdit::Password);

    //Indicamos que no estamos editando un registro existente
    editando=FALSE;
}

//Funcion que se llama al finalizar el formulario
void frmUsuarios::done(int r)
{
    //Verificamos si ha ocurrido un error en la ejecucion de la aplicacion e
indicamos con un mensaje
    if(r==1)
        QMessageBox::critical( this, tr("Usuarios"),tr("Abortando
aplicacion..."));
    QDialog::done(r);
}

void frmUsuarios::MoverPrimero()
{
    //Verificamos si estamos en el primer registro
    if(query->at()==0)
        return;

    //Tratar de movernos al primer registro y cargar los datos
    if(query->first())
        CargarDatos();
    else
    {
        QMessageBox::critical( this,tr("Usuarios"),tr("Error al moverse al
primer registro"));
    }
}
```

```
        done(1);
    }
}

void frmUsuarios::MoverAnterior()
{
    //Verificamos si estamos en el primer registro
    if(query->at()==0)
    {
        QMessageBox::information( this, tr("Usuarios"),tr("Este es el primer
registro"));
        return;
    }

    //Tratar de movernos al registro anterior y cargar los datos
    if(query->prev())
        CargarDatos();
    else
    {
        QMessageBox::critical( this, tr("Usuarios"),tr("Error al moverse al
registro anterior"));
        done(1);
    }
}

void frmUsuarios::MoverSiguiente()
{
    //Verificamos si estamos en el ultimo registro
    if(query->at()==query->size()-1)
    {
        QMessageBox::information( this, tr("Usuarios"),tr("Este es el ultimo
registro"));
        return;
    }

    //Tratar de movernos al registro siguiente y cargar los datos
    if(query->next())
        CargarDatos();
    else
    {
        QMessageBox::critical( this, tr("Usuarios"),tr("Error al moverse al
registro siguiente"));
        done(1);
    }
}

void frmUsuarios::MoverUltimo()
{
    //Verificamos si estamos en el primer registro
    if(query->at()==query->size()-1)
        return;

    //Tratar de movernos al primer registro y cargar los datos
    if(query->last())
        CargarDatos();
}
```

```
        else
        {
            QMessageBox::critical( this,tr("Usuarios"),tr("Error al moverse al
ultimo registro"));
            done(1);
        }
    }

void frmUsuarios::Nuevo()
{
    LimpiarControles();

    editando=FALSE;//Indicando que no estamos editando un registro existente

    //Desbloquear los controles, bloquear los botones y dejar solo activados
acceptary cancelar
    BloquearControles(TRUE);
    BloquearBotones(FALSE);
    pushButtonGuardar->setEnabled(TRUE);
    pushButtonCancelar->setEnabled(TRUE);
}

void frmUsuarios::Editar()
{
    nombre_editando=lineEditNombre->text();//Guardar el nombre original del
registro actual
    pos_editando=query->at();//Guardar la posicion del registro actual
    editando=TRUE;//Indicando que estamos editando un registro existente

    //Desbloquear los controles, bloquear los botones y dejar solo activados
acceptary cancelar
    BloquearControles(TRUE);
    BloquearBotones(FALSE);
    pushButtonGuardar->setEnabled(TRUE);
    pushButtonCancelar->setEnabled(TRUE);
}

void frmUsuarios::Borrar()
{
    //Preguntar antes de borrar el registro actual
    if(QMessageBox::question(this, tr("Borrar registro actual -- Usuarios"),
        tr("Una vez borrado no podra deshacer los cambios \n Desea borrar el
usuario "+lineEditNombre->text()+"?"),
        QMessageBox::Yes|QMessageBox::Default,
        QMessageBox::No|QMessageBox::Escape ) ==QMessageBox::Yes)
    {
        int posicion_actual=query->at();//Guardar la posicion del registro
actual
        //Tratar de borrar el registro actual
        query->exec("SELECT nombre FROM usuarios WHERE tipo=1");
        if(query->size()==1)
        {
            query->next();
            if(query->value(0).toString()==lineEditNombre->text())
            {
```



```
        QMessageBox::information( this,tr("Borrado --Usuarios"),"No puede
borrar este usuario porque es el unico administrador!!!");
        return;
    }
}
if(query->exec("DELETE FROM usuarios WHERE nombre='"+lineEditNombre-
>text()+"'"))
{
    QMessageBox::information( this,tr("Borrado --Usuarios"),tr("Registro
borrado correctamente!!!"));

    query->exec("SELECT * FROM usuarios");//Seleccionar todos los
registros

    /*Si la base de datos no esta vacia nos posicionamos en el registro
anterior al borrado, si borramos el ultimo elemento dejamos los controles
limpios*/
    if(!BaseDatosVacia())
    {
        //Verificamos si existe un nuevo registro en la posicion que
teniamos
        if(posicion_actual<=query->size())
        {
            if(posicion_actual>0)
                query->seek(posicion_actual-1);
            else
                query->seek(posicion_actual);
            CargarDatos();
        }
    }
    else
        LimpiarControles();
}
else
{
    QMessageBox::critical( this,tr("Borrado --Usuarios"),tr("Error al
borrar el registro actual"));
}
}
}

void frmUsuarios::Guardar()
{
    //Verificar si los datos estan completos
    if(lineEditNombre->text()=="")
    {
        QMessageBox::critical( this,tr("Verifique --Usuarios"),tr("El campo
nombre esta vacio"));
        return;
    }

    if(lineEditContrasenya->text()=="")
    {
        QMessageBox::critical( this,tr("Verifique --Usuarios"),tr("El campo
clave esta vacio"));
    }
}
```



```
        return;
    }

    if(myPixmapLabelFoto->pixmap()->isNull())
    {
        if(QMessageBox::question(this, tr("Verifique -- Usuarios"),
        tr("La imagen esta vacia, desea dejarla asi?"),
        QMessageBox::Yes|QMessageBox::Default,
        QMessageBox::No|QMessageBox::Escape ) ==QMessageBox::No)
            return;
    }

    //Verificamos que el usuario no exista
    query->exec("SELECT nombre FROM usuarios where nombre='"+lineEditNombre-
>text()+"'");

    if(query->next())
    {
        //Verifico si estoy insertando un usuario nuevo
        if(!editando)
        {
            QMessageBox::critical( this, "Usuarios","Ese usuario ya existe" );
            return;
        }

        //Verifico si estoy editando ese usuario
        if(editando && query->value(0).toString() != nombre_editando)
        {
            QMessageBox::critical( this, "Usuarios","Ese usuario ya existe" );
            return;
        }
    }

    QByteArray ba_foto;//Maneja la foto

    if(!myPixmapLabelFoto->pixmap()->isNull())
    {
        QBuffer buf(ba_foto);
        buf.open(IO_WriteOnly);
        if (not myPixmapLabelFoto->pixmap()->save(&buf,"PNG" ) )
        {
            QMessageBox::information( this, tr("Usuarios"),tr("error al guardar
en el buffer"));
            return;
        }
        buf.close();
    }

    //Verificamos si estamos editando un registro existente o guardamos uno
nuevo
    if(editando)
    {
        //Actualizamos el registro existente
        query->prepare("UPDATE usuarios SET nombre=?,password=?,tipo=?,foto=?
WHERE nombre='"+lineEditNombre->text()+"'");
```

```
query->bindValue(0,lineEditNombre->text());
query->bindValue(1,lineEditContrasenya->text());
query->bindValue(2,checkBoxTipo->isChecked());
query->bindValue(3,ba_foto);
query->exec();

query->exec("SELECT * FROM usuarios");//Recargar los registros
if(!BaseDatosVacia())
{
    query->seek(pos_editando);//Movernos al registro que acabamos de
editar
    CargarDatos();
}
editando=FALSE;
}
else
{
    //Insertamos el nuevo registro
query->prepare("INSERT INTO usuarios (nombre,password,tipo,foto) VALUES
(?,?,?,?)");
query->bindValue(0,lineEditNombre->text());
query->bindValue(1,lineEditContrasenya->text());
query->bindValue(2,checkBoxTipo->isChecked());
query->bindValue(3,ba_foto);
query->exec();

query->exec("SELECT * FROM usuarios");//Recargar los registros
if(!BaseDatosVacia())
MoverUltimo();//Movernos al registro que acabamos de insertar
}
QMessageBox::information( this, "Hecho -- Usuarios","Usuario guardado
correctamente" );
}

void frmUsuarios::Cancelar()
{
    query->exec("SELECT * FROM usuarios");//Recargar los registros
    if(BaseDatosVacia())
        LimpiarControles();
    else
    {
        if(editando)
            query->seek(pos_editando);//Recargamos el registro sin cambios
        else
            MoverUltimo();
        CargarDatos();
    }
}

void frmUsuarios::Buscar()
{
    bool ok;
    QString nombre_buscar = QDialog::getText(tr("Buscar --Usuarios"),
tr("Escriba el nombre del usuario a buscar:"), QLineEdit::Normal,
QString::null, &ok, this );
```



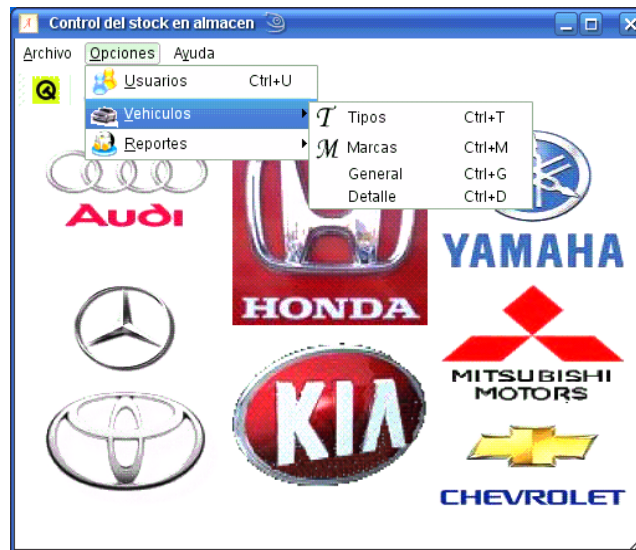
```
if ( ok && !nombre_buscar.isEmpty() )
{
    query->seek(-1);
    BuscarSiguiente:
    while(query->next())
        if(query->value(0).toString().contains(nombre_buscar,FALSE))
        {
            CargarDatos();
            //Preguntar si este es el usuario buscado
            if(QMessageBox::question(this, tr("Buscar -- Usuarios"),
            tr("Buscar
siguiente?"),QMessageBox::Yes|QMessageBox::Default,QMessageBox::No|QMessageBox
::Escape)==QMessageBox::Yes)
                goto BuscarSiguiente;
            else
                return;
        }

        query->last();
        CargarDatos();
        QMessageBox::information( this, "Buscar -- Usuarios","Usuario no
encontrado");
    }
    else
        return;
}
//Fin archivo frmUsuarios.ui.h
```



Dentro del menú Opciones encontramos el submenú Vehículos. Este submenú nos sirve para mostrar los siguientes formularios:

- Tipos
- Marcas
- General
- Detalle



El formulario Tipo verifica si la base de datos de tipos esta vacía y nos lo indica mediante un mensaje...



Luego de la verificación nos muestra el formulario Tipos mediante el cual podemos realizar todas las opciones de manipulación del mismo.



### Formulario frmTipos

```
//Archivo frmTipos.ui.h
#include <qdialog.h>
#include<qlabel.h>
#include<qlineedit.h>
#include<qinputdialog.h>
#include<qmessagebox.h>
#include <mysql.h>
#define ALLOC 0
#include "globales.h"
```

```
void frmTipos::BloquearControles(bool activado)
{
    //Bloquear/desbloquear controles
    lineEditTipo->setReadOnly(!activado);
```



```
        textEditDesc->setReadOnly(!activado);
    }

void frmTipos::BloquearBotones( bool activado )
{
    //Bloquear/desbloquear los botones de desplazamiento
    pushButtonPrimero->setEnabled(activado);
    pushButtonAnterior->setEnabled(activado);
    pushButtonSiguiente->setEnabled(activado);
    pushButtonUltimo->setEnabled(activado);

    //Bloquear/desbloquear los botones de modificacion de registros
    pushButtonNuevo->setEnabled(activado);
    pushButtonEditar->setEnabled(activado);
    pushButtonBorrar->setEnabled(activado);
    pushButtonGuardar->setEnabled(activado);
    pushButtonCancelar->setEnabled(activado);
    pushButtonBuscar->setEnabled(activado);
}

bool frmTipos::BaseDatosVacía()
{
    //Verificar si existen registros en la base de datos
    if(query->size()>0)
    {
        BloquearControles(FALSE); //Bloquear los controles
        BloquearBotones(TRUE); //Desbloquear los botones
        //Desactivar los botones guardar y cancelar
        pushButtonGuardar->setEnabled(FALSE);
        pushButtonCancelar->setEnabled(FALSE);
        return FALSE;
    }
    else
    {
        BloquearControles(FALSE);
        BloquearBotones(FALSE);
        pushButtonNuevo->setEnabled(TRUE); //Activar solo boton nuevo
        QMessageBox::information( 0, tr("Tipos"),tr("Base de datos vacía"));
        LimpiarControles();
        return TRUE;
    }
}

void frmTipos::CargarDatos()
{
    //Cargar los datos de la base de datos a la aplicacion
    lineEditId->setText(query->value(0).toString());
    lineEditTipo->setText(query->value(1).toString());
    textEditDesc->setText(query->value(2).toString());
}

void frmTipos::LimpiarControles()
{
    //Limpiar los datos de los controles
    lineEditId->setText("");
}
```



```
        lineEditTipo->setText("");
        textEditDesc->setText("");
    }

//Funcion que se llama al iniciar el formulario
void frmTipos::init()
{
    query=new QSqlQuery();
    query->exec("SELECT * FROM tipos_v");//Seleccionar todos los registros

    /*Si la base de datos no esta vacia nos posicionamos en el siguiente
registro valido ya que luego de realizar la consulta nos pone en la posicion -
1. Y cargamos los campos del mismo*/
    if(!BaseDatosVacía())
    {
        query->next();
        CargarDatos();
    }

    //Indicamos que no estamos editando un registro existente
    editando=FALSE;
}

//Funcion que se llama al finalizar el formulario
void frmTipos::done(int r)
{
    //Verificamos si ha ocurrido un error en la ejecucion de la aplicacion e
indicamos con un mensaje
    if(r==1)
        QMessageBox::critical( this, tr("Tipos"),tr("Abortando aplicacion..."));
    else
        r=0;
    QDialog::done(r);
}

void frmTipos::MoverPrimero()
{
    //Verificamos si estamos en el primer registro
    if(query->at()==0)
        return;

    //Tratar de movernos al primer registro y cargar los datos
    if(query->first())
        CargarDatos();
    else
    {
        QMessageBox::critical( this,tr("Tipos"),tr("Error al moverse al primer
registro"));
        done(1);
    }
}

void frmTipos::MoverAnterior()
{
    //Verificamos si estamos en el primer registro
```

```
        if(query->at()==0)
        {
            QMessageBox::information( this, tr("Tipos"),tr("Este es el primer
registro"));
            return;
        }

        //Tratar de movernos al registro anterior y cargar los datos
        if(query->prev())
            CargarDatos();
        else
        {
            QMessageBox::critical( this,tr("Tipos"),tr("Error al moverse al registro
anterior"));
            done(1);
        }
    }

void frmTipos::MoverSiguiente()
{
    //Verificamos si estamos en el ultimo registro
    if(query->at()==query->size()-1)
    {
        QMessageBox::information( this,tr("Tipos"),tr("Este es el ultimo
registro"));
        return;
    }

    //Tratar de movernos al registro siguiente y cargar los datos
    if(query->next())
        CargarDatos();
    else
    {
        QMessageBox::critical( this, tr("Tipos"),tr("Error al moverse al
registrosiguiente"));
        done(1);
    }
}

void frmTipos::MoverUltimo()
{
    //Verificamos si estamos en el primer registro
    if(query->at()==query->size()-1)
        return;

    //Tratar de movernos al primer registro y cargar los datos
    if(query->last())
        CargarDatos();
    else
    {
        QMessageBox::critical( this,tr("Tipos"),tr("Error al moverse al ultimo
registro"));
        done(1);
    }
}
```

```
void frmTipos::Nuevo()
{
    LimpiarControles();

    editando=FALSE;//Indicando que no estamos editando un registro existente

    //Desbloquear los controles, bloquear los botones y dejar solo activados
    aceptary cancelar
    BloquearControles(TRUE);
    BloquearBotones(FALSE);
    pushButtonGuardar->setEnabled(TRUE);
    pushButtonCancelar->setEnabled(TRUE);

    lineEditTipo->setFocus();
}

void frmTipos::Editar()
{
    nombre_editando=lineEditId->text();//Guardar el nombre original del
registro actual
    pos_editando=query->at();//Guardar la posicion del registro actual
    editando=TRUE;//Indicando que estamos editando un registro existente

    //Desbloquear los controles, bloquear los botones y dejar solo activados
    aceptary cancelar
    BloquearControles(TRUE);
    BloquearBotones(FALSE);
    pushButtonGuardar->setEnabled(TRUE);
    pushButtonCancelar->setEnabled(TRUE);
}

void frmTipos::Borrar()
{
    //Preguntar antes de borrar el registro actual
    if(QMessageBox::question(this, tr("Borrar registro actual -- Tipos"),
        tr("Una vez borrado no podra deshacer los cambios \n Desea borrar el
tipo "+lineEditId->text()+"?"),
        QMessageBox::Yes|QMessageBox::Default,
        QMessageBox::No|QMessageBox::Escape ) ==QMessageBox::Yes)
    {
        int posicion_actual=query->at();//Guardar la posicion del registro
actual
        //Tratar de borrar el registro actual
        if(query->exec("DELETE FROM tipos_v WHERE id_tipo_v='"+lineEditId-
>text()+"'"))
        {
            QMessageBox::information( this,tr("Borrado --tipos"),tr("Registro
borrado correctamente!!!"));
            query->exec("SELECT * FROM tipos_v");//Seleccionar todos los
registros

            /*Si la base de datos no esta vacia nos posicionamos en el registro
anterior al borrado, si borramos el ultimo elemento dejamos los controles
limpios*/
```

```
        if(!BaseDatosVacia())
        {
            //Verificamos si existe un nuevo registro en la posicion que
teniamos
            if(posicion_actual<=query->size())
            {
                if(posicion_actual>0)
                    query->seek(posicion_actual-1);
                else
                    query->seek(posicion_actual);
                CargarDatos();
            }
        }
        else
            LimpiarControles();
    }
    else
    {
        QMessageBox::critical( this,tr("Borrado --tipos"),tr("Error al borrar
el registro actual"));
    }
}
}

void frmTipos::Guardar()
{
    //Verificar si los datos estan completos
    if(lineEditTipo->text()=="")
    {
        QMessageBox::critical( this,tr("Verifique --tipos"),tr("El campo tipo
esta vacio"));
        return;
    }

    //Verificamos que el tipo no exista
    query->exec("SELECT nombre FROM tipos_v where tipo_v='"+lineEditTipo-
>text()+"'");

    if(query->next())
    {
        //Verifico si estoy insertando un tipo nuevo
        if(!editando)
        {
            QMessageBox::critical( this, "Tipos","Ese tipo ya existe" );
            return;
        }

        //Verifico si estoy editando ese tipo
        if(editando && query->value(0).toString() != nombre_editando)
        {
            QMessageBox::critical( this, "Tipos","Ese tipo ya existe" );
            return;
        }
    }
}
```

```
//Verificamos si estamos editando un registro existente o guardamos uno
nuevo
if(editando)
{
    //Actualizamos el registro existente
    query->prepare("UPDATE tipos_v SET tipo_v=?,descripcion=? WHERE
id_tipo_v='"+lineEditId->text()+"'");
    query->bindValue(0,lineEditTipo->text());
    query->bindValue(1,textEditDesc->text());
    query->exec();

    query->exec("SELECT * FROM tipos_v");//Recargar los registros
    if(!BaseDatosVacia())
    {
editar
        query->seek(pos_editando);//Movernos al registro que acabamos de
            CargarDatos();
        }
        editando=FALSE;
    }
else
{
    //Insertamos el nuevo registro
    query->exec("INSERT INTO tipos_v (tipo_v,descripcion) VALUES (?,?)");
    query->bindValue(0,lineEditTipo->text());
    query->bindValue(1,textEditDesc->text());
    query->exec();

    query->exec("SELECT * FROM tipos_v");//Recargar los registros
    if(!BaseDatosVacia())
        MoverUltimo();//Movernos al registro que acabamos de insertar
    }
    QMessageBox::information( this, "Hecho -- Tipos","Tipo guardado
correctamente" );
}

void frmTipos::Cancelar()
{
    query->exec("SELECT * FROM tipos_v");//Recargar los registros
    if(BaseDatosVacia())
        LimpiarControles();
    else
    {
        if(editando)
            query->seek(pos_editando);//Recargamos el registro sin cambios
        else
            MoverUltimo();
        CargarDatos();
    }
}

void frmTipos::Buscar()
{
    bool ok;
```



```

QString tipo_buscar = QDialog::getText(tr("Buscar -- Tipos"),
    tr("Escriba el nombre del tipo a buscar:"), QLineEdit::Normal,
    QString::null, &ok, this );

if ( ok && !tipo_buscar.isEmpty() )
{
    query->seek(-1);
    BuscarSiguiente:
    while(query->next())
        if(query->value(1).toString().contains(tipo_buscar, FALSE))
        {
            CargarDatos();
            //Preguntar si este es el usuario buscado
            if(QMessageBox::question(this, tr("Buscar --Tipos"),
                tr("Buscar
siguiente?"), QMessageBox::Yes | QMessageBox::Default, QMessageBox::No | QMessageBox
::Escape) == QMessageBox::Yes)
                goto BuscarSiguiente;
            else
                return;
        }

        query->last();
        CargarDatos();
        QMessageBox::information( this, "Buscar -- Tipos", "Tipo no encontrado");
    }
    else
        return;
}

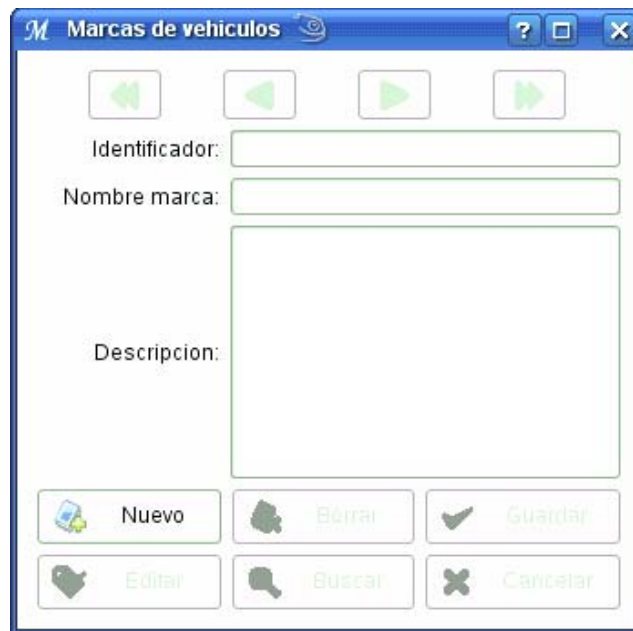
//Fin archivo frmTipos.ui.h

```

El siguiente formulario verifica si la base de datos de marcas esta vacia y lo indica mediante un mensaje.



Después de realizada la verificación se procede a mostrar el formulario en cual podemos realizar las modificaciones necesarias a la base de datos marcas.



### Formulario frmMarcas

```
//Archivo frmMarcas.ui.h
#include <qdialog.h>
#include<qlabel.h>
#include<qlineedit.h>
#include<qinputdialog.h>
#include<qmessagebox.h>
#include <qsqquery.h>
#define ALLOC 0
#include "globales.h"
```

```
void frmMarcas::BloquearControles(bool activado)
{
    //Bloquear/desbloquear controles
    lineEditMarca->setReadOnly(!activado);
    textEditDesc->setReadOnly(!activado);
}
```

```
void frmMarcas::BloquearBotones( bool activado )
{
```

```
//Bloquear/desbloquear los botones de desplazamiento
pushButtonPrimero->setEnabled(activado);
pushButtonAnterior->setEnabled(activado);
pushButtonSiguiente->setEnabled(activado);
pushButtonUltimo->setEnabled(activado);

//Bloquear/desbloquear los botones de modificacion de registros
pushButtonNuevo->setEnabled(activado);
pushButtonEditar->setEnabled(activado);
pushButtonBorrar->setEnabled(activado);
pushButtonGuardar->setEnabled(activado);
pushButtonCancelar->setEnabled(activado);
pushButtonBuscar->setEnabled(activado);
}

bool frmMarcas::BaseDatosVacía()
{
    //Verificar si existen registros en la base de datos
    if(query->size()>0)
    {
        BloquearControles(FALSE);//Bloquear los controles
        BloquearBotones(TRUE);//Desbloquear los botones
        //Desactivar los botones guardar y cancelar
        pushButtonGuardar->setEnabled(FALSE);
        pushButtonCancelar->setEnabled(FALSE);
        return FALSE;
    }
    else
    {
        BloquearControles(FALSE);
        BloquearBotones(FALSE);
        pushButtonNuevo->setEnabled(TRUE);//Activar solo boton nuevo
        QMessageBox::information( 0, tr("Marcas"),tr("Base de datos vacía"));
        LimpiarControles();
        return TRUE;
    }
}

void frmMarcas::CargarDatos()
{
    //Cargar los datos de la base de datos a la aplicacion
    lineEditId->setText(query->value(0).toString());
    lineEditMarca->setText(query->value(1).toString());
    textEditDesc->setText(query->value(2).toString());
}

void frmMarcas::LimpiarControles()
{
    //Limpiar los datos de los controles
    lineEditId->setText("");
    lineEditMarca->setText("");
    textEditDesc->setText("");
}

//Funcion que se llama al iniciar el formulario
```



```
void frmMarcas::init()
{
    query=new QSqlQuery();
    query->exec("SELECT * FROM marcas_v");//Seleccionar todos los registros

    /*Si la base de datos no esta vacia nos posicionamos en el siguiente
registro valido ya que luego de realizar la consulta nos pone en la posicion -
1. Y cargamos los campos del mismo*/
    if(!BaseDatosVacía())
    {
        query->next();
        CargarDatos();
    }

    //Indicamos que no estamos editando un registro existente
    editando=FALSE;
}

//Funcion que se llama al finalizar el formulario
void frmMarcas::done(int r)
{
    //Verificamos si ha ocurrido un error en la ejecucion de la aplicacion e
indicamos con un mensaje
    if(r==1)
        QMessageBox::critical( this, tr("Marcas"),tr("Abortando
aplicacion..."));
    else
        r=0;
    QDialog::done(r);
}

void frmMarcas::MoverPrimero()
{
    //Verificamos si estamos en el primer registro
    if(query->at()==0)
        return;

    //Tratar de movernos al primer registro y cargar los datos
    if(query->first())
        CargarDatos();
    else
    {
        QMessageBox::critical( this,tr("Marcas"),tr("Error al moverse al primer
registro"));
        done(1);
    }
}

void frmMarcas::MoverAnterior()
{
    //Verificamos si estamos en el primer registro
    if(query->at()==0)
    {
        QMessageBox::information( this, tr("Marcas"),tr("Este es el primer
registro"));
    }
}
```

```
        return;
    }

    //Tratar de movernos al registro anterior y cargar los datos
    if(query->prev())
        CargarDatos();
    else
    {
        QMessageBox::critical( this,tr("Marcas"),tr("Error al moverse al
registro anterior"));
        done(1);
    }
}

void frmMarcas::MoverSiguiente()
{
    //Verificamos si estamos en el ultimo registro
    if(query->at()==query->size()-1)
    {
        QMessageBox::information( this,tr("Marcas"),tr("Este es el ultimo
registro"));
        return;
    }

    //Tratar de movernos al registro siguiente y cargar los datos
    if(query->next())
        CargarDatos();
    else
    {
        QMessageBox::critical( this, tr("Marcas"),tr("Error al moverse al
registrosiguiente"));
        done(1);
    }
}

void frmMarcas::MoverUltimo()
{
    //Verificamos si estamos en el primer registro
    if(query->at()==query->size()-1)
        return;

    //Tratar de movernos al primer registro y cargar los datos
    if(query->last())
        CargarDatos();
    else
    {
        QMessageBox::critical( this,tr("Marcas"),tr("Error al moverse al ultimo
registro"));
        done(1);
    }
}

void frmMarcas::Nuevo()
{
    LimpiarControles();
}
```

```
editando=FALSE;//Indicando que no estamos editando un registro existente

//Desbloquear los controles, bloquear los botones y dejar solo activados
acceptary cancelar
BloquearControles(TRUE);
BloquearBotones(FALSE);
pushButtonGuardar->setEnabled(TRUE);
pushButtonCancelar->setEnabled(TRUE);

lineEditMarca->setFocus();
}

void frmMarcas::Editar()
{
    nombre_editando=lineEditId->text();//Guardar el nombre original del
registro actual
    pos_editando=query->at();//Guardar la posicion del registro actual
    editando=TRUE;//Indicando que estamos editando un registro existente

//Desbloquear los controles, bloquear los botones y dejar solo activados
acceptary cancelar
BloquearControles(TRUE);
BloquearBotones(FALSE);
pushButtonGuardar->setEnabled(TRUE);
pushButtonCancelar->setEnabled(TRUE);
}

void frmMarcas::Borrar()
{
//Preguntar antes de borrar el registro actual
if(QMessageBox::question(this, tr("Borrar registro actual -- Marcas"),
tr("Una vez borrado no podra deshacer los cambios \n Desea borrar el
Marca "+lineEditId->text()+"?"),
QMessageBox::Yes|QMessageBox::Default,
QMessageBox::No|QMessageBox::Escape ) ==QMessageBox::Yes)
{
    int posicion_actual=query->at();//Guardar la posicion del registro
actual
//Tratar de borrar el registro actual
if(query->exec("DELETE FROM marcas_v WHERE id_marca_v="+lineEditId-
>text()))
{
    QMessageBox::information( this,tr("Borrado --Marcas"),tr("Registro
borrado correctamente!!!"));
    query->exec("SELECT * FROM marcas_v");//Seleccionar todos los
registros

/*Si la base de datos no esta vacia nos posicionamos en el registro
anterior al borrado, si borramos el ultimo elemento dejamos los controles
limpios*/
    if(!BaseDatosVacía())
    {
        //Verificamos si existe un nuevo registro en la posicion que
teniamos
```

```
        if(posicion_actual<=query->size())
        {
            if(posicion_actual>0)
                query->seek(posicion_actual-1);
            else
                query->seek(posicion_actual);
            CargarDatos();
        }
    }
    else
        LimpiarControles();
}
else
{
    QMessageBox::critical( this,tr("Borrado --Marcas"),tr("Error al borrar
el registro actual"));
}
}
}

void frmMarcas::Guardar()
{
    //Verificar si los datos estan completos
    if(lineEditMarca->text()=="")
    {
        QMessageBox::critical( this,tr("Verifique --Marcas"),tr("El campo Marca
esta vacio"));
        return;
    }

    //Verificamos que el Marca no exista
    query->exec("SELECT nombre FROM marcas_v where marca_v='"+lineEditMarca-
>text()+"'");

    if(query->next())
    {
        //Verifico si estoy insertando un Marca nuevo
        if(!editando)
        {
            QMessageBox::critical( this, "Marcas","Ese Marca ya existe" );
            return;
        }

        //Verifico si estoy editando ese Marca
        if(editando && query->value(0).toString() != nombre_editando)
        {
            QMessageBox::critical( this, "Marcas","Ese Marca ya existe" );
            return;
        }
    }

    //Verificamos si estamos editando un registro existente o guardamos uno
nuevo
    if(editando)
```



```
{
    //Actualizamos el registro existente
    query->prepare("UPDATE marcas_v SET marca_v=?,descripcion=? WHERE
id_marca_v='"+lineEditId->text()+"'");
    query->bindValue(0,lineEditMarca->text());
    query->bindValue(1,textEditDesc->text());
    query->exec();

    query->exec("SELECT * FROM marcas_v");//Recargar los registros
    if(!BaseDatosVacia())
    {
editar      query->seek(pos_editando);//Movernos al registro que acabamos de
            CargarDatos();
    }
    editando=FALSE;
}
else
{
    //Insertamos el nuevo registro
    query->prepare("INSERT INTO marcas_v (marca_v,descripcion) VALUES
(?,?)");
    query->bindValue(0,lineEditMarca->text());
    query->bindValue(1,textEditDesc->text());
    query->exec();

    query->exec("SELECT * FROM marcas_v");//Recargar los registros
    if(!BaseDatosVacia())
        MoverUltimo();//Movernos al registro que acabamos de insertar
}
    QMessageBox::information( this, "Hecho -- Marcas","Marca guardado
correctamente" );
}

void frmMarcas::Cancelar()
{
    query->exec("SELECT * FROM marcas_v");//Recargar los registros
    if(BaseDatosVacia())
        LimpiarControles();
    else
    {
        if(editando)
            query->seek(pos_editando);//Recargamos el registro sin cambios
        else
            MoverUltimo();
        CargarDatos();
    }
}

void frmMarcas::Buscar()
{
    bool ok;
    QString marca_buscar = QDialog::getText(tr("Buscar -- Marcas"),
tr("Escriba el nombre de la marca a buscar:"), QLineEdit::Normal,
QString::null, &ok, this );
```



```

if ( ok && !marca_buscar.isEmpty() )
{
    query->seek(-1);
BuscarSiguiente:
    while(query->next())
        if(query->value(1).toString().contains(marca_buscar,FALSE))
        {
            CargarDatos();
            //Preguntar si este es el usuario buscado
            if(QMessageBox::question(this, tr("Buscar --Marcas"),
            tr("Buscar
siguiente?"),QMessageBox::Yes|QMessageBox::Default,QMessageBox::No|QMessageBox
::Escape)==QMessageBox::Yes)
                goto BuscarSiguiente;
            else
                return;
        }

        query->last();
        CargarDatos();
        QMessageBox::information( this, "Buscar -- Marcas","Marca no
encontrado");
    }
    else
        return;
}
//Fin archivo frmMarcas.ui.h

```

La opción General muestra el formulario Vehículos siempre que existan tipos y marcas en la base de datos de lo contrario invocará al formulario Tipos o Marcas según cual base de datos esta vacía.



**Formulario frmVehiculos**

Con este formulario podemos agregar, modificar y eliminar clasificaciones generales de vehículos así como buscar un identificador de vehículo específico.

```
//Archivo frmVehiculos.ui.h
#include <qdialog.h>
#include<qmessagebox.h>
#include<qlabel.h>
#include<qlineedit.h>
#include<qcombobox.h>
#include <qsqlquery.h>
#include <qsqldatabase.h>
#include<qinputdialog.h>
#define ALLOC 0
#include "globales.h"
#include "frmTipos.h"
#include "frmMarcas.h"

void frmVehiculos::BloquearControles(bool activado)
{
    //Bloquear/desbloquear controles
    lineEditId->setEnabled(activado);
    comboBoxTipo->setEnabled(activado);
    comboBoxMarca->setEnabled(activado);
}

void frmVehiculos::BloquearBotones( bool activado )
{
    //Bloquear/desbloquear los botones de desplazamiento
    pushButtonPrimero->setEnabled(activado);
    pushButtonAnterior->setEnabled(activado);
    pushButtonSiguiente->setEnabled(activado);
    pushButtonUltimo->setEnabled(activado);

    //Bloquear/desbloquear los botones de modificacion de registros
    pushButtonNuevo->setEnabled(activado);
    pushButtonEditar->setEnabled(activado);
    pushButtonBorrar->setEnabled(activado);
    pushButtonGuardar->setEnabled(activado);
    pushButtonCancelar->setEnabled(activado);
    pushButtonBuscar->setEnabled(activado);
}

bool frmVehiculos::BaseDatosVacía()
{
    //Verificar si existen registros en la base de datos
    if(query->size(>0)
    {
        BloquearControles(FALSE);//Bloquear los controles
        BloquearBotones(TRUE);//Desbloquear los botones
        //Desactivar los botones guardar y cancelar
        pushButtonGuardar->setEnabled(FALSE);
        pushButtonCancelar->setEnabled(FALSE);
    }
}
```

```
        return FALSE;
    }
    else
    {
        BloquearControles(FALSE);
        BloquearBotones(FALSE);
        pushButtonNuevo->setEnabled(TRUE); //Activar solo boton nuevo
        QMessageBox::information( 0, tr("Vehiculos"),tr("Base de datos vacia"));
        return TRUE;
    }
}

void frmVehiculos::CargarDatos()
{
    //Cargar los datos de la base de datos a la aplicacion
    lineEditId->setText(query->value(0).toString());
    comboBoxTipo->clear();
    comboBoxTipo->insertItem(query->value(3).toString());
    comboBoxTipo->setCurrentText(comboBoxTipo->text(0));
    comboBoxMarca->clear();
    comboBoxMarca->insertItem(query->value(4).toString());
    comboBoxMarca->setCurrentText(comboBoxMarca->text(0));
}

void frmVehiculos::LimpiarControles()
{
    //Limpiar los datos de los controles
    lineEditId->setText("");
    comboBoxTipo->setCurrentText("");
    comboBoxMarca->setCurrentText("");
}

void frmVehiculos::CargarCombos()
{
    query->exec("SELECT tipo_v FROM tipos_v");//Seleccionar los tipos
    existentes
    comboBoxTipo->clear();
    while(query->next())
        comboBoxTipo->insertItem(query->value(0).toString());

    query->exec("SELECT marca_v FROM marcas_v");//Seleccionar las marcas
    existentes
    comboBoxMarca->clear();
    while(query->next())
        comboBoxMarca->insertItem(query->value(0).toString());
}

//Funcion que se llama al iniciar el formulario
void frmVehiculos::init()
{
    query=new QSqlQuery();

    query->exec("SELECT tipo_v FROM tipos_v");//Seleccionar los tipos
    existentes
    if(!query->size())//Si no existen tipos de vehiculos...
```



```
{
    QMessageBox::information(this, tr("Vehiculos -- No hay tipos"),tr("No
    existen tipos de vehiculos, primero debe crear al menos uno"));
    QDialog::close();
    return;
}

    query->exec("SELECT marca_v FROM marcas_v");//Seleccionar las marcas
    existentes
    if(!query->size())//Si no existen marcas de vehiculos...
    {
        QMessageBox::information(this, tr("Vehiculos -- No hay marcas"),tr("No
        existen marcas de vehiculos, primero debe crear al menos una"));
        QDialog::close();
        return;
    }

    query->exec("SELECT id_v,v.id_tipo_v,v.id_marca_v,tipo_v,marca_v FROM
    vehiculos AS v INNER JOIN tipos_v AS t on v.id_tipo_v=t.id_tipo_v INNER JOIN
    marcas_v AS m on v.id_marca_v=m.id_marca_v");//Seleccionar todos los registros

    //Si la base de datos no esta vacia nos posicionamos en el siguiente
    registro valido ya que luego de realizar la consulta nos pone en la posicion -
    1. Y cargamos los campos del mismo
    if(!BaseDatosVacia())
    {
        query->next();
        CargarDatos();
    }

    //Indicamos que no estamos editando un registro existente
    editando=FALSE;
}

//Funcion que se llama al finalizar el formulario
void frmVehiculos::done(int r)
{
    //Verificamos si ha ocurrido un error en la ejecucion de la aplicacion e
    indicamos con un mensaje
    if(r==1)
        QMessageBox::critical( this, tr("Vehiculos"),tr("Abortando
    aplicacion..."));
    else
        r=0;

    QDialog::done(r);
}

void frmVehiculos::MoverPrimero()
{
    //Verificamos si estamos en el primer registro
    if(query->at()==0)
        return;

    //Tratar de movernos al primer registro y cargar los datos
```



```
        if(query->first())
            CargarDatos();
        else
        {
            QMessageBox::critical( this,tr("Vehiculos"),tr("Error al moverse al
primer registro"));
            done(1);
        }
    }

void frmVehiculos::MoverAnterior()
{
    //Verificamos si estamos en el primer registro
    if(query->at()==0)
    {
        QMessageBox::information( this, tr("Vehiculos"),tr("Este es el primer
registro"));
        return;
    }

    //Tratar de movernos al registro anterior y cargar los datos
    if(query->prev())
        CargarDatos();
    else
    {
        QMessageBox::critical( this,tr("Vehiculos"),tr("Error al moverse al
registro anterior"));
        done(1);
    }
}

void frmVehiculos::MoverSiguiente()
{
    //Verificamos si estamos en el ultimo registro
    if(query->at()==query->size()-1)
    {
        QMessageBox::information( this,tr("Vehiculos"),tr("Este es el ultimo
registro"));
        return;
    }

    //Tratar de movernos al registro siguiente y cargar los datos
    if(query->next())
        CargarDatos();
    else
    {
        QMessageBox::critical( this, tr("Vehiculos"),tr("Error al moverse al
registro siguiente"));
        done(1);
    }
}

void frmVehiculos::MoverUltimo()
{
    //Verificamos si estamos en el primer registro
```



```
if(query->at()==query->size()-1)
    return;

//Tratar de movernos al primer registro y cargar los datos
if(query->last())
    CargarDatos();
else
{
    QMessageBox::critical( this,tr("Vehiculos"),tr("Error al moverse al
ultimo registro"));
    done(1);
}
}

void frmVehiculos::Nuevo()
{
    LimpiarControles();
    CargarCombos();
    editando=FALSE;//Indicando que no estamos editando un registro existente

    //Desbloquear los controles, bloquear los botones y dejar solo activados
acceptary cancelar
    BloquearControles(TRUE);
    BloquearBotones(FALSE);
    pushButtonGuardar->setEnabled(TRUE);
    pushButtonCancelar->setEnabled(TRUE);
}

void frmVehiculos::Editar()
{
    pos_editando=query->at();//Guardar la posicion del registro actual
    editando=TRUE;//Indicando que estamos editando un registro existente
    QString t=comboBoxTipo->currentText();
    QString m=comboBoxMarca->currentText();

    //Desbloquear los controles, bloquear los botones y dejar solo activados
acceptary cancelar
    BloquearControles(TRUE);
    BloquearBotones(FALSE);
    pushButtonGuardar->setEnabled(TRUE);
    pushButtonCancelar->setEnabled(TRUE);
    CargarCombos();
    comboBoxTipo->setCurrentText(t);
    comboBoxMarca->setCurrentText(m);
}

void frmVehiculos::Borrar()
{
    //Preguntar antes de borrar el registro actual
    if(QMessageBox::question(this, tr("Borrar registro actual -- Vehiculos"),
        tr("Una vez borrado no podra deshacer los cambios \n Desea borrar el
Vehiculo "+lineEditId->text()+"?"),
        QMessageBox::Yes|QMessageBox::Default,
        QMessageBox::No|QMessageBox::Escape ) ==QMessageBox::Yes)
    {
```



```
int posicion_actual=query->at();//Guardar la posicion del registro
actual
//Tratar de borrar el registro actual
if(query->exec("DELETE FROM vehiculos WHERE id_v='"+lineEditId-
>text()+"'"))
{
    QMessageBox::information( this,tr("Borrado --Vehiculos"),tr("Registro
borrado correctamente!!!"));

    query->exec("SELECT id_v,v.id_tipo_v,v.id_marca_v,tipo_v,marca_v FROM
vehiculos AS v INNER JOIN tipos_v AS t on v.id_tipo_v=t.id_tipo_v INNER JOIN
marcas_v AS m on v.id_marca_v=m.id_marca_v");//Seleccionar todos los registros

    /*Si la base de datos no esta vacia nos posicionamos en el registro
anterior al borrado, si borramos el ultimo elemento dejamos los controles
limpios*/
    if(!BaseDatosVacía())
    {
        //Verificamos si existe un nuevo registro en la posicion que
teniamos
        if(posicion_actual<=query->size())
        {
            if(posicion_actual>0)
                query->seek(posicion_actual-1);
            else
                query->seek(posicion_actual);
            CargarDatos();
        }
    }
    else
        LimpiarControles();
}
else
{
    QMessageBox::critical( this,tr("Borrado --Vehiculos"),tr("Error al
borrar el registro actual"));
}
}
}

void frmVehiculos::Guardar()
{
    //Verificar si los datos estan completos
    if(comboBoxTipo->currentText()=="")
    {
        QMessageBox::critical( this,tr("Verifique --Vehiculos"),tr("El campo
tipo esta vacio"));
        return;
    }

    if(comboBoxMarca->currentText()=="")
    {
        QMessageBox::critical( this,tr("Verifique --Vehiculos"),tr("El campo
marca esta vacio"));
        return;
    }
}
```



```
}

//Verificamos que el Vehiculo no exista
query->prepare("SELECT id_v,tipo_v,marca_v FROM vehiculos AS v INNER JOIN
tipos_v AS t ON v.id_tipo_v=t.id_tipo_v INNER JOIN marcas_v AS m ON
v.id_marca_v=m.id_marca_v WHERE tipo_v=? AND marca_v=?");
query->bindValue(0,comboBoxTipo->currentText());
query->bindValue(1,comboBoxMarca->currentText());
query->exec();

if(query->next())
{
//Verifico si estoy insertando un Vehiculo nuevo
if(!editando)
{
QMessageBox::critical( this, "Vehiculos","Ese Vehiculo ya
existe");
return;
}

//Verifico si estoy editando ese Vehiculo
if(editando && query->value(0).toString() !=lineEditId->text())
{
QMessageBox::critical( this, "Vehiculos",QString("Ese Vehiculo ya
existe"));
return;
}
}

//Verificamos si estamos editando un registro existente o guardamos uno
nuevo
if(editando)
{
//Actualizamos el registro existente
query->prepare("UPDATE vehiculos SET id_tipo_v=(SELECT t.id_tipo_v from
tipos_v as t where t.tipo_v=?), id_marca_v=(SELECT m.id_marca_v FROM marcas_v
as m where m.marca_v=?) WHERE id_v=?");
query->bindValue(0,comboBoxTipo->currentText());
query->bindValue(1,comboBoxMarca->currentText());
query->bindValue(2,lineEditId->text().toInt());
query->exec();

query->exec("SELECT id_v,v.id_tipo_v,v.id_marca_v,tipo_v,marca_v FROM
vehiculos AS v INNER JOIN tipos_v AS t on v.id_tipo_v=t.id_tipo_v INNER JOIN
marcas_v AS m on v.id_marca_v=m.id_marca_v");//Recargar los registros
if(!BaseDatosVacia())
{
query->seek(pos_editando);//Movernos al registro que acabamos de
editar
CargarDatos();
}
editando=FALSE;
}
else
{
```





```
//Insertamos el nuevo registro
query->prepare("INSERT INTO vehiculos (id_tipo_v,id_marca_v)
VALUES((SELECT t.id_tipo_v from tipos_v as t where t.tipo_v=?),(SELECT
m.id_marca_v FROM marcas_v as m where m.marca_v=?))");
query->bindValue(0,comboBoxTipo->currentText());
query->bindValue(1,comboBoxMarca->currentText());
query->exec();

query->exec("SELECT id_v,v.id_tipo_v,v.id_marca_v,tipo_v,marca_v FROM
vehiculos AS v INNER JOIN tipos_v AS t on v.id_tipo_v=t.id_tipo_v INNER JOIN
marcas_v AS m on v.id_marca_v=m.id_marca_v");//Recargar los registros
if(!BaseDatosVacia())
MoverUltimo();//Movernos al registro que acabamos de insertar
}
QMessageBox::information( this, "Hecho -- Vehiculos","Vehiculo guardado
correctamente" );
}

void frmVehiculos::Cancelar()
{
query->exec("SELECT id_v,v.id_tipo_v,v.id_marca_v,tipo_v,marca_v FROM
vehiculos AS v INNER JOIN tipos_v AS t on v.id_tipo_v=t.id_tipo_v INNER JOIN
marcas_v AS m on v.id_marca_v=m.id_marca_v");//Recargar los registros
if(BaseDatosVacia())
LimpiaControles();
else
{
if(editando)
query->seek(pos_editando);//Recargamos el registro sin cambios
else
MoverUltimo();
CargarDatos();
}
}

void frmVehiculos::Buscar()
{
bool ok;
QString nombre_buscar = QDialog::getText(tr("Buscar --Vehiculos"),
tr("Escriba el id del Vehiculo a buscar:"), QLineEdit::Normal,
QString::null, &ok, this );

if ( ok && !nombre_buscar.isEmpty() )
{
query->seek(-1);
BuscarSiguiente:
while(query->next())
if(query->value(0).toInt()==nombre_buscar.toInt())
{
CargarDatos();
//Preguntar si este es el Vehiculo buscado
if(QMessageBox::question(this, tr("Buscar -- Vehiculos"),
tr("Buscar siguiente?"),QMessageBox::Yes|QMessageBox::Default,
QMessageBox::No|QMessageBox::Escape)==QMessageBox::Yes)
goto BuscarSiguiente;
}
}
}
```

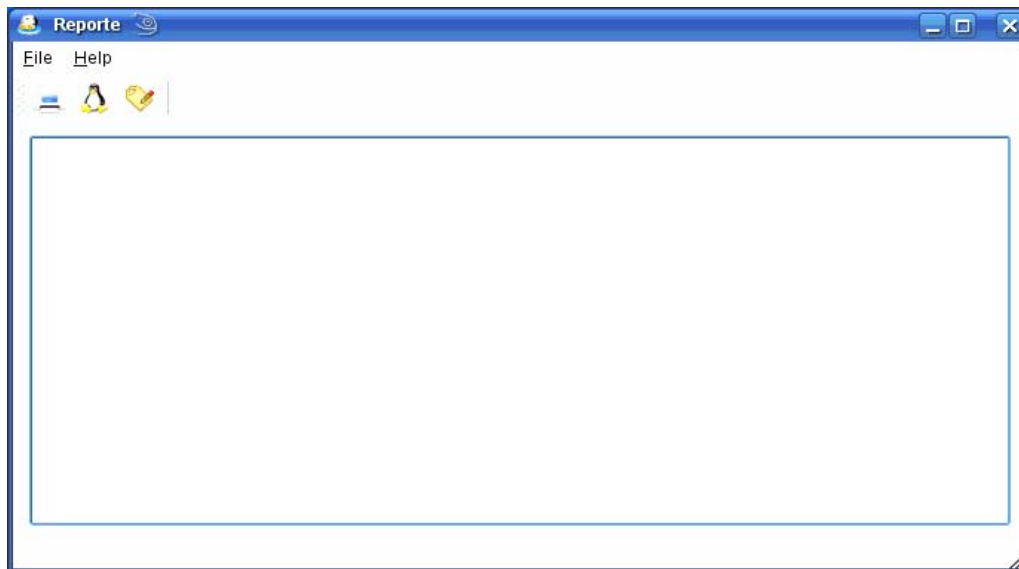
```

        else
            return;
    }

    query->last();
    CargarDatos();
    QMessageBox::information( this, "Buscar -- Vehiculos","Vehiculo no
encontrado");
}
else
    return;
}
//Fin archivo frmVehiculos.ui.h

```

Este formulario nos permite realizar reportes de cualquier consulta al sistema.



### Formulario frmReporte

```

//Archivo frmReporte.ui.h
#include <qmessagebox.h>
#include <qprinter.h>
#include <qpainter.h>
#include <qpaintdevicemetrics.h>
#include <qsimplerichtext.h>
#include <sqlquery.h>
#include <sqldatabase.h>
#include <sqlrecord.h>
#include <qvariant.h>
#include <qstatusbar.h>
#include <qdatetime.h>

```



```
#include <qvariant.h>
#include <qinputdialog.h>
#define ALLOC 0
#include "globales.h"

void frmReporte::filePrint()
{
    statusBar()->message("Imprimiendo...",2000);//Indicamos que empezamos a
    imprimir
    if ( !textEditReporte )
        return;
#ifdef QT_NO_PRINTER
    QPrinter printer;

    printer.setFullPage(TRUE);
    if ( printer.setup( this ) ) {
        QPainter p( &printer );
        // Verificar si existe un dispositivo valido para imprimir en el
        if ( !p.device() ) return;
        QPaintDeviceMetrics metrics( p.device() );

#ifdef Q_OS_WIN //Si el sistema operativo es Windows amplia 3 veces el
        painter
            p.scale(3.0,3.0);
#endif

        int dpix = metrics.logicalDpiX(); //Posicion donde empieza a dibujar
        en el eje X
        int dpiy = metrics.logicalDpiY();//Posicion donde empieza a dibujar en
        el eje Y

        const int margin = 72; // Margen del lado izquierdo en puntos por
        pulgada
        QRect body(margin * dpix / 72, margin * dpiy / 72,
            metrics.width() - margin * dpix / 72 * 2,
            metrics.height() - margin * dpiy / 72 * 2 );

        QFont font( "Sans Serif", 12 );
        QSimpleRichText richText( textEditReporte->text(), font,
            textEditReporte->context(), textEditReporte->styleSheet(),textEditReporte-
            >mimeSourceFactory(), body.height() );

        richText.setWidth( &p, body.width() );
        QRect view( body );
        int page = 1;
        do {
            richText.draw( &p, body.left(), body.top(), view, colorGroup() );

            view.moveBy( 0, body.height() );
            p.translate( 0 , -body.height() );
            //p.setFont( font );

            p.drawText( view.right() - p.fontMetrics().width( QString::number(
            page ) ),
```



```
        view.bottom() + p.fontMetrics().ascent() + 5,
QString::number( page ) );
    if ( view.top() >= richText.height() )
        break;
    printer.newPage();
    page++;
    statusBar()->message("Imprimiendo pagina:
"+QVariant(page).toString(),2000); //Indicamos que pagina estamos imprimiendo
    } while (TRUE);
}
#endif
    statusBar()->message("Impresion terminada",2000); //Indicamos que termino
la impresion
}

void frmReporte::fileExit()
{
    close(true);
}

void frmReporte::helpAbout()
{
    statusBar()->message("Acerca de Almacen...",2000);
    //Crear y mostrar un mensaje indicando quien construyo esta aplicacion :->
    QMessageBox::about(this,"Acerca de \"Almacen\"...", "<center><b>Elaborado
bajo licencia GPL por estudiantes de la Universidad Nacional Autonoma de
Nicaragua UNAN - Leon. 2007</b><br>Armando Rodriguez Lopez<br>Darling Maria
Sevilla <br>Mauricio Antonio Perez</center>");
}

/*El parametro 'externo' de la funcion generar es para saber si se usara una
consulta externa al reporte la cual utiliza la variable global 'query'*/
void frmReporte::generar(QString titulo,QString consulta ,int alto_foto,int
ancho_foto)
{
    statusBar()->message("Generando reporte...",2000); //Indicamos que empezamos
generar el reporte
    if(consulta=="") //Comprobar si existe una consulta para el reporte
    {
        QMessageBox::critical(this,"Consulta no valida","Debe ingresar una
consulta");
        return;
    }

    if(!consulta.left(6).contains("SELECT",FALSE)) //Comprobar si es una
consulta tipo SELECT
    {
        QMessageBox::critical(this,"Consulta no valida","Solo se permiten hacer
consultas tipo SELECT");
        return;
    }

    if(titulo=="") //Verificar si el reporte posee un titulo
        titulo="Reporte";
}
```



```
query->exec(consulta); //Ejecutar la consulta del reporte

QString datos; //Variable para almacenar los datos en formato html

if(query->size()<=0)
{
    if(QMessageBox::question(this,tr("Consulta vacia"),tr("No existen
registros en esta consulta.\n Desea crear el reporte de todas
maneras?"),tr("&Yes"), tr("&No"),QString::null, 0, 1 ))return;
    else
        datos+="Sin datos";
}

QVariant i; //Variable para auxiliarnos en la creacion del nombre de la
imagen
QString img_actual; //Nombre de la imagen actual
QMimeSourceFactory *imagenes=new QMimeSourceFactory(); //QMimeSourceFactory
que utilizaremos para almacenar todas las imagenes de la base de datos
QMimeSourceFactory *img; //QMimeSourceFactory que nos sirve para almacenar
la imagen actual que sera agregada a imagenes
//Creamos un QStyleSheetItem para cambiar el tipo de letra del encabezado
del reporte
QStyleSheetItem * mytag= new QStyleSheetItem( textEditReporte->styleSheet(),
"mytag");
extern QSqlDatabase *db; //Declaramos el puntero db como extern para poder
utilizarlo debido a que no existe en este ambito
int col; //Variable que nos permite recorrer las columnas de la consulta
extern int foto_actual;

mytag->setFontSize ( 20 ); //Cambiar el tamanyo de la fuente
mytag->setFontFamily ( "Sans Serif" ); //Cambiar el tipo de fuente

//A continuacion creamos una tabla en el textEditReporte para mostrar los
datos
datos+="<table border=1><tr>";
//Introducimos los encabezados de las columnas
for(col = 0; col < db->record(*query).count(); col++)
    datos+="<th><center>"+db->record(*query).field(col)-
>name()+"</center></th>";
datos+="</tr>";

statusBar()->message("Obteniendo datos...",2000);
while(query->next())
{
    i=foto_actual;
    datos+="<tr>";
    //Recorrer las columnas para extraer los valores de los campos
    for(col = 0; col < db->record(*query).count(); col++)
    {
        //Verificamos si este campo posee una foto
        if(db->record(*query).field(col)->name().contains("foto",FALSE))
        {
            if(QPixmap(query->value(col).toByteArray()).isNull())
                datos+="<td><center>Sin Foto</center></td >";
        }
    }
}
```

```
        else
        {
            img_actual="img"+i.toString();//Creamos el nombre de la imagen
actual
            img=new QMimeSourceFactory(); //Creamos un nuevo
QMimeSourceFactory para almacenar la imagen
            img->setPixmap(img_actual,QPixmap(query-
>value(col).toByteArray()));//Copiamos la imagen de la base de datos al
QMimeSourceFactory
            imagenes->addFactory(img);//Agregamos el nuevo QMimeSourceFactory
al QMimeSourceFactory que utilizaremos para el QSimpleRichText
            //Variable que guarda los datos necesarios para poner la imagen
actual en el textEditReporte
            QString img = "<img src=\""+img_actual+"\" alt=\"sin foto\"
width="+QVariant(ancho_foto).toString()+
height="+QVariant(alto_foto).toString()+"border=\"0\">";
            datos+="<td><center>"+img+"</center></td >";
        }
    }
    else //Si este campo no posee foto...
        datos+="<td><center>"+query->value(col).toString()+"</center></td
>";
    }
    datos+="</tr>";
    foto_actual++;
}

datos+="</table>";//Terminamos de dibujar la tabla

//Agregamos el titulo del reporte
textEditReporte->append("<center><mytag>"+titulo+"</mytag></center>");

//Agregamos la fecha del reporte
QDate fecha;

textEditReporte->append("<small>Fecha:
"+fecha.currentDate().toString("dd")+ " de "+
fecha.currentDate().toString("MMMM")+ " del
"+fecha.currentDate().toString("yyyy")+ "</small><hr>");

textEditReporte->append("<center>"+datos+"</center>");
//Establecemos como QMimeSource de textEditReporte imagenes que posee todas
las imagenes de los registros.
textEditReporte->setMimeSourceFactory(imagenes);
statusBar()->message("Hecho!!!",2000);
}

void frmReporte::EditarReporte()
{
    statusBar()->message("Editando reporte...",2000);
    if(!tipo_usuario)//Si el usuario no es administrador no se permite esta
opcion
    {
        QMessageBox::critical( 0, tr("Almacen"),tr("Error: Usted no es
administrador \n No puede utilizar esta opcion"));
    }
}
```

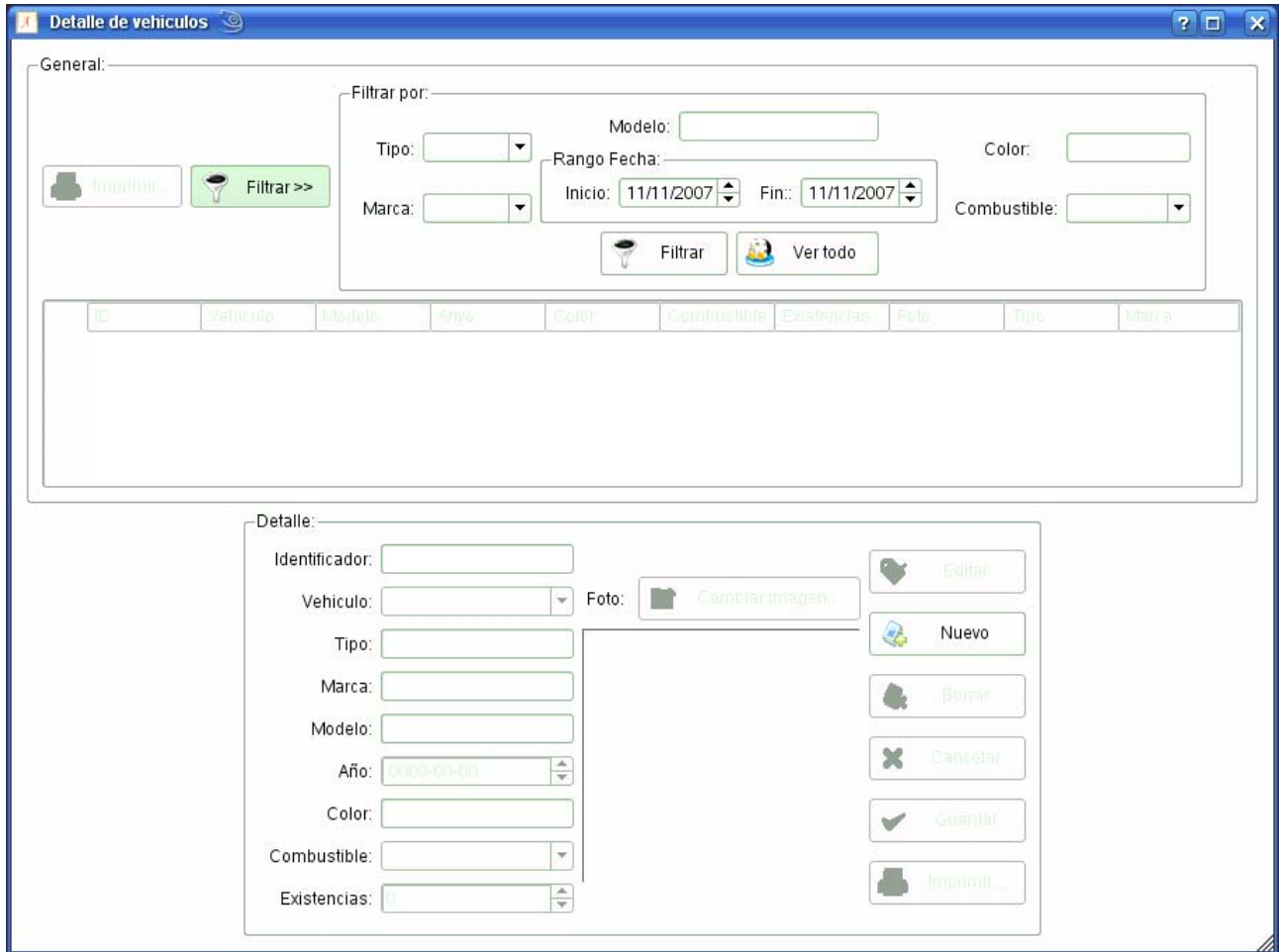


```
        return;
    }
    bool ok=false;
    //Cambiar el titulo
    QString titulo=QInputDialog::getText("Personalizar titulo", "Escriba el
    titulo del reporte:", QLineEdit::Normal,QString::null, 0, this );

    //Cambiar la consulta
    QString consulta=QInputDialog::getText("Personalizar consulta", "Escriba la
    consulta del reporte:", QLineEdit::Normal,QString::null, &ok, this );

    if ( ok )
    {
        textEditReporte->setText("");
        generar(titulo,consulta,30,40);
    }
    else
        return;
}
//Fin archivo frmReporte.ui.h
```

La última opción de este submenú nos muestra el formulario DetalleV verificando previamente la existencia de tipos y marcas en la base de datos.



The screenshot shows a software interface for vehicle management. The main window is titled "Detalle de vehiculos".

**General Section:**

- Buttons: Imprimir, Filtrar >>
- Filter by (Filtrar por):
  - Tipo: [Dropdown]
  - Modelo: [Text Input]
  - Color: [Text Input]
  - Marca: [Dropdown]
  - Rango Fecha:
    - Inicio: 11/11/2007
    - Fin: 11/11/2007
  - Combustible: [Dropdown]
- Buttons: Filtrar, Ver todo

**Table:**

ID	Vehiculo	Modelo	Año	Color	Combustible	Existencias	Foto	Tipo	Marca

**Detalle Section:**

- Identificador: [Text Input]
- Vehiculo: [Dropdown]
- Tipo: [Text Input]
- Marca: [Text Input]
- Modelo: [Text Input]
- Año: [Date Picker]
- Color: [Text Input]
- Combustible: [Dropdown]
- Existencias: [Text Input]
- Foto: Cambiar imagen
- Buttons: Editar, Nuevo, Borrar, Cancelar, Guardar, Imprimir

### Formulario frmDetalleV

Desde este formulario podemos realizar consultas de los vehículos en el almacén mediante filtros así como imprimir reportes ya sea de manera específica (un solo vehículo) o de manera general (vehículos resultantes de la consulta).

```
//Archivo frmVehiculos.ui.h
#include <qgroupbox.h>
```





```
#include <qpixmap.h>
#include <qbuffer.h>
#include <qcstring.h>
#include <qwmatrix.h>
#include <qsqldatabase.h>
#include <sqlrecord.h>
#include <sqlquery.h>
#include <qstringlist.h>
#include <qfiledialog.h>
#include <qinputdialog.h>
#define ALLOC 0
#include "globales.h"
#include "conexion.h"
#include "frmVehiculos.h"
#include "MyFilePreview.h"
#include "MyPixmapLabel.h"
#include "frmReporte.h"

extern QSqlDatabase *db;
QString consul; //Consulta utilizada para el reporte

void frmDetalleV::MostrarFiltro( bool mostrar)
{
    if(mostrar)
        groupBoxFiltrar->show();
    else
        groupBoxFiltrar->hide();
    adjustSize();
}

void frmDetalleV::BloquearControles(bool activado)
{
    //Bloquear/desbloquear controles
    comboBoxIdV->setEnabled(activado);
    lineEditModelo->setReadOnly(!activado);
    dateEditAnyo->setEnabled(activado);
    lineEditColor->setReadOnly(!activado);
    comboBoxCombustible->setEnabled(activado);
    spinBoxExistencias->setEnabled(activado);
    myPixmapLabelFoto->setAcceptDrops(activado);
    //Estos botones deben activarse y desactivarse con los controles
    pushButtonCambiarImagen->setEnabled(activado);
    if(!activado) MostrarFiltro(false); //Ocultar el filtro
    pushButtonFiltrar->setEnabled(!activado);
}

void frmDetalleV::BloquearBotones( bool activado )
{
    tableDatos->setEnabled(activado); //La tabla se activa segun los botones
    //Bloquear/desbloquear los botones de modificacion de registros
    pushButtonNuevo->setEnabled(activado);
    pushButtonEditar->setEnabled(activado);
    pushButtonBorrar->setEnabled(activado);
    pushButtonGuardar->setEnabled(activado);
    pushButtonCancelar->setEnabled(activado);
}
```



```
        pushButtonImprimir->setEnabled(activado);
        pushButtonImprimir2->setEnabled(activado);
    }

bool frmDetalleV::BaseDatosVacia()
{
    //Verificar si existen registros en la base de datos
    if(query->size()>0)
    {
        BloquearControles(FALSE);//Bloquear los controles
        BloquearBotones(TRUE);//Desbloquear los botones
        //Desactivar los botones guardar y cancelar
        pushButtonGuardar->setEnabled(FALSE);
        pushButtonCancelar->setEnabled(FALSE);
        return FALSE;
    }
    else
    {
        BloquearControles(FALSE);
        BloquearBotones(FALSE);
        pushButtonNuevo->setEnabled(TRUE);//Activar solo boton nuevo
        QMessageBox::information(0, tr("Detalle de vehiculos"),tr("Base de datos
vacía"));
        LimpiarControles();
        return TRUE;
    }
}

void frmDetalleV::CargarDatos()
{
    consul=query->lastQuery();
    //Cargar los datos de la base de datos a la aplicacion
    int col;//Contador para las columnas

    QStringList ColsLabels;//Variable para almacenar las etiquetas de las
columnas
    tableDatos->setNumRows(query->size());//Establecer el numero de filas de
la tabla igual al numero de registros
    tableDatos->setNumCols(db->record(*query).count());//Establecer el numero
de columnas

    //Obtenemos los nombre de las columnas y establecemos en los titulos de
las mismas la propiedad stretchable a true
    for(col = 0;col < db->record(*query).count();col++)
    {
        ColsLabels.append(db->record(*query).field(col)->name());
        tableDatos->setColumnStretchable(col, true);
    }

    tableDatos->setColumnLabels(ColsLabels);//Establecer las etiquetas de las
columnas de la tabla

    int fila=0;//Contador para recorrer las filas
    while(query->next())
    {
```

```
//Rellenar las columnas de cada fila
for(col = 0; col < db->record(*query).count(); col++)
{
    //Verificar si esa columna posee una foto
    if(db->record(*query).field(col)->name().contains("foto",FALSE))
    {
        tableDatos->setPixmap(fila,col,QPixmap());
        //Redimensionar las fotos para que alcancen en una celda
        QPixmap foto(query->value(col).toByteArray());

        if(foto.isNull())
            tableDatos->setText(fila,col,"Sin foto");
        else
        {
            while(foto.width()>100 || foto.height()>100)//Si la imagen
es muy grande, la reducimos
            {
                QPixmap m; //Creamos una nueva matriz para escalar el
tamaño de la imagen
                m.scale( 0.5, 0.5 );//Crear la escala para la imagen de
0.5% horizontal y vertical
                foto=foto.xForm(m);//Guardamos la imagen escalada
            }
            tableDatos->setPixmap(fila,col,foto);
        }
    }
    //Verificar si tenemos el campo booleano Combustib
    else if(db->record(*query).field(col)->name()=="Combustible")
    {
        if(query->value(col).toBool())
            tableDatos->setText(fila,col,"Gasolina");
        else
            tableDatos->setText(fila,col,"Diesel");;
    }
    else
    {
        tableDatos->setText(fila,col,query->value(col).toString());
    }
    tableDatos->adjustRow(fila);
    fila++;//Siguiente fila
}
}

void frmDetalleV::TableDatosDoubleClicked(int fila)
{
    if(fila<0)return;

    query->seek(fila);

    //Establecer los datos de la fila seleccionada en el area de detalle del
formulario
    lineEditId->setText(tableDatos->text(fila,0));
    comboBoxIdV->setCurrentText(tableDatos->text(fila,1));
    lineEditModelo->setText(tableDatos->text(fila,2));
}
```



```
//Fecha en formato (yyyy-mm-dd)
dateEditAnyo->setDate(QDate::fromString(tableDatos->
text(fila,3),Qt::ISODate));

lineEditColor->setText(tableDatos->text(fila,4));

//Seleccionar un elemento del comboBoxCombustible
comboBoxCombustible->setCurrentText(tableDatos->text(fila,5));

spinBoxExistencias->setValue(tableDatos->text(fila,6).toInt());
myPixmapLabelFoto->setPixmap(query->value(7).toByteArray());
lineEditTipo->setText(tableDatos->text(fila,8));
lineEditMarca->setText(tableDatos->text(fila,9));
}

void frmDetalleV::CargarCombos()
{
    //Comprobar si existen vehiculos en la base de datos
    query->exec("SELECT * FROM vehiculos");
    while(query->next())
        comboBoxIdV->insertItem(query->value(0).toString());

    //Insertar los elementos del combo combustible
    comboBoxCombustible->insertItem("Diesel");
    comboBoxCombustible->insertItem("Gasolina");
    comboBoxCombustible_2->insertItem("Diesel");
    comboBoxCombustible_2->insertItem("Gasolina");

    //Comprobar si existen tipos en la base de datos
    query->exec("SELECT * FROM tipos_v");
    while(query->next())
        comboBoxTipo->insertItem(query->value(1).toString());

    //Comprobar si existen tipos en la base de datos
    query->exec("SELECT * FROM marcas_v");
    while(query->next())
        comboBoxMarca->insertItem(query->value(1).toString());
}

void frmDetalleV::LimpiarControles()
{
    //Limpiar los datos de los controles
    lineEditId->setText("");
    lineEditModelo->setText("");
    lineEditTipo->setText("");
    lineEditMarca->setText("");
    dateEditAnyo->setDate(QDate(0,0,0));
    lineEditColor->setText("");
    spinBoxExistencias->setValue(0);
    myPixmapLabelFoto->setPixmap(QPixmap());
    //Limpiar combos
    comboBoxIdV->clear();
    comboBoxCombustible->clear();
    comboBoxCombustible_2->clear();
    comboBoxTipo->clear();
}
```



```
        comboBoxMarca->clear();
    }

void frmDetalleV::init()
{
    query=new QSqlQuery();

    //Inicializar los controles de fechas
    dateEditAnyoIni->setDate(QDate::currentDate());
    dateEditAnyoFin->setDate(QDate::currentDate());
    dateEditAnyo->setDate(QDate::currentDate());

    MostrarFiltro(false);
    CargarCombos();
    query->exec("SELECT D.id_detalle_v AS ID, D.id_v AS Vehiculo, D.modelo_v
AS Modelo, D.anyo_v AS Anyo, D.color_v AS Color, D.combustible_v AS
Combustible, D.existencias AS Existencias, D.foto_v AS Foto, T.tipo_v AS Tipo,
M.marca_v AS Marca FROM detalle_v AS D INNER JOIN vehiculos AS V ON
D.id_v=V.id_v INNER JOIN tipos_v AS T ON V.id_tipo_v=T.id_tipo_v INNER JOIN
marcas_v AS M ON V.id_marca_v=M.id_marca_v");
    CargarDatos();

    dateEditAnyo->setSeparator("-");//Establecer el separador para el
QDateEdit

    if(!BaseDatosVacía())
    {
        pushButtonFiltrar->setEnabled(true);
        tableDatos->setEnabled(true);
        TableDatosDoubleClick(0);
    }

    //Indicamos que no estamos editando un registro existente
    editando=FALSE;
    move(0,0);
}

void frmDetalleV::IdVChange()
{
    //Comprobar si existen tipos de vehiculos en la base de datos
    query->exec("SELECT id_v, tipo_v,marca_v FROM vehiculos AS V INNER JOIN
tipos_v AS T ON V.id_tipo_v=T.id_tipo_v INNER JOIN marcas_v AS M ON
V.id_marca_v=M.id_marca_v WHERE id_v="+comboBoxIdV->currentText());
    if(query->size()==0)
    {
        QMessageBox::information(this,"Verifique!!!","Primero debe crear al
menos un vehiculo");
        frmVehiculos *v=new frmVehiculos();
        v->exec();
        return;
    }
    else
    {
        query->next();
        lineEditTipo->setText(query->value(1).toString());
    }
}
```



```
        lineEditMarca->setText(query->value(2).toString());
    }
}

void frmDetalleV::Nuevo()
{
    LimpiarControles();
    editando=FALSE;//Indicando que no estamos editando un registro existente

    //Desbloquear los controles, bloquear los botones y dejar solo activados
acceptary cancelar
    BloquearControles(TRUE);
    BloquearBotones(FALSE);
    pushButtonGuardar->setEnabled(TRUE);
    pushButtonCancelar->setEnabled(TRUE);
    pushButtonFiltrar->setEnabled(FALSE);
    tableDatos->setEnabled(FALSE);
    CargarCombos();
    IdVChange();
    lineEditModelo->setFocus();
}

void frmDetalleV::Editar()
{
    editando=true;
    id_editando=lineEditId->text().toInt();
    pos_editando=query->at();//Guardar el identificador del registro actual
    BloquearControles(TRUE);
    BloquearBotones(FALSE);
    pushButtonGuardar->setEnabled(TRUE);
    pushButtonCancelar->setEnabled(TRUE);
    lineEditModelo->setFocus();
}

void frmDetalleV::CambiarImagen()
{
    //Tipos de imagenes soportadas
    QString types("Image files (*.jpeg *.pbm *.pgm *.png *.xbm *.ppm *.jpg
*.bmp);;");
    QFileDialog *fd=new QFileDialog();//Crear un nuevo dialogo de navegacion
de ficheros
    fd->setFilters( types );//Establecer los filtros deseados
    MyFilePreview *p = new MyFilePreview;//Crear la vista previa a utilizar
    fd->setContentsPreviewEnabled( TRUE );//Activar el uso de vista previa
    fd->setContentsPreview( p,p );//Indicamos que la vista previa la
mostraremos en p y que p es la vista previa ya que esta derivado de QLabel y
QFilePreview
    fd->setPreviewMode( QFileDialog::Contents );//Indicamos que deseamos usar
la vista previa del fichero actual seleccionado en el dialogo de navegacion de
ficheros

    //Mostrar el dialogo de navegacion de ficheros y verificar si se
seleccione alguna imagen
    if(fd->exec()==QDialog::Accepted)
        myPixmapLabelFoto->setPixmap(QPixmap(fd->selectedFile()));
}
```

```
}

void frmDetalleV::Borrar()
{
    //Preguntar antes de borrar el registro actual
    if(QMessageBox::question(this, tr("Borrar -- Detalle vehiculos"),
tr("Desea borrar el registro actual?"), QMessageBox::Yes|QMessageBox::Default,
QMessageBox::No|QMessageBox::Escape ) ==QMessageBox::No)    return;

    //Tratar de borrar el registro actual
    if(query->exec("DELETE FROM detalle_v WHERE id_detalle_v="+lineEditId-
>text()))
    {
        QMessageBox::information( this,tr("Borrado -- Detalle
vehiculos"),tr("Registro borrado correctamente!!!"));

        query->exec("SELECT D.id_detalle_v AS ID, D.id_v AS Vehiculo,
D.modelo_v AS Modelo, D.anyo_v AS Anyo, D.color_v AS Color, D.combustible_v AS
Combustible, D.existencias AS Existencias, D.foto_v AS Foto, T.tipo_v AS Tipo,
M.marca_v AS Marca FROM detalle_v AS D INNER JOIN vehiculos AS V ON
D.id_v=V.id_v INNER JOIN tipos_v AS T ON V.id_tipo_v=T.id_tipo_v INNER JOIN
marcas_v AS M ON V.id_marca_v=M.id_marca_v");//Seleccionar todos los registros

        LimpiarControles();
        CargarDatos();
        BaseDatosVacía();
    }
    else
    {
        QMessageBox::critical( this,tr("Borrado -- Detalle
vehiculos"),tr("Error al borrar el registro actual"));
    }
}
```

```
void frmDetalleV::Guardar()
{
    if(myPixmapLabelFoto->pixmap()->isNull())
    {
        if(QMessageBox::question(this, tr("Verifique -- Detalle vehiculos"),
tr("La imagen esta vacia, desea dejarla asi?"),
QMessageBox::Yes|QMessageBox::Default, QMessageBox::No|QMessageBox::Escape )
==QMessageBox::No)    return;
    }

    if(QMessageBox::question(this, tr("Guardar -- Detalle vehiculos"),
tr("Desea guardar los cambios?"), QMessageBox::Yes|QMessageBox::Default,
QMessageBox::No|QMessageBox::Escape ) ==QMessageBox::No)    return;

    query->exec("SELECT D.id_detalle_v AS ID, D.id_v AS Vehiculo, D.modelo_v
AS Modelo, D.anyo_v AS Anyo, D.color_v AS Color, D.combustible_v AS
Combustible, D.existencias AS Existencias, D.foto_v AS Foto, T.tipo_v AS Tipo,
M.marca_v AS Marca FROM detalle_v AS D INNER JOIN vehiculos AS V ON
D.id_v=V.id_v INNER JOIN tipos_v AS T ON V.id_tipo_v=T.id_tipo_v INNER JOIN
marcas_v AS M ON V.id_marca_v=M.id_marca_v WHERE D.id_v="+comboBoxIdV-
>currentText() +" AND D.modelo_v ='"+ lineEditModelo->text() +' AND
D.anyo_v='"+ dateEditAnyo->date().toString("yyyy-MM-dd")+' AND
D.color_v='"+lineEditColor->text()+' AND
D.combustible_v="+QVariant(comboBoxCombustible->currentItem()).toString());

    if(query->next())
    {
        //Verifico si estoy insertando un Vehiculo nuevo
        if(!editando)
        {
            QMessageBox::critical( this, "Detalle vehiculos -- guardar","Ese
vehiculo ya existe.");
            return;
        }

        //Verifico si estoy editando ese Vehiculo
        if(editando && query->value(0).toString()!=lineEditId->text())
        {
            QMessageBox::critical( this, "Detalle vehiculos -- guardar","Ese
Vehiculo ya existe");
            return;
        }
    }

    QByteArray ba_foto;//Maneja la foto
    if(!myPixmapLabelFoto->pixmap()->isNull())
    {
        QBuffer buf(ba_foto);
        buf.open(IO_WriteOnly);
        if (not myPixmapLabelFoto->pixmap()->save(&buf,"PNG" ) )
        {
            QMessageBox::information( this, tr("Usuarios"),tr("error al
guardar en el buffer"));
            return;
        }
    }
}
```



```
        buf.close();
    }

    if(editando)//Verificamos si estamos editando un registro existente
    {
        query->prepare("UPDATE detalle_v SET
id_v=?,modelo_v=?,anyo_v=?,color_v=?,combustible_v=?,existencias=?,foto_v=?
WHERE id_detalle_v="+lineEditId->text());
    }
    else
    {
        query->prepare("INSERT INTO detalle_v
(id_v,modelo_v,anyo_v,color_v,combustible_v,existencias,foto_v)
VALUES(?,?,?,?,?,?,?)");
    }
    query->bindValue(0,comboBoxIdV->currentText().toInt());
    query->bindValue(1,lineEditModelo->text());
    query->bindValue(2,dateEditAnyo->date().toString("yyyy-MM-dd"));
    query->bindValue(3,lineEditColor->text());
    query->bindValue(4,comboBoxCombustible->currentItem());
    query->bindValue(5,spinBoxExistencias->value());
    query->bindValue(6,ba_foto);

    if(query->exec())
        QMessageBox::information(this,"Detalle Vehiculos --
Hecho","Registro guardado correctamente!!!");
    else
        QMessageBox::critical(this,"Detalle Vehiculos -- Error","Error al
intentar guardar el registro!!! "+query->lastError().text());

    query->exec("SELECT D.id_detalle_v AS ID, D.id_v AS Vehiculo, D.modelo_v
AS Modelo, D.anyo_v AS Anyo, D.color_v AS Color, D.combustible_v AS
Combustible, D.existencias AS Existencias, D.foto_v AS Foto, T.tipo_v AS Tipo,
M.marca_v AS Marca FROM detalle_v AS D INNER JOIN vehiculos AS V ON
D.id_v=V.id_v INNER JOIN tipos_v AS T ON V.id_tipo_v=T.id_tipo_v INNER JOIN
marcas_v AS M ON V.id_marca_v=M.id_marca_v");//Seleccionar todos los registros

    /*Si la base de datos no esta vacia nos posicionamos en el ultimo
registro*/
    CargarDatos();
    if(!BaseDatosVacía())
    {
        TableDatosDoubleClicked(query->size()-1);
    }
    else
        LimpiarControles();
}

void frmDetalleV::Cancelar()
{
    query->exec("SELECT D.id_detalle_v AS ID, D.id_v AS Vehiculo, D.modelo_v
AS Modelo, D.anyo_v AS Anyo, D.color_v AS Color, D.combustible_v AS
Combustible, D.existencias AS Existencias, D.foto_v AS Foto, T.tipo_v AS Tipo,
M.marca_v AS Marca FROM detalle_v AS D INNER JOIN vehiculos AS V ON
```



```
D.id_v=V.id_v INNER JOIN tipos_v AS T ON V.id_tipo_v=T.id_tipo_v INNER JOIN
marcas_v AS M ON V.id_marca_v=M.id_marca_v");
    if(BaseDatosVacia())
        LimpiarControles();
    else
    {
        if(editando)
            query->seek(pos_editando);//Recargamos el registro sin cambios
        else
            query->seek(query->size()-1);//Movernos al ultimo registro
        TableDatosDoubleClicked(pos_editando);//Recargar el registro
    }
}

void frmDetalleV::ImprimirTabla()
{
    //Crear un objeto de tipo frmReporte para la presentacion del reporte
    personalizado
    frmReporte *report_v=new frmReporte(this);

    //Cambiar el titulo
    QString titulo=QInputDialog::getText("Personalizar titulo", "Escriba el
    titulo del reporte:", QLineEdit::Normal,QString::null, 0, this );

    report_v->generar(titulo,consul,50,50);
    report_v->show();
}

void frmDetalleV::ImprimirDetalle()
{
    //Crear un objeto de tipo frmReporte para la presentacion del reporte
    personalizado
    frmReporte *report_v=new frmReporte(this);

    QString selec_detalle="SELECT D.id_detalle_v AS ID, D.id_v AS Vehiculo,
    D.modelo_v AS Modelo, D.anyo_v AS Anyo, D.color_v AS Color, D.combustible_v AS
    Combustible, D.existencias AS Existencias, foto_v AS Foto, T.tipo_v AS Tipo,
    M.marca_v AS Marca FROM detalle_v AS D INNER JOIN vehiculos AS V ON
    D.id_v=V.id_v INNER JOIN tipos_v AS T ON V.id_tipo_v=T.id_tipo_v INNER JOIN
    marcas_v AS M ON V.id_marca_v=M.id_marca_v WHERE D.id_v="+comboBoxIdV-
    >currentText() +" AND D.modelo_v ='"+ lineEditModelo->text() +' AND
    D.anyo_v='"+ dateEditAnyo->date().toString("yyyy-MM-dd")+"' AND
    D.color_v='"+lineEditColor->text()+"' AND
    D.combustible_v="+QVariant(comboBoxCombustible->currentItem()).toString();

    //Cambiar el titulo
    QString titulo=QInputDialog::getText("Personalizar titulo", "Escriba el
    titulo del reporte:", QLineEdit::Normal,QString::null, 0, this );

    report_v->generar(titulo,selec_detalle,200,200);
    report_v->show();
}

void frmDetalleV::VerTodo()
{

```



```
//Seleccionar todos los registros de la tableDatos
query->exec("SELECT D.id_detalle_v AS ID, D.id_v AS Vehiculo, D.modelo_v AS
Modelo, D.anyo_v AS Anyo, D.color_v AS Color, D.combustible_v AS Combustible,
D.existencias AS Existencias, D.foto_v AS Foto, T.tipo_v AS Tipo, M.marca_v AS
Marca FROM detalle_v AS D INNER JOIN vehiculos AS V ON D.id_v=V.id_v INNER
JOIN tipos_v AS T ON V.id_tipo_v=T.id_tipo_v INNER JOIN marcas_v AS M ON
V.id_marca_v=M.id_marca_v");
    CargarDatos();
}

void frmDetalleV::FiltrarDatos()
{
    if(dateEditAnyoIni->date().toString("yyyy-MM-dd")==")
        dateEditAnyoIni->setDate(QDate::fromString("1900-01-01",Qt::ISODate));

    if(dateEditAnyoFin->date().toString("yyyy-MM-dd")==")
        dateEditAnyoFin->setDate(QDate::fromString("1900-01-01",Qt::ISODate));

    query->exec("SELECT D.id_detalle_v AS ID, D.id_v AS Vehiculo, D.modelo_v
AS Modelo, D.anyo_v AS Anyo, D.color_v AS Color, D.combustible_v AS
Combustible, D.existencias AS Existencias, D.foto_v AS Foto, T.tipo_v AS Tipo,
M.marca_v AS Marca FROM detalle_v AS D INNER JOIN vehiculos AS V ON
D.id_v=V.id_v INNER JOIN tipos_v AS T ON V.id_tipo_v=T.id_tipo_v INNER JOIN
marcas_v AS M ON V.id_marca_v=M.id_marca_v WHERE T.tipo_v LIKE
'%" + comboBoxTipo->currentText() + "%' AND M.marca_v LIKE '%" + comboBoxMarca-
>currentText() + "%' AND D.modelo_v LIKE '%" + lineEditModelo_2->text() + "%' AND
D.anyo_v BETWEEN '" + dateEditAnyoIni->date().toString("yyyy-MM-dd") + "' AND
'" + dateEditAnyoFin->date().toString("yyyy-MM-dd") + "' AND D.color_v LIKE
'%" + lineEditColor_2->text() + "%' AND
D.combustible_v=" + QVariant(comboBoxCombustible_2-
>currentItem()).toString()); //Seleccionar todos los registros
    CargarDatos();
}
//Fin archivo frmVehiculos.ui.h

//Archivo Almacen.pro
TEMPLATE      = app
LANGUAGE      = C++

CONFIG        += qt release

INCLUDEPATH += . ./MyPixmapLabel ./MyFilePreview ./conexion ./login
./marcas ./tipos ./vehiculos ./detalle_v ./usuarios ./globales

HEADERS       += ./globales/globales.h \
    ./login/MaskLogin.h \
    ./conexion/conexion.h \
    ./MyPixmapLabel/MyPixmapLabel.h \
    ./MyFilePreview/MyFilePreview.h

SOURCES       += main.cpp \
    ./login/MaskLogin.cpp \
    ./conexion/conexion.cpp \
    ./MyPixmapLabel/MyPixmapLabel.cpp \
    ./MyFilePreview/MyFilePreview.cpp
```

```
FORMS = frmAlmacen.ui \  
      ../login/frmLogin.ui \  
      ../detalle_v/frmDetalleV.ui \  
      ../marcas/frmMarcas.ui \  
      ../tipos/frmTipos.ui \  
      ../usuarios/frmUsuarios.ui \  
      ../vehiculos/frmVehiculos.ui \  
      ../reporte/frmReporte.ui  
  
IMAGES      = ../Imágenes/A.png \  
      ../Imágenes/DiagramaER.png \  
      ../Imágenes/car.png \  
      ../Imágenes/agregar2.png \  
      ../Imágenes/borrar2.png \  
      ../Imágenes/editar.png \  
      ../Imágenes/ok.png \  
      ../Imágenes/cancel.png \  
      ../Imágenes/inicio.png \  
      ../Imágenes/final.png \  
      ../Imágenes/siguiente.png \  
      ../Imágenes/anterior.png \  
      ../Imágenes/security.png \  
      ../Imágenes/pass.png \  
      ../Imágenes/cancel2.png \  
      ../Imágenes/login.png \  
      ../Imágenes/server.png \  
      ../Imágenes/tux.png \  
      ../Imágenes/conectando.png \  
      ../Imágenes/m.png \  
      ../Imágenes/t.png \  
      ../Imágenes/abrir.png \  
      ../Imágenes/agregar.png \  
      ../Imágenes/borrar.png \  
      ../Imágenes/buscar.png \  
      ../Imágenes/usuarios.png \  
      ../Imágenes/qtlogo.png \  
      ../Imágenes/help.png \  
      ../Imágenes/logo_audi.png \  
      ../Imágenes/logo_chevrolet.png \  
      ../Imágenes/logo_honda.png \  
      ../Imágenes/logo_kia.png \  
      ../Imágenes/logo_mercedes.png \  
      ../Imágenes/logo_mitsubishi.png \  
      ../Imágenes/logo_toyota.png \  
      ../Imágenes/logo_yamaha.png \  
      ../Imágenes/penguin.png \  
      ../Imágenes/print.png \  
      ../Imágenes/reading.png  
  
TARGET = Almacen  
DESTDIR = ../bin  
MOC_DIR = ../moc  
OBJECTS_DIR = ../obj  
UI_HEADERS_DIR = ../ui_h
```

---



UI\_SOURCES\_DIR = ../ui\_cpp

```
#####
# Automatically generated by qmake (1.07a) Tue May 29 19:23:53 2007
#####

# Input
//Fin archivo Almacen.pro

//Archivo main.cpp
#include <qapplication.h>
#include "frmAlmacen.h"
#define ALLOC 1
#include "globales.h"

int main( int argc, char ** argv )
{
    QApplication a( argc, argv );

    frmAlmacen w;

    a.setMainWidget(&w);
    w.show();

    a.connect( &a, SIGNAL( lastWindowClosed() ), &a, SLOT( quit() ) );
    return a.exec();
}
//Fin archivo main.cpp
```

### Lista de archivos generados automaticamente por qt

Archivos generados por el MOC (Meta Object Compiler):

- moc\_frmAlmacen.cpp
- moc\_frmDetalleV.cpp
- moc\_frmLogin.cpp
- moc\_frmMarcas.cpp
- moc\_frmReporte.cpp
- moc\_frmTipos.cpp
- moc\_frmUsuarios.cpp
- moc\_frmVehiculos.cpp
- moc\_MyFilePreview.cpp
- moc\_MyPixmapLabel.cpp

Archivos generados por el UIC (User Interface Compiler):

- frmAlmacen.cpp
- frmDetalleV.cpp
- frmLogin.cpp
- frmMarcas.cpp
- frmReporte.cpp
- frmTipos.cpp



```
frmUsuarios.cpp  
frmVehiculos.cpp  
qmake_image_collection.cpp  
frmAlmacen.h  
frmDetalleV.h  
frmLogin.h  
frmMarcas.h  
frmReporte.h  
frmTipos.h  
frmUsuarios.h  
frmVehiculos.h
```

Archivos objetos:

```
conexion.o  
frmAlmacen.o  
frmDetalleV.o  
frmLogin.o  
frmMarcas.o  
frmReporte.o  
frmTipos.o  
frmUsuarios.o  
frmVehiculos.o  
main.o  
MaskLogin.o  
moc_frmAlmacen.o  
moc_frmDetalleV.o  
moc_frmLogin.o  
moc_frmMarcas.o  
moc_frmReporte.o  
moc_frmTipos.o  
moc_frmUsuarios.o  
moc_frmVehiculos.o  
moc_MyFilePreview.o  
moc_MyPixmapLabel.o  
MyFilePreview.o  
MyPixmapLabel.o  
qmake_image_collection.o
```