UNIVERSIDAD NACIONAL AUTONOMA DE NICARAGUA
UNAN-LEON
FACULTAD DE CIENCIAS
DEPARTAMENTO DE COMPUTACION

DISEÑO DE GUIAS DE LABORATORIO PARA BASES DE DATOS I BAJO PLATAFORMA LINUX

MONOGRAFIA PARA OPTAR AL TITULO DE
INGENIERO EN SISTEMAS DE INFORMACION

Presentado por:

Br. Marvin Guillermo Chacón Gutiérrez.
Br. Abelardo José Alvarado Martínez
Br. Juan Carlos Antón Soto.

Tutor:

Msc. Ernesto Espinoza.

León, Nicaragua, Agosto del 2007

DEDICATORIA

Dedico este trabajo de tesis:

A mi Madre:

Luisa Margarita Gutierrez Ríos, por darme la vida y por apoyarme incondicionalmente en cada etapa de mi formación, por darme aliento en las situaciones más difíciles a pesar de las necesidades y adversidades por las que tuve que pasar para poder lograr mis metas y sobre todo por depositar su confianza en mi, al delegar responsabilidades sin dudar que seria capaz de vencer cualquier obstáculo para alcanzar mis metas.

A mis Hermanos:

Anayanci, Ernesto y Harlingston Chacón Gutiérrez, por su paciencia y preocupación hacia mí ante cualquier problema de salud, por su cariño y afecto como hermanos y sobre todo por ser una de mis fuentes de lucha y esperanza en esta vida.

A mis Compañeros de Trabajo:

Abelardo y Juan Carlos, a quienes agradezco su paciencia y su entrega en este trabajo, a sus familias por brindarme su hospitalidad, en especial a la Señora Maria Elena Martínez por su amabilidad y cariño a lo largo de toda la carrera.

Y muy especialmente a mi Padre:

Que a pesar de no estar ya conmigo, agradezco el apoyo, el cariño y dedicación que me dio durante tuvo vida, porque se que desde donde se encuentre esta celebrando conmigo este triunfo.

Marvin Guillermo Chacón Gutiérrez.

DEDICATORIA

Dedico este trabajo de tesis:

A DIOS:

Por estar conmigo en todo momento de mi vida, dándome salud y las fuerzas necesarias para la terminación de este trabajo.

A mis padres:

MI MADRE:

Maria Elena Martínez Noguera la que me dio la vida, la que esta conmigo en las buenas y en las malas, la que esta ahí cuando la necesito, la que me ha visto crecer desde hace 23 años, la mujer de la que estoy orgulloso de ser su hijo.

MI PADRE:

Abelardo Sebastián Alvarado Martínez por apoyarme en todos los momentos de mi vida

A mis hermanos:

Marlene del Socorro Alvarado Martínez y Juan Pablo Alvarado Martínez por ayudarme de una u otra forma en las etapas de mi vida

A mi Esposa:

Darling Maria Sevilla Jiménez por estar conmigo en los momentos mas difíciles de mi vida, por apoyarme y comprenderme en algunas situaciones.

Abelardo José Alvarado Martínez

DEDICATORIA

Dedico esta tesis con todo amor a:

Dios, el ser quien siempre está a mi lado y me ha dado su misericordia y privilegio e ilumina mis pasos, a quien agradezco inmensamente por ayudarme a lograr esta etapa.

Mis adorados padres, quienes han preparado su camino con entrega y amor, los que dedicaron todo su tiempo, esfuerzo y sacrificio para poder coronar mi carrera, a ellos los seres más importantes y a quienes les debo todo lo que soy:

Manuel Antón Urbina.
Miriam Soto Delgado.

A Ana María Salgado, una de tantos profesores que me aconsejó en seguir adelante con la carrera, a quién le estoy muy agradecido y al resto de profesores, quienes me brindaron sus conocimientos durante toda la carrera, muchas gracias.

A todas las personas que de una u otra forma me han ayudado a terminar con éxito.

Juan Carlos Antón Soto.

AGRADECIMIENTO

Agradecemos:

A Dios:

Por darnos la vida, fuerza, esperanza y la sabiduría en cada momento de nuestra vida.

A nuestras Familias:

Por su entrega y abnegación en la lucha por sacarnos adelante y lograr nuestra principal meta: Graduarnos como Ingenieros en Sistemas de Información.

Al Msc Ernesto Espinoza:

Por su apoyo como guía en este trabajo, por su paciencia, su tiempo, y sobre todo por transmitirnos conocimientos mas avanzados en el área de las Bases de Datos.

A todo el claustro de profesores del Departamento de Computación:

Por su empeño y dedicación en transmitirnos todos y cada uno de los conocimientos que hoy son nuestros y que pondremos en practica de aquí en adelante.

A todas aquellas personas que de una u otra manera hicieron posible el desarrollo de este trabajo.

INDICE

I. INTRODUCCION

Son muchas las necesidades de automatización de sistemas, en las cuales nosotros como estudiantes de la carrera de Ingeniería en Sistemas y desarrolladores de ideas podemos impulsar.

Con base en lo anterior, nos hemos planteado diseñar las guías de laboratorio de Base de Datos I, con el objetivo de ayudar con el desarrollo de la asignatura y proponer ejercicios prácticos en base a nuevas aplicaciones que poseen las Bases de Datos y que aun no se han puesto en practica en el programa de la asignatura, los cuales irán acorde a los conocimientos impartidos en las clases teóricas.

Además se pretende migrar la plataforma de trabajo (Windows) que se ha estado utilizando hasta el momento, a una plataforma de software libre (Linux), para garantizar la facilidad y accesibilidad del software en un futuro, previendo las posibles políticas de Licenciamiento de utilización de software.

Diseñaremos cada una de las guias cumpliendo con los requerimientos necesarios para satisfacer las necesidades del programa de la asignatura, y que sirva de apoyo al docente encargado de impartir el curso de Bases de Datos I.

II. ANTECEDENTES

Las Bases de Datos como sistema de almacenamiento de información han tenido un desarrollo muy importante en las aplicaciones en las que se requiere un respaldo y seguridad de nuestros datos, por tal motivo se han venido tratando de ajustar las prácticas de laboratorio al avance en cuanto a software y diversas tecnologías que actualmente están surgiendo en el mercado.

En un primer momento, cuando surge la carrera de Computación se utilizaba DataIsi como sistema gestor de bases de datos y visual fox-pro como lenguaje de programación, esto no duro mucho tiempo debido a problemas de licenciamiento con DataIsi. Al poco tiempo surge Visual Basic como una mejor alternativa para el desarrollo de las aplicaciones con acceso a Bases de Datos.

A partir de ese momento las prácticas de laboratorio se han venido desarrollando en plataforma Windows, utilizando como sistema gestor de Bases de Datos en su momento SQL-Server y posteriormente MySQL, Access para la generación de tablas y Visual Basic como lenguaje de programación.

Actualmente, se trata de utilizar las nuevas herramientas que han surgido en software libre y proponer una mejor facilidad y accesibilidad a las mismas para el desarrollo de las prácticas de laboratorio propuestas en este trabajo como apoyo al curso de Bases de Datos I.

III. JUSTIFICACION

El desarrollo de este trabajo es para apoyar el Curso de Bases de Datos I impartido a los estudiantes de las carreras del departamento de Computación, mediante propuestas practicas que sirvan como ejercicio de consolidación de conocimientos y un mejor desarrollo de aprendizaje de los alumnos, logrando con esto ofrecer una nueva herramienta de trabajo al docente y brindarle al alumno el material y las herramientas necesarias para cumplir con los objetivos del curso.

Además de eso, la gran demanda y necesidad que existe actualmente de utilizar software libre como una herramienta que ofrece mayores posibilidades de accesibilidad y facilidad de utilización de Licenciamiento en un futuro.

IV. OBJETIVOS

GENERAL

- Crear ejercicios adecuados para cada uno de los temas específicos del programa del Curso de Bases de Datos I impartido a los estudiantes de las carreras del Departamento de Computación.

ESPECIFICOS

- Facilitar a los estudiantes ejercicios resueltos y propuestos que ayuden a poner en práctica los conocimientos adquiridos en la asignatura de base de datos I.

- Facilitar al docente encargado de impartir la asignatura una nueva herramienta de trabajo.

- Desarrollar las aplicaciones en una nueva plataforma (Linux).

V. MARCO TEORICO

1. Aspectos Generales.

1.1 Historia de las Bases de Datos.

Tuvieron sus orígenes en 1960 - 1962, cuando se empezaron a usar las máquinas que codificaban la información en tarjetas perforadas por medio de agujeros. Las bases de datos se crean con el objetivo de almacenar grandes cantidades de datos que antes se almacenaba en libros, lo que era lento, costoso y complejo (cualquier actualización a realizar, había que hacerla en cada uno de los libros en los que apareciera dicha información a modificar).

Antiguamente, los predecesores de los sistemas de bases de datos fueron los sistemas de ficheros. No hay un momento concreto en que los sistemas de ficheros hayan cesado y hayan dado comienzo los sistemas de bases de datos. De hecho, todavía existen sistemas de ficheros en uso. Cuando los ordenadores evolucionan, aparecen las cintas y los discos, a la vez que las máquinas son dotadas de mucha más potencia y facilidad de manipulación, es por tanto en ese momento cuando las bases de datos comienzan a ser realmente útiles.

Se dice que los sistemas de bases de datos tienen sus raíces en el proyecto estadounidense Apolo de mandar al hombre a la luna, en los años sesenta. En aquella época, no había ningún sistema que permitiera gestionar la inmensa cantidad de información que requería el proyecto.

La primera empresa encargada del proyecto, NAA (North American Aviation), desarrolló un *software* denominado GUAM (General Update Access Method) que estaba basado en el concepto de que varias piezas pequeñas se unen para formar una pieza más grande, y así sucesivamente hasta que el producto final este ensamblado. Esta estructura, que tiene la forma de un árbol, es lo que se denomina una estructura jerárquica.

A mediados de los sesenta, IBM se unió a NAA para desarrollar GUAM en lo que ahora se conoce como IMS (Information Management System). El motivo por el cual IBM restringió IMS al manejo de jerarquías de registros fue el de permitir el uso de dispositivos de almacenamiento serie, más exactamente las cintas magnéticas, ya que era un requisito del mercado por aquella época.

A mitad de los sesenta, se desarrolló IDS (Integrated Data Store), de General Electric. Este trabajo fue dirigido por uno de los pioneros en los sistemas de bases de datos, Charles Bachmann. IDS era un nuevo tipo de sistema de bases de datos conocido como *sistema de red*, que produjo un gran efecto sobre los sistemas de información de aquella generación.

Los sistemas jerárquico y de red constituyen la primera generación de los SGBD. Pero estos sistemas presentan algunos inconvenientes:

- Es necesario escribir complejos programas de aplicación para responder a cualquier tipo de consulta de datos, por simple que ésta sea.
- La independencia de datos es mínima.
- No tienen un fundamento teórico.

Hoy en día, existen cientos de SGBD relacionales, tanto para microordenadores como para sistemas multiusuario, aunque muchos no son completamente fieles al modelo relacional.

Los SGBD relacionales constituyen la segunda generación de los SGBD. Sin embargo, el modelo relacional también tiene sus fallos, siendo uno de ellos su limitada capacidad al modelar los datos.

La gestión de las bases de datos ha evolucionado desde una aplicación informática especializada hasta una parte esencial de un entorno informático moderno. Como tal, el conocimiento acerca de sistemas de bases de datos se ha convertido en una parte esencial en la formación en informática.

Hoy en día las empresas manejan una gran cantidad de datos. Cualquier empresa que se precie debe tener almacenados todos estos datos en una base de datos para poder realizarlos mediante una aplicación profesional; sin esta funcionalidad resultaría imposible tratar y manejar en su totalidad los datos que lleva a cabo la empresa y se perdería un tiempo y un dinero muy valiosos.

Los sistemas de bases de datos están disponibles en máquinas que van desde las computadoras personales más pequeñas hasta las mainframes más grandes.

1.2 Conceptos Elementales de las Bases de Datos.

Base de Datos: Cualquier conjunto de datos organizados para su almacenamiento en la memoria de un ordenador o computadora, diseñado para facilitar su mantenimiento y acceso de una forma estándar. Los datos suelen aparecer en forma de texto, números o gráficos.

Sistema de Gestión de Bases de Datos (SGBD): consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos, cuyo objetivo es proporcionar un entorno que sea tanto práctico como eficiente de usar en la recuperación y el almacenamiento de la información de la base de datos.

Base de Datos Relacional: Tipo de base de datos o sistema de administración de bases de datos, que almacena información en tablas (filas y columnas de datos) y realiza búsquedas utilizando los datos de columnas especificadas de una tabla para encontrar datos adicionales en otra tabla.

Modelo de Datos: es un conjunto de herramientas conceptuales para describir los datos, las relaciones entre ellos, su semántica y sus limitantes.

Modelo Relacional: Se usa una colección de tablas para representar tanto los datos como las relaciones entre esos datos.

Modelo Entidad Relación (E-R): Esta basado en una percepción del mundo real que consta de una colección de objetos básicos llamados entidades, y de relaciones entre estos objetos.

1.3 Lenguajes de bases de datos

Un sistema de base de datos proporciona dos tipos de lenguajes diferentes:

Uno para especificar el esquema de bases de datos (LDD) y el otro para expresar las consultas y actualizaciones de la base de datos (LMD).

1.3.1 Lenguaje de definición de datos (LDD)

El resultado de la compilación de las instrucciones del LDD es un conjunto de tablas que se almacenan en un archivo especial llamado diccionario de datos o directorio de datos.

1.3.2 Lenguaje de manipulación de datos (LMD)

Por manipulación de datos se entiende:

- La recuperación de la información almacenada en la base de datos.
- La inserción de información nueva en la base de datos.

- El borrado de información de la base de datos.
- La modificación de información almacenada en la base de datos.

1.4 Estructura de un sistema de Base de Datos

Un sistema de Base de Datos se divide en módulos que se encargan de cada una de las responsabilidades del sistema completo. Algunas de estas funciones del sistema de Base de Datos las puede proporcionar el sistema operativo de la computadora. En la mayoría de los casos los sistemas operativos proporcionan solo los servicios más básicos y los sistemas de Bases de Datos deben construirse sobre esta base. Así, el diseño de un sistema de Base de Datos debe incluir consideraciones de la interfaz entre el sistema de base de datos y el sistema operativo.

Los componentes funcionales de un sistema de Base de Datos se pueden dividir a grandes rasgos en componentes de procesamiento de consultas y componentes de gestión de almacenamiento.

Los componentes de procesamiento de consultas incluyen:

- **Compilador del LMD:** Traduce las instrucciones del LMD en lenguaje de consultas a instrucciones a bajo nivel que entiende el motor de evaluación de consulta y además intenta transformar las peticiones del usuario en otras equivalentes pero más eficientes, encontrando así una buena estrategia para ejecutar la consulta.
- **Precompilador del LMD incorporado:** que convierte las instrucciones del LMD incorporadas en un programa de aplicación en llamadas a procedimientos normales en el lenguaje anfitrión. El precompilador debe interactuar con el compilador del LMD para generar el código apropiado.

- **Interprete del LDD:** interpreta las instrucciones del LDD y las registra en un conjunto de tablas que contiene *metadatos.*
- **Motor de evaluación de consultas:** Ejecuta las instrucciones a bajo nivel generadas por el compilador del LMD.

Los componentes de gestión de almacenamiento incluyen:

- **Gestor de autorización e integridad:** comprueban que se satisfagan las ligaduras de integridad y la autorización de los usuarios para acceder a los datos.

- **Gestor de transacciones:** asegura que la Base de Datos quede en un estado consistente (correcto) a pesar de los fallos del sistema y que las ejecuciones de transacciones concurrentes ocurran sin conflictos.

- **Gestor de archivos:** gestiona la reserva de espacio de almacenamiento de disco y las estructuras de datos usadas para representar la información almacenada en disco.

- **Gestor de memoria intermedia:** es responsable de traer los datos del disco de almacenamiento a memoria principal y decidir qué datos tratar en la memoria caché.

Además, se necesitan varias estructuras de datos como parte de la implementación física del sistema:

- **Archivos de datos:** almacenan la Base de Datos en sí.
- **Diccionario de datos:** almacena metadatos acerca de la estructura de la Base de Datos.

- **Índices:** proporcionan acceso rápido a elementos de datos que tienen valores particulares.

- **Datos estadísticos:** almacenan información estadística sobre los datos en la Base de Datos. El procesador de consulta usa esta información para seleccionar las formas eficientes para ejecutar una consulta.

2. Diseño de Base de Datos

2.1 Consideraciones de diseño

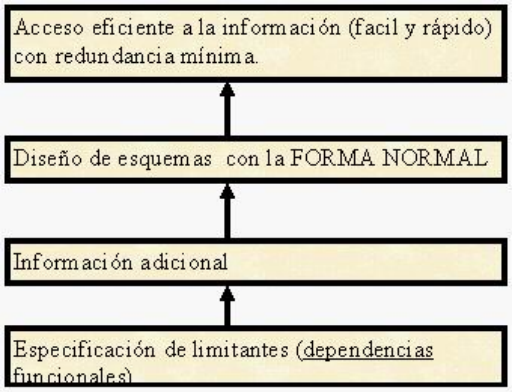
Son muchas las consideraciones a tomar en cuenta al momento de hacer el diseño de la base de datos, quizá las más fuertes sean:

- La velocidad de acceso,
- El tamaño de la información,
- El tipo de la información,
- Facilidad de acceso a la información,
- Facilidad para extraer la información requerida,
- El comportamiento del manejador de bases de datos con cada tipo de información.

No obstante que pueden desarrollarse sistemas de procesamiento de archivo e incluso manejadores de bases de datos basándose en la experiencia del equipo de desarrollo de software logrando resultados altamente aceptables, siempre es recomendable la utilización de determinados estándares de diseño que garantizan el nivel de eficiencia mas alto en lo que se refiere a almacenamiento y recuperación de la información.

2.2 Objetivo del diseño de Base de Datos

Entre las metas más importantes que se persiguen al diseñar un modelo de bases de datos, se encuentran las siguientes que pueden observarse en esta figura.



2.3 Modelo de los datos

Los diferentes modelos de datos se clasifican en tres grupos distintos:

- Modelos lógicos basados en objetos.
- Modelos lógicos basados en registros.
- Modelos físicos.

Modelos lógicos basados en objetos: Se caracterizan por el hecho de que proporcionan capacidades estructurales muy flexibles y permiten que las ligaduras de datos sean especificadas explícitamente. Algunos de los más ampliamente conocidos son: modelo entidad-relación, modelo orientada a objetos, modelo de datos semánticos y modelo de datos funcional.

Modelos lógicos basados en registros: Se usan para describir datos en los niveles lógico y de vistas. Al contrario de los modelos de datos basados en objetos, se usan tanto para especificar la estructura lógica completa de la Base de Datos, para proporcionar una descripción de alto nivel de la implementación. Los tres modelos basados en registro mas ampliamente aceptados son: el modelo relacional, modelo de red y el modelo jerárquico.

Modelos físicos: Se usan para describir datos en un nivel mas bajo. Al contrario con el modelo de datos lógico, hay pocos modelos de datos físicos en uso. Dos de los más conocidos son el modelo de unificación y el modelo de memoria por marcos.

3. Modelo Entidad – Relación.

3.1 Conceptos Básicos.

Entidad: 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.

En el modelado de bases de datos trabajaremos con conjuntos de entidades, y no con entidades individuales. La idea es generalizar de modo que el modelo se ajuste a las diferentes situaciones por las que pasará el proceso modelado a lo largo de su vida. Será el usuario final de la base de datos el que trabaje con entidades. Esas entidades constituirán los datos que manejará con la ayuda de la base de datos.

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

Según el conjunto de entidades al que hallamos asignado cada entidad, algunos de sus atributos podrán ser irrelevantes, y por lo tanto, no aparecerán; pero también pueden ser necesarios otros. Es decir, el conjunto de atributos que usaremos para una misma entidad dependerá del conjunto de entidades al que pertenezca, y por lo tanto del proceso modelado.

Dominio: conjunto de valores posibles para un atributo.

Con nombres o textos, los dominios limitarán su longitud máxima.

Sin embargo, los dominios no son demasiado importantes en el modelo E-R, nos preocuparemos mucho más de ellos en el modelo relacional y en el físico.

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.

Clave: es un conjunto de atributos que identifican de forma unívoca una entidad. 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.

En fin, que en ocasiones, por un motivo u otro, creamos un atributo artificial para usarlo sólo como clave. Esto es perfectamente legal en el modelo E-R, y se hace frecuentemente porque resulta cómodo y lógico.

Interrelación: es la asociación o conexión entre conjuntos de entidades.

Claves candidatas: es cada una de las claves mínimas existente en un conjunto de entidades. 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*.

Clave principal: (o primaria), 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. De este modo, el identificador de una interrelación es el conjunto de las claves primarias de cada una de las entidades interrelacionadas.

Generalización: es el proceso según el cual se crea un conjunto de entidades a partir de otros que comparten ciertos atributos.

A veces existen situaciones en que sea conveniente crear una entidad como una fusión de otras, en principio, diferentes, aunque con atributos comunes. Esto disminuye el número de conjuntos de entidades y facilita el establecimiento de interrelaciones.

La desventaja de la generalización es que se desperdicia espacio de almacenamiento, ya que sólo algunos de los atributos no comunes contienen información en cada entidad, el resto se desperdicia.

La ventaja es que podemos establecer el mismo tipo de interrelación con cualquier entidad del conjunto.

Especialización: es el proceso según el cual se crean varios tipos de entidades a partir de uno. Cada una de los conjuntos de entidades resultantes contendrá sólo algunos de los atributos del conjunto original.

La idea es lógica: si la generalización tiene ventajas e inconvenientes, cuando los inconvenientes superan a las ventajas, será conveniente hacer una especialización.

3.2. Cardinalidades de un conjunto Entidad.

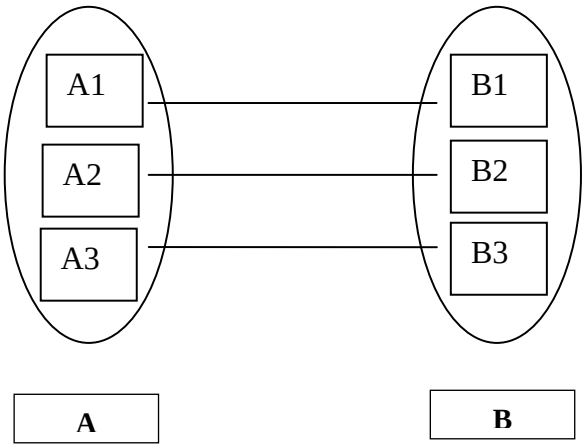
Grado: número de conjuntos de entidades que intervienen en una interrelación.

Cuando intervienen interrelaciones de dos entidades, diremos que es de grado 2 o binaria. También existen interrelaciones de grado 3, 4, etc. Pero las más frecuentes son las interrelaciones binarias.

Podemos establecer una interrelación ternaria (de grado tres) entre personas, de modo que dos personas sean padre y madre, respectivamente, de una tercera.

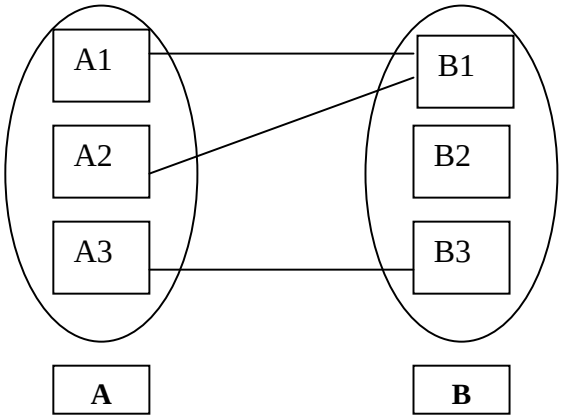
Existen además cuatro tipos distintos de interrelaciones binarias, dependiendo del número de entidades del primer conjunto de entidades y del segundo. Así hablaremos de interrelaciones 1:1 (uno a uno), 1:N (uno a muchos) y N:M (muchos a muchos).

1:1 (uno a uno): Una entidad en A se asocia con a lo sumo una entidad en B, y una entidad en B se asocia con a lo sumo una entidad en A.

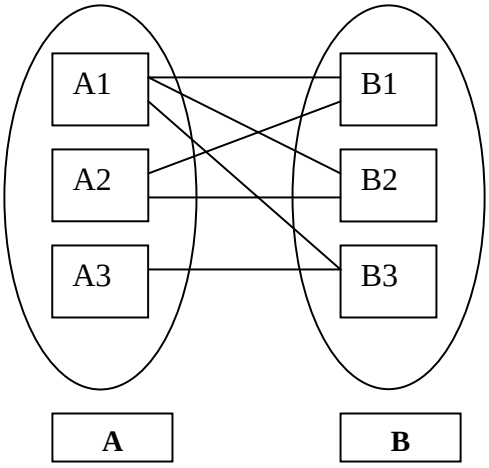


1: N (uno a varios): Una entidad en A se asocia con cualquier número de entidades en B. Una entidad en B, sin embargo, se puede asociar con a lo sumo una entidad en A.

N:1 (varios a uno): Una entidad en A se asocia con a lo sumo una entidad en B. Una entidad en B, sin embargo, se puede asociar con cualquier número de entidades en A.



N:M (varios a varios): Una entidad en A se asocia con cualquier número en B, y una entidad en B se asocia con cualquier número de entidades en A.



3.3. Conjunto de entidades débiles y fuertes.

Un conjunto de entidades puede no tener suficientes atributos para formar una clave primaria. Tal conjunto de entidades se denomina conjunto de entidades débiles. Un conjunto de entidades que tiene una clave primaria se denomina conjunto de entidades fuertes.

Para que un conjunto de entidades débiles tenga sentido, debe formar parte de un conjunto de relaciones uno a varios. Este conjunto de relaciones no debería tener atributos descriptivos, ya que cualquier atributo requerido puede estar asociado con el conjunto de entidades débiles.

Un miembro de un conjunto de entidades fuerte es por definición una entidad dominante, mientras que un miembro de un conjunto de entidades débiles es una entidad subordinada.

La clave primaria de un conjunto de entidades débiles se forma mediante la clave primaria del conjunto de entidades fuerte de cuya existencia depende el conjunto de entidades débiles, mas el discriminante del conjunto de entidades débiles. El discriminante de un conjunto de entidades débil es un conjunto de atributos que permite que esta distinción se haga.

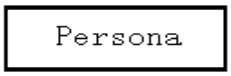
3.3.1 Dependencia de existencia.

Decimos que existe una dependencia de existencia entre una entidad, *subordinada*, y otra, *dominante*, cuando la eliminación de la entidad dominante, conlleva también la eliminación de la entidad o entidades subordinadas.

Desde cierto punto de vista, podemos considerar que las entidades dominantes y sus entidades subordinadas forman parte de una misma entidad. Es decir, una entidad está formada por ella misma y sus circunstancias (citando a Ortega :-). Esas circunstancias podrían ser, en el caso de nuestro vehículo, además de los viajes que ha hecho, los dueños que ha tenido, las revisiones que se le han efectuado, averías, etc. Es decir, todo su historial.

3.4. Diagrama Entidad – Relación.

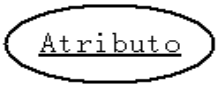
Entidad: Las entidades se representan con un rectángulo, y en su interior el nombre de la entidad:



Las entidades débiles pueden representarse mediante dos rectángulos inscritos. Ya sabemos que existe una dependencia de existencia entre la entidad débil y la fuerte, esto se representa también añadiendo una flecha a la línea que llega a la entidad débil.

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.



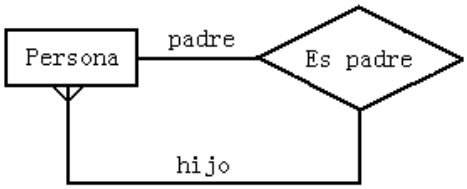
También es frecuente usar una doble elipse para indicar atributos multivaluados:

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.

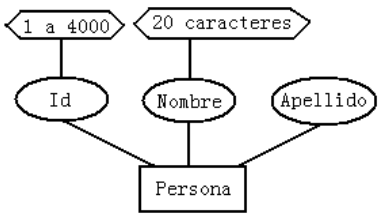
Interrelación: Las interrelaciones se representan mediante rombos, y en su interior el nombre de la interrelación:

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:



Diagrama

Un diagrama E-R consiste en representar mediante estas 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.

La idea es simple, aparentemente, pero a la hora de construir modelos sobre realidades concretas es cuando surgen los problemas. La realidad es siempre compleja. Las entidades tienen muchos atributos diferentes, de los cuales debemos aprender a elegir sólo los que necesitemos. Lo mismo cabe decir de las interrelaciones. Además, no siempre está perfectamente claro qué es un atributo y qué una entidad; o que ventajas obtenemos si tratamos a ciertos atributos como entidades y viceversa.

Al final, nuestra mejor arma es la práctica. Cuantos más problemas diferentes modelemos más aprenderemos sobre el proceso y sobre los problemas que pueden surgir. Podremos aplicar la experiencia obtenida en otros proyectos, y, si no reducir el tiempo empleado en el modelado, al menos sí reducir los retoques posteriores, el mantenimiento y el tiempo necesario para realizar modificaciones sobre el modelo.

Construir un modelo E-R

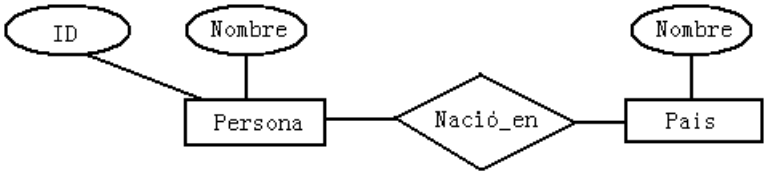
Podemos dividir el proceso de construir un modelo E-R en varias tareas más simples. El proceso completo es iterativo, es decir, una vez terminado debemos volver al comienzo, repasar el modelo obtenido y, probablemente, modificarlo. Una vez satisfechos con el resultado (tanto nosotros, los programadores, como el cliente), será el momento de pasar a la siguiente fase: el modelo lógico.

Uno de los primeros problemas con que nos encontraremos será decidir qué son entidades y qué atributos.

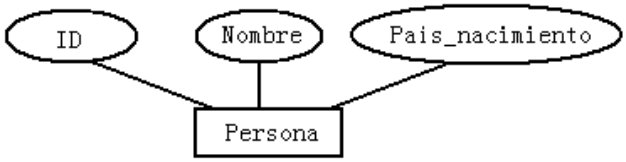
La regla principal es que una entidad sólo debe contener información sobre un único objeto real.

Pero en ciertos casos esto nos puede obligar a crear entidades con un único atributo. Por ejemplo, si creamos una entidad para representar una persona, uno de los atributos puede ser el lugar de nacimiento. El lugar de nacimiento: población, provincia o país, puede ser considerado como una entidad. Sin embargo en nuestro

caso concreto, tal vez, esta información sea sólo eso: un lugar de nacimiento. ¿Debemos pues almacenar esa información como un atributo de persona o debemos, por el contrario, crear una entidad independiente?.

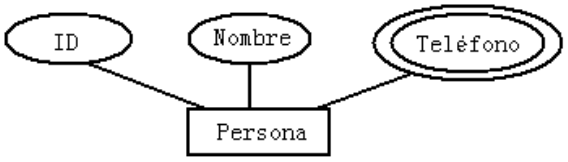


Una regla que puede ayudarnos en esta decisión es que si una entidad sólo tiene un atributo, que sirve para identificarlo, entonces esa entidad puede ser considerara como un atributo.

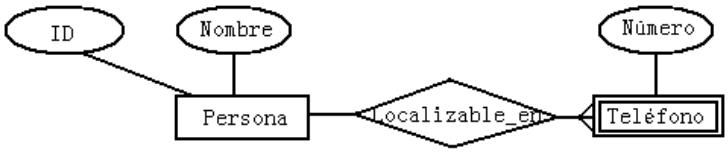


Otro problema frecuente se presenta con los atributos multivaluados.

Por ejemplo, cada persona puede ser localizada en varios números de teléfono. Considerando el teléfono de contacto como un atributo de persona, podemos afirmar que tal atributo es multivaluado.



Pero, aunque como su propio nombre indica no dejan de ser atributos, es mejor considerar a los atributos multivaluados como entidades débiles subordinadas. Esto nos evitará muchos problemas con el modelo lógico relacional.



3.5. Reducción de un esquema E-R a tablas.

Una base de datos que se constituye en un esquema de bases de datos E-R se puede representar por una colección de tablas. Para cada conjunto de entidades de la base de datos y para cada conjunto de relaciones hay una única tabla a la que se asigna el nombre del conjunto de entidades o del conjunto de relaciones correspondientes.

Ambos modelos, el modelo E-R y el modelo de base de datos relacionales, son representaciones abstractas y lógicas del desarrollo del mundo real. Debido a que los dos modelos emplean principios de diseño similares, se puede convertir un diseño E-R en un diseño relacional.

Convertir una representación de bases de datos de un diagrama E-R a un formato de tabla es la base para la derivación de un diseño de base de datos relacional desde un diagrama E-R.

En el modelo E-R anterior las entidades *persona* y *teléfono* y la relación *localizable_en* se convierten en tablas de la siguiente manera:

Tabla: Persona	
Id	***Nombre***
123456	Marvin Chacon
654321	Juan Antón

Tabla: Teléfono
Numero
315-0659
315-3464

Tabla: localizable_en
Id
123456
654321

4. Modelo Relacional.

4.1. Estructura de las Bases de Datos relacional.

Una base de datos relacional consiste en un conjunto de tablas, a cada una de las cuales se le asigna un nombre exclusivo. Cada fila de una tabla representa una relación entre un conjunto de valores. Dado que cada tabla es un conjunto de dichas relaciones, hay una fuerte correspondencia entre el concepto de tabla y el concepto matemático de relación, del que toma su nombre el modelo de datos relacional.

4.2. Lenguajes de Consulta.

Un lenguaje de consulta es un lenguaje en el que un usuario solicita información de la base de datos. Estos lenguajes suelen ser de un nivel superior que el de los lenguajes de programación habituales y pueden clasificarse como: *procedimentales o no procedimentales.*

En los lenguajes procedimentales, el usuario instruye al sistema para que lleve a cabo una serie de operaciones en la base de datos para calcular el resultado deseado. En los lenguajes no procedimentales, el usuario describe la información deseada sin dar un procedimiento concreto para obtener esa información.

La mayor parte de los sistemas comerciales de bases de datos relacionales ofrecen un lenguaje de consulta que incluye elementos de los enfoques procedimental y no procedimental.

4.3 Álgebra Relacional

El álgebra relacional es un lenguaje de consulta procedimental. Consta de un conjunto de operaciones que toman como entrada una o dos relaciones y producen como resultado una nueva relación. Las operaciones fundamentales del álgebra relacional son: *selección, proyección, unión, diferencia de conjuntos, producto cartesiano y renombramiento.*

Las operaciones selección, proyección y renombramiento se denominan operaciones unarias porque operan sobre una sola relación, las otras tres operaciones operan sobre pares de relaciones y se denominan operaciones binarias.

Operación selección: selecciona tuplas que satisfacen un predicado. Se utiliza la letra sigma (σ) minúscula para denotar la selección y el predicado aparece como subíndice de σ. La relación del argumento aparece entre paréntesis a continuación de σ.

Ejemplo: Seleccionar las tuplas de la relación alumnos en que la edad es 20.

$\sigma_{edad = 20}$ (alumnos).

Además se permiten las comparaciones que utilizan =, ≠, <, ≤, >, ≥, en el predicado de selección y se pueden combinar varios predicados en uno mayor usando las conectivas y (Λ) y 0 (V).

Ejemplo: Seleccionar las tuplas de la relación alumnos en que el nombre es Mario y la edad > 18.

σ nombre = ” Mario” Λ edad > 18 (alumnos).

Operación Proyección: Es una operación unaria que devuelve su relación de argumentos, excluyendo algunos argumentos. Dado que las relaciones son conjuntos se eliminan todas las filas duplicadas. Se denota por la silaba griega pi (π). Se crea una lista de los atributos que se desea que aparezcan en el resultado como subíndice de π. La relación de argumentos se escribe a continuación entre paréntesis.

Ejemplo: Crear una lista de todos los nombres de los alumnos con su numero de carnet.

$\Pi_{n\_carnet,\ nombre}$ (alumnos)

Operación Renombramiento: A diferencia de las relaciones de las bases de datos, los resultados del álgebra relacional no tienen un nombre que se pueda utilizar para referirse a ellas. El operador renombramiento denotado por la letra griega ro (ρ) minúscula permite hacer esta tarea.

Ejemplo: Dada una expresión E del álgebra relacional, la expresión

ρ_x (E) devuelve el resultado de E con el nombre de x

Las relaciones de la base de datos por si mismas se consideran expresiones (triviales) del álgebra relacional, por tanto también se pueden aplicar la operación de renombramiento a una relación para obtener la misma relación con un nombre nuevo.

Operación Unión: Es una operación binaria denotada como en la teoría de conjuntos por U. Para que una operación Unión de dos relaciones r y s sea valida debe cumplir dos condiciones:

1. Las relaciones r y s deben tener el mismo numero de atributos.

2. Los dominios de los atributos i-esimos de r y de s deben ser iguales para todo i.

Téngase en cuenta que r y s pueden ser, en general, relaciones temporales que sean resultado del álgebra relacional.

Operación Diferencia de Conjuntos: Permite buscar las tuplas que estén en una relación pero no en la otra, se denota por el signo (-). Al igual que en la operación unión, para que una operación diferencia de conjuntos sea valida debe cumplir las anteriores condiciones.

Operación Producto Cartesiano: Permite combinar información de cualesquiera dos relaciones, se denota por una x. Por ejemplo el producto cartesiano de las relaciones r1 y r2 se denota por r1 x r2.

Hay que recordar que las relaciones se definen como subconjuntos del producto cartesiano de un conjunto de dominios. A partir de esta definición ya se debe tener una intuición sobre la definición de la operación producto cartesiano. Sin embargo dado que el mismo nombre de atributo puede aparecer tanto en r1 como en r2, hay que crear un esquema de denominaciones para distinguir entre ambos atributos. En este caso se logra adjuntando al atributo el nombre de la relación de la que proviene originalmente.

Por ejemplo: El esquema del producto cartesiano de las relaciones alumnos y profesores quedaría de la siguiente manera:

(alumnos.n_carnet, alumnos.nombre, alumnos.edad, profesores.codigo_profesor, profesores.estado_civil)

5. Sistema Gestor de Base de Datos (MySQL).

5.1 Introducción.

¿Qué es MySQL?

- Administrador de Base de Datos.
- Sistema administrador de Base de Datos relacionales.
- Software de fuente abierta.

Características técnicas de MySQL.

MySQL es un sistema Cliente/Servidor que consta de un servidor SQL multi-hilo que soporta diferentes backends, variados programas clientes y de librerías, administrador de herramientas y un programa de interfaces.

Los valores centrales de MySQL son:

- La mejor y más usada base en el mundo.
- Disponible y accesible para todos.
- Fácil de usar.
- Se está perfeccionando continuamente mientras permanece rápida y segura.
- Divertida para usar y perfeccionar.
- Libre de molestias.

Características principales de MySQL.

Algunas de las características más importantes de MySQL son:

- Escrito en C y C++, testado con GCC 2.7.2.1. Usa GNU autoconf para portabilidad.

- Clientes C, C++, Eiffel, PHP, Python, JAVA, Perl, TCL.
- Multiproceso, es decir puede usar varias CPU si éstas están disponibles.
- Puede trabajar en distintas plataformas y S.O. distintos.
- Sistemas de contraseñas y privilegios muy flexibles y seguros.
- Todas las palabras de paso viajan encriptadas en la red.
- Registros de longitud fija y variable.
- Todas las columnas pueden tener valores por defecto.
- Utiliza *Isamchk* para chequear, optimizar y reparar tablas.
- Los clientes usan TCP o UNIX Socket para conectarse al servidor.
- Todos los comandos tienen –help ó –? para las ayudas.
- Soporta diversos tipos de columnas como enteros de 1, 2, 3, 4 y 8 bytes, coma flotante, doble precisión, carácter, fecha, enumerados, etc.

Uso de MySQL.

MySQL es muy rápido, seguro y fácil de usar. Ha desarrollado un conjunto de características muy prácticas, en estrecha cooperación con otros usuarios.

MySQL fue desarrollado para manejar grandes Bases de Datos mucho más rápido que las soluciones existentes y ha sido usado exitosamente en ambientes de producción con altas demandas por varios años.

Aunque está bajo un desarrollo constante, MySQL siempre ofrece conjunto de funciones muy poderoso y eficiente. La conectividad, velocidad y seguridad hace de MySQL una suite poderosa para acceder a Bases de Datos en Internet.

5.2. DDL de MySQL.

Creación de Bases 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.

Debido a esto, 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.

Vamos a crear y manipular nuestra propia base de datos, al tiempo que nos familiarizamos con la forma de trabajar de **MySQL**.

Para empezar, crearemos una base de datos para nosotros solos, y la llamaremos ***"prueba".*** Para crear una base de datos se usa una sentencia ***CREATE DATABASE:***

```
mysql> CREATE DATABASE prueba;
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           |
+--------------------+
| mysql              |
| prueba             |
| test               |
+--------------------+
3 rows in set (0.00 sec)
```

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 prueba;
Database changed
```

Creación de tablas.

La sentencia CREATE TABLE 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. Las iremos viendo paso a paso, y en poco tiempo sabremos usar muchas de sus posibilidades.

En su forma más simple, la sentencia CREATE TABLE creará una tabla con las columnas que indiquemos. Crearemos, como ejemplo, una tabla que nos permitirá almacenar nombres de personas y sus fechas de nacimiento. Deberemos indicar el nombre de la tabla y los nombres y tipos de las columnas:

```
mysql> USE prueba
Database changed
mysql> CREATE TABLE gente (nombre VARCHAR(40), fecha DATE);
Query OK, 0 rows affected (0.53 sec)
```

Hemos creado una tabla llamada "gente" con dos columnas: "nombre" que puede contener cadenas de hasta 40 caracteres y "fecha" de tipo fecha.

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

Pero tenemos muchas más opciones a la hora de definir columnas. Además del tipo y el nombre, podemos definir valores por defecto, permitir o no que contengan valores nulos, crear una clave primaria, indexar,etc.

La sintaxis para definir columnas es:

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

Valores nulos

Al definir cada columna podemos decidir si podrá o no contener valores nulos.

Debemos recordar que, aquellas columnas que son o forman parte de una clave primaria no pueden contener valores nulos.

Veremos que, si definimos una columna como clave primaria, automáticamente se impide que pueda contener valores nulos, pero este no es el único caso en que puede ser interesante impedir la asignación de valores nulos para una columna.

La opción por defecto es que se permitan valores nulos, *NULL*, y para que no se permitan, se usa *NOT NULL*. Por ejemplo:

```
mysql> CREATE TABLE ciudad1 (nombre CHAR(20) NOT NULL, poblacion INT NULL);
Query OK, 0 rows affected (0.98 sec)
```

Valores por defecto

Para cada columna también se puede definir, opcionalmente, un valor por defecto. El valor por defecto se asignará de forma automática a una columna cuando no se especifique un valor determinado al añadir filas.

Si una columna puede tener un valor nulo, y no se especifica un valor por defecto, se usará NULL como valor por defecto. En el ejemplo anterior, el valor por defecto para *poblacion* es NULL.

Por ejemplo, si queremos que el valor por defecto para *poblacion* sea 5000, podemos crear la tabla como:

```
mysql> CREATE TABLE ciudad2 (nombre CHAR(20) NOT NULL,
    -> poblacion INT NULL DEFAULT 5000);
Query OK, 0 rows affected (0.09 sec)
```

Claves primarias

También se puede definir una clave primaria sobre una columna, usando la palabra clave *KEY* o *PRIMARY KEY*.

Sólo puede existir una clave primaria en cada tabla, y la columna sobre la que se define una clave primaria no puede tener valores *NULL*. Si esto no se especifica de forma explícita, **MySQL** lo hará de forma automática.

Por ejemplo, si queremos crear un índice en la columna *nombre* de la tabla de ciudades, crearemos la tabla así:

```
mysql> CREATE TABLE ciudad3 (nombre CHAR(20) NOT NULL PRIMARY KEY,
    -> poblacion INT NULL DEFAULT 5000);
Query OK, 0 rows affected (0.20 sec)
```

Usar *NOT NULL PRIMARY KEY* equivale a *PRIMARY KEY*, *NOT NULL KEY* o sencillamente *KEY*.

Existe una sintaxis alternativa para crear claves primarias, que en general es preferible, ya que es más potente. De hecho, la que hemos explicado es un alias para la forma general, que no admite todas las funciones (como por ejemplo, crear claves primarias sobre varias columnas).

Columnas autoincrementadas

En **MySQL** tenemos la posibilidad de crear una columna autoincrementada, aunque esta columna sólo puede ser de tipo entero.

Si al insertar una fila se omite el valor de la columna autoincrementada 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.

Generalmente, estas columnas se usan como claves primarias 'artificiales'. **MySQL** está optimizado para usar valores enteros como claves primarias, de modo que la combinación de clave primaria, que sea entera y autoincrementada es ideal para usarla como clave primaria artificial:

```
mysql> CREATE TABLE ciudad5 (clave INT AUTO_INCREMENT PRIMARY KEY,
    -> nombre CHAR(20) NOT NULL,
    -> poblacion INT NULL DEFAULT 5000);
Query OK, 0 rows affected (0.11 sec)
mysql>
```

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:

```
mysql> CREATE TABLE ciudad6
    -> (clave INT AUTO_INCREMENT PRIMARY KEY COMMENT 'Clave principal',
    -> nombre CHAR(50) NOT NULL,
    -> poblacion INT NULL DEFAULT 5000);
Query OK, 0 rows affected (0.08 sec)
```

5.3. DML de Mysql.

Inserción de datos.

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.

```
mysql> INSERT INTO gente VALUES ('Fulano','1974-04-12');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO gente VALUES ('Mengano','1978-06-15');
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO gente VALUES
    -> ('Tulano','2000-12-02'),
    -> ('Pegano','1993-02-10');
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Si no necesitamos asignar un valor concreto para alguna columna, podemos asignarle el valor por defecto indicado para esa columna cuando se creó la tabla, usando la palabra *DEFAULT*:

```
mysql> INSERT INTO ciudad2 VALUES ('Perillo', DEFAULT);
Query OK, 1 row affected (0.03 sec)
mysql> SELECT * FROM ciudad2;
+---------+-----------+
| nombre  | población |
+---------+-----------+
| Perillo |      5000 |
+---------+-----------+
1 row in set (0.02 sec)
```

En este caso, como habíamos definido un valor por defecto para población de 5000, se asignará ese valor para la fila correspondiente a 'Perillo'.

Otra opción consiste en indicar una lista de columnas para las que se van a suministrar valores. A las columnas que no se nombren en esa lista se les asigna el valor por defecto. Este sistema, además, permite usar cualquier orden en las columnas, con la ventaja, con respecto a la anterior forma, de que no necesitamos conocer el orden de las columnas en la tabla para poder insertar datos:

```
mysql> INSERT INTO ciudad5 (poblacion,nombre) VALUES
    -> (7000000, 'Madrid'),
    -> (9000000, 'París'),
    -> (3500000, 'Berlín');
Query OK, 3 rows affected (0.05 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM ciudad5;
+-------+--------+-----------+
| clave | nombre | poblacion |
+-------+--------+-----------+
|     1 | Madrid |   7000000 |
|     2 | París  |   9000000 |
|     3 | Berlín |   3500000 |
+-------+--------+-----------+
3 rows in set (0.03 sec)
mysql>
```

Cuando creamos la tabla "ciudad5" definimos tres columnas: 'clave', 'nombre' y 'población' (por ese orden). Ahora hemos insertado tres filas, en las que hemos omitido la clave, y hemos alterado el orden de 'nombre' y 'población'. El valor de la 'clave' se calcula automáticamente, ya que lo hemos definido como auto-incrementado.

Existe otra sintaxis alternativa, que consiste en indicar el valor para cada columna:

```
mysql> insert into ciudad5
    -> SET nombre='Roma', poblacion=8000000;
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM ciudad5;
+-------+--------+-----------+
| clave | nombre | poblacion |
+-------+--------+-----------+
|     1 | Madrid |   7000000 |
|     2 | París  |   9000000 |
|     3 | Berlín |   3500000 |
|     4 | Roma   |   8000000 |
+-------+--------+-----------+
4 rows in set (0.03 sec)
```

Una vez más, a las columnas para las que no indiquemos valores se les asignarán sus valores por defecto. También podemos hacer esto usando el valor *DEFAULT*.

Para las sintaxis que lo permiten, podemos observar que cuando se inserta más de una fila en una única sentencia, obtenemos un mensaje desde **MySQL** que indica el número de filas afectadas, el número de filas duplicadas y el número de avisos.

Para que una fila se considere duplicada debe tener el mismo valor que una fila existente para una clave principal o para una clave única.

En tablas en las que no exista clave primaria ni índice de clave única no tiene sentido hablar de filas duplicadas. Es más, en esas tablas es perfectamente posible que existan filas con los mismos valores para todas las columnas.

Por ejemplo, en *mitabla5* tenemos una clave única sobre la columna 'nombre':

```
mysql> INSERT INTO mitabla5 (id, nombre) VALUES
    -> (1, 'Carlos'),
    -> (2, 'Felipe'),
    -> (3, 'Antonio'),
    -> (4, 'Carlos'),
    -> (5, 'Juan');
ERROR 1062 (23000): Duplicate entry 'Carlos' for key 1
```

Si intentamos insertar dos filas con el mismo valor de la clave única se produce un error y la sentencia no se ejecuta. Pero existe una opción que podemos usar para los casos de claves duplicadas: *ON DUPLICATE KEY UPDATE*. En este caso podemos indicar a **MySQL** qué debe hacer si se intenta insertar una fila que ya existe en la tabla. Las opciones son limitadas: no podemos insertar la nueva fila, sino únicamente modificar la que ya existe.

Por ejemplo, en la tabla 'ciudad3' podemos usar el último valor de población en caso de repetición:

```
mysql> INSERT INTO ciudad3 (nombre, poblacion) VALUES
    -> ('Madrid', 7000000);
Query OK, 1 rows affected (0.02 sec)
```

```
mysql> INSERT INTO ciudad3 (nombre, poblacion) VALUES
    -> ('París', 9000000),
    -> ('Madrid', 7200000)
    -> ON DUPLICATE KEY UPDATE poblacion=VALUES(poblacion);
Query OK, 3 rows affected (0.06 sec)
Records: 2  Duplicates: 1  Warnings: 0

mysql> SELECT * FROM ciudad3;
+--------+-----------+
| nombre | poblacion |
+--------+-----------+
| Madrid |   7200000 |
| París  |   9000000 |
+--------+-----------+
2 rows in set (0.00 sec)
mysql>
```

En este ejemplo, la segunda vez que intentamos insertar la fila correspondiente a 'Madrid' se usará el nuevo valor de población. Si en lugar de *VALUES(poblacion)* usamos poblacion el nuevo valor de población se ignora. También podemos usar cualquier expresión:

```
mysql> INSERT INTO ciudad3 (nombre, poblacion) VALUES
    -> ('París', 9100000)
    -> ON DUPLICATE KEY UPDATE poblacion=poblacion;
Query OK, 2 rows affected (0.02 sec)
mysql> SELECT * FROM ciudad3;
+--------+-----------+
| nombre | poblacion |
+--------+-----------+
| Madrid |   7200000 |
| París  |   9000000 |
+--------+-----------+
2 rows in set (0.00 sec)

mysql> INSERT INTO ciudad3 (nombre, poblacion) VALUES
    -> ('París', 9100000)
    -> ON DUPLICATE KEY UPDATE poblacion=0;
Query OK, 2 rows affected (0.01 sec)
```

```
mysql> SELECT * FROM ciudad3;
+--------+-----------+
| nombre | poblacion |
+--------+-----------+
| Madrid |   7200000 |
| París  |         0 |
+--------+-----------+
2 rows in set (0.00 sec)
mysql>
```

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}, ...

mysql> REPLACE INTO ciudad3 (nombre, poblacion) VALUES
    -> ('Madrid', 7200000),
    -> ('París', 9200000),
    -> ('Berlín', 6000000);
Query OK, 5 rows affected (0.05 sec)
Records: 3  Duplicates: 2  Warnings: 0

mysql> SELECT * FROM ciudad3;
+--------+-----------+
| nombre | poblacion |
+--------+-----------+
| Berlín |   6000000 |
| Madrid |   7200000 |
| París  |   9200000 |
+--------+-----------+
3 rows in set (0.00 sec)
```

En este ejemplo se sustituyen las filas correspondientes a 'Madrid' y 'París', que ya existían en la tabla y se inserta la de 'Berlín' que no estaba previamente.

Las mismas sintaxis que existen para INSERT, están disponibles para REPLACE:

```
mysql> REPLACE INTO ciudad3 VALUES ('Roma', 9500000);
Query OK, 1 rows affected (0.03 sec)
mysql> REPLACE INTO ciudad3 SET nombre='Londres', poblacion=10000000;
Query OK, 1 row affected (0.03 sec)
mysql> SELECT * FROM ciudad3;
+---------+-----------+
| nombre  | poblacion |
+---------+-----------+
| Berlín  |   6000000 |
| Londres |  10000000 |
| Madrid  |   7200000 |
| París   |   9200000 |
| Roma    |   9500000 |
+---------+-----------+
5 rows in set (0.00 sec)
mysql>
```

Selección de datos.

Ahora aprenderemos a extraer datos de una base de datos. Para ello vamos a usar la sentencia SELECT.

La sintaxis de SELECT es compleja, pero en este capítulo no explicaremos todas sus opciones. Una forma más general consiste en la siguiente sintaxis:

```
SELECT [ALL | DISTINCT | DISTINCTROW]
   expresion_select,...
   FROM referencias_de_tablas
   WHERE condiciones
   [GROUP BY {nombre_col | expresion | posicion}
        [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>mysql> SELECT * FROM gente;
+---------+------------+
| nombre  | fecha      |
+---------+------------+
| Fulano  | 1985-04-12 |
| Mengano | 1978-06-15 |
| Tulano  | 2001-12-02 |
| Pegano  | 1993-02-10 |
+---------+------------+
```

Limitar las columnas: proyección

Recordemos que 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. En la sintaxis que hemos mostrado, la selección de columnas corresponde con la parte "expresion_select". En el ejemplo anterior hemos usado '*', que quiere decir que se muestran todas las columnas.

Pero podemos usar una lista de columnas, y de ese modo sólo se mostrarán esas columnas:

```
mysql> SELECT nombre FROM gente;
+---------+
| nombre  |
+---------+
| Fulano  |
| Mengano |
| Tulano  |
| Pegano  |
+---------+
4 rows in set (0.00 sec)
mysql> SELECT clave,poblacion FROM ciudad5;
Empty set (0.00 sec)
```

Las expresiones_select no se limitan a nombres de columnas de tablas, pueden ser otras expresiones, incluso aunque no correspondan a ninguna tabla.

Sentencia DELETE, DROP y ALTER TABLE

Sentencia Delete.

El borrado se expresa de igual modo que una consulta. Se pueden borrar solo filas completas, es decir, no se pueden borrar valores de atributos concretos.

Sintaxis: *delete from r where p;*

r: representa una relación.

p: representa un predicado.

La declaración *delete* selecciona primero todas las filas en r para las que p es cierto y a continuación las borra de r. Si se omite la cláusula *where* se borran todas las filas de r.

La orden *delete* opera solo sobre una relación. Si se desea borrar filas de varias relaciones, se deberá utilizar una orden *delete* por cada relación. El predicado de la cláusula *where* puede ser tan complicado como otro *where* con cláusula *select*, o tan simple como una cláusula *where* vacía.

Por ejemplo: Borrar de la tabla ciudad3 el registro Madrid.

```
mysql> delete from ciudad3 where nombre = "Madrid";
Query OK, 1 row affected (0.00 sec)
mysql> select * from ciudad3;
+--------+-----------+
| nombre | poblacion |
+--------+-----------+
| berlin |   6000000 |
| Paris  |   9200000 |
+--------+-----------+
2 rows in set (0.00 sec)
```

Sentencia Drop.

Se utiliza para borrar una relación de una base de datos. Dicha orden borra de la base de datos toda la información sobre la relación eliminada.

Sintaxis: *drop table r;*

r: representa la relación a eliminar.

Esta instrucción tiene una repercusión más drástica que la instrucción: *delete from r.*

La diferencia en ambas instrucciones es que en la primera se borran todas las filas de la relación y la relación misma, en cambio, en la segunda solo se borra todas las filas, es decir, se conserva la relación.

Ejemplo: Borrar los registros de la relación ciudad3.

```
mysql> drop table ciudad3;
Query OK, 0 rows affected (0.03 sec)

mysql> select * from ciudad3;
ERROR 1146 (42S02): Table 'alumno.ciudad3' doesn't exist.
```

Nota: Como se ve fueron borrados tanto los registros de la tabla ciudad3 como ella misma.

Sentencia Alter Table.

Se utiliza para añadir atributos a una relación existente.

Sintaxis: *alter table r add A D;*

r: representa el nombre de la relación existente.

A: representa el nombre del atributo que se desea añadir.

D: representa el dominio del atributo A.

Además de añadir atributos a una relación se pueden eliminar utilizando la orden:

alter table r drop A; donde

r: representa el nombre de la relación.

A: representa el nombre de un atributo de la relación.

Ejemplo: Añadir a la tabla persona (numero,nombre,direccion) el atributo teléfono.

```
mysql> alter table persona add telefono varchar(8);
Query OK, 1 row affected (0.03 sec)
Records: 1  Duplicates: 0  Warnings: 0
mysql> select * from persona;
+--------+--------+-----------+----------+
| numero | nombre | direccion | telefono |
+--------+--------+-----------+----------+
|      1 | mario  | abc       | NULL     |
+--------+--------+-----------+----------+
1 row in set (0.01 sec)
```

Sentencia Update.

En determinadas situaciones puede ser deseable cambiar un valor dentro de una fila, sin cambiar todos los valores de la misma. Para este tipo de situaciones se utiliza la instrucción update. Al igual que ocurre con insert y delete, se pueden elegir las filas que van a ser actualizadas mediante una consulta.

Sintaxis: update r set o where p

r: representa una relación.

o: operación de actualización.

p: representa un predicado.

Ejemplo: Actualizar en un 2% los registros para los cuales la población sea mayor a 6000000 ubicados en la tabla ciudad3.

```
mysql> update ciudad3
    -> set poblacion = poblacion * 0.02
    -> where poblacion > 6000000;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> select * from ciudad3;
+--------+-----------+
| nombre | poblacion |
+--------+-----------+
| berlin |   6000000 |
| Paris  |    184000 |
+--------+-----------+
2 rows in set (0.00 sec)
```

Cláusulas WHERE, HAVING y ORDER BY.

WHERE: Sirve para aplicar condiciones (o restricciones) en la selección de datos.

Sintaxis: SELECT <nombre_cols> FROM <nombre_tablas> WHERE <condiciones_booleanas>;

Operadores de comparación aplicables en la cláusula WHERE: =, >, <, >=, <=, <>.

Operadores lógicos aplicables en la cláusula WHERE: AND / OR / NOT

Predicados aplicables en la cláusula WHERE:

<atributo> **BETWEEN** <limit_1>**AND** <limit_2> {Rango de valores}

<atributo> **LIKE** <expr> {cadena de caracteres } Comodines: '%', '_'

<atributo> **IS [NOT] NULL** {consultar si el atributo tiene valor o no}

GROUP BY: Existen situaciones en las cuales sería deseable aplicar las funciones de agregación no solo a un único conjunto de tuplas, sino también aún grupo de conjuntos de tuplas, esto se especifica usando la cláusula ***group by***. El atributo o atributos especificados en la cláusula ***group by*** se usan para formar grupos. Las tuplas con el mismo valor en todos los atributos especificados en la cláusula ***group by*** se colocan en un grupo.

HAVING: Se utiliza en aquellas consultas en las que es mas útil establecer una condición que se aplique a los grupos construidos por la cláusula ***group by*** que una que se aplique a las tuplas.

Los predicados de la cláusula ***having*** se aplican después de la formación de grupos, de modo que se pueden usar las funciones de agregación.

ORDER BY: Hace que las tuplas resultantes de una consulta se presenten en un cierto orden. De manera predeterminada, la cláusula ***order by*** lista los elementos en orden ascendente. Para especificar el tipo de ordenación se puede incluir la cláusula ***desc*** para orden descendente ó ***asc*** para orden ascendente.

Sintaxis: SELECT <nombre_cols> FROM <nombre_tablas> [WHERE <condiciones_booleanas>] **ORDER BY <atributo_1>, ..., <atributo_N>;**

Funciones de Agregación SUM, MAX, MIN, COUNT y AVG

Son funciones que toman una colección (un conjunto o multiconjunto) de valores como entradas y producen un único valor como salida.

La entrada a ***SUM*** y ***AVG*** debe ser una colección de números, pero las otras funciones pueden operar sobre colecciones de datos de tipo no numéricos, tales como: cadenas.

COUNT (<fila>) {devuelve el total de filas seleccionadas}

SUM (<columna>) {suma los valores de una columna}

MIN (<columna>) {devuelve el valor mínimo de la columna}

MAX (<columna>) {devuelve el valor máximo de la columna}

AVG (<columna>) {devuelve la mediana de la columna}

6. Entorno de desarrollo (Gambas).

6.1 Introducción.

Gambas fue inicialmente creado por Benoit Minisini, un programador con experiencia en la escritura de compiladores que estaba harto de luchar contra los fallos de diseño de Visual Basic, y deseaba usar un entorno de GNU/Linux fácil en su distribución, comenzó a desarrollar su propio entorno para Linux basado en BASIC.

El 28 de febrero de 2002 puso en Internet la primera versión publica de Gambas: gambas 0.20 Benoit eliminó del diseño del lenguaje bastantes de los problemas que Visual Basic tenía, como la gestión de errores, y le añadió características comunes en los lenguajes actuales más modernos, como la orientación a objetos y la propia estructura de los programas. Como prueba de fuego, el propio entorno de desarrollo fue programado en Gambas desde la primera versión, sirviendo a un tiempo de demostración de la potencia del lenguaje y de detección de necesidades y corrección de errores que se fueron incorporando a las distintas versiones.

GAMBAS, quiere decir, ***Gambas almost means BASIC***, Gambas abre el entorno de la programación visual en Linux a todo el mundo, como lo hizo en su día Visual Basic en Windows. La ampliación del lenguaje BASIC alcanza con Gambas amplias cotas de potencia, profesionalidad y modernidad, sin abandonar nunca la sencillez y claridad de este lenguaje de programación de alto nivel.

Gambas no es sólo un lenguaje de programación, es también un entorno de programación visual para desarrollar aplicaciones gráficas o de consola. Hace posible el desarrollo de aplicaciones complicadas muy rápidamente.

El programador diseña las ventanas de forma gráfica, arrastra objetos desde la **Caja de Herramientas** y escribe código en Basic para cada objeto. Gambas está orientado a eventos, lo que significa que llama automáticamente a los procedimientos cuando el usuario de la aplicación elige un menú, hace clic con el ratón, mueve objetos en la pantalla, etc.

Gambas está bajo la licencia pública GNU. Gambas se ejecuta en la mayoría de las plataformas Linux y la actual versión estable es la Release 1.0.9.

Un entorno libre.

Gambas es un entorno de desarrollo que se distribuye con la licencia GPL GNU (General Public Licence). Esto significa que se distribuye siempre con el código fuente y respeta las cuatro libertades que define la Free Software Foundation:

- La libertad de usar el programa con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa y adaptarlo a las propias necesidades (libertad 1). El acceso al código fuente es una condición previa para esto.
- La libertad de distribuir copias, con las que se puede ayudar al vecino (libertad 2).
- La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie (libertad 3). El acceso al código fuente es un requisito previo para esto.

Uno de los engaños más comunes en el uso de Software Libre es la creencia de que este modelo de desarrollo obliga a que el trabajo se publique gratis, lo que es del todo incierto. Estas cuatro libertades permiten que, quien lo desee, venda copias de

Gambas (entregando siempre el código fuente y respetando esas cuatro libertades) y, por supuesto, de cualquier aplicación desarrollada con este programa. Las aplicaciones desarrolladas con Gambas pueden o no acogerse a la licencia GPL.

También cualquier programador es libre de alterar el propio lenguaje y modificarlo a su gusto, siempre y cuando entreguen el código correspondiente a esas modificaciones y respete los derechos de autor de los desarrolladores originales.

Aparte de estas libertades propias de la naturaleza de un proyecto de Software Libre sobre GNU/Linux, Gambas añade más facilidades para el programador:

- Una ayuda muy completa del lenguaje y cada uno de los componentes, algo que es muy de agradecer para los que empiezan a programar en Gambas, y que no es habitual en los proyectos de Software Libre.

- Una API (Interfaz para programar la aplicación) sencilla y bien documentada, lo que facilita a los programadores crear nuevos componentes para Gambas.

La API no es de utilidad inmediata para quien desarrolle con este lenguaje, pero permite a los programadores avanzados que lo deseen añadir funcionalidades al entorno de desarrollo y crear nuevas herramientas para Gambas.

Elementos de Gambas.

Para poder desarrollar y ejecutar programas hechos con Gambas, son necesarios distintos elementos:

- Un **Compilador,** que se encargará de transformar todo el código fuente y archivos que formen parte de un proyecto hecho en Gambas, en un programa ejecutable.
- Un **Intérprete** capaz de hacer que los programas hechos en Gambas sean ejecutados por el sistema operativo.
- Un **Entorno de desarrollo** que facilite la programación y diseño de las interfaces gráficas de los programas.
- **Componentes** que añaden funcionalidades al lenguaje. La palabra *componente* en Gambas tiene un significado específico, ya que no alude a partes genéricas, sino a librerías específicas que le doten de más posibilidades. En la actualidad existen componentes para usar xml, conexiones de red, opengl, sdl, ODBC, distintas bases de datos, expresiones regulares, escritorios basados en qt, en gtk, etc. Estos componentes son desarrollados por distintos programadores, siguiendo las directrices de la API de Gambas y la documentación publicada por Benoit Minisini.

Cómo obtenerlo.

Las nuevas versiones de Gambas se publican a través de la página oficial del proyecto: **http://gambas.sourceforge.net.** En la actualidad existen dos ramas de Gambas: la rama estable o 1.0 y la rama de desarrollo o 1.9 que desembocará en la versión 2.0. De la rama estable sólo se publican nuevas versiones cuando es para corregir algún fallo que se halla descubierto.

En el momento de escribir estas líneas, la versión estable era la 1.0.11 y no se prevé que haya cambios en el futuro. La versión de desarrollo está en continuo cambio, mejora y adición de nuevos componentes. Esto la hace muy atractiva, debido a la existencia de importantes funcionalidades que no están en la versión estable (como los componentes gtk y ODBC). Sin embargo, al ser una rama en desarrollo, es muy probable que tenga fallos no descubiertos y es seguro que sufrirá cambios en un futuro próximo.

Las nuevas versiones se publican siempre en forma de código fuente, para que los usuarios que lo deseen compilen el código y obtengan todas las partes que Gambas tiene. Los autores de algunos de los componentes que se han hecho para Gambas, publican de forma separada en distintos sitios Web las versiones nuevas de estos, pero todas se envían a Benoit Minisini y pasan a formar parte de la publicación completa de este lenguaje de programación en la siguiente versión. De este modo, se puede decir que cuando Benoit hace pública una nueva, el paquete del código fuente contiene las últimas versiones de todo el conjunto en ese momento.

Como la compilación de Gambas y todos los componentes asociados pueden ser una tarea difícil para usuarios no expertos, es común que se creen paquetes binarios con la compilación ya hecha y listos para ser instalados en distintas distribuciones de gnu/Linux.

En la misma página web donde se puede bajar el código fuente se encuentran los enlaces para la descarga de los paquetes compilados para estas distribuciones.

6.2. Compilación y Dependencias

Si en lugar de instalar paquetes ya compilados para la distribución de gnu/Linux deseamos compilar Gambas desde el código fuente, deberemos seguir los pasos habituales en los sistemas GNU. Es decir, descomprimir el archivo con las fuentes y, desde el directorio que se crea al descomprimir y usando un terminal, ejecutar las siguientes instrucciones:

```
./configure
make
make install
```

La última de ellas debemos hacerla como root, si queremos que el programa este disponible para todos los usuarios del ordenador. Si estamos habituados a compilar aplicaciones en sistemas GNU, disponemos ya de un compilador instalado y de bastantes librerías de desarrollo. Las instrucciones anteriores trataran de compilar e instalar todos los componentes de Gambas, que son muchos. Si no tenemos las librerías correspondientes a algunos de ellos, simplemente no se compilaran y la instrucción ./configure nos informará de ello. Es importante saber que el entorno de desarrollo esta hecho sobre las librerías graficas qt, por tanto, para poder usar el entorno necesitaremos tener instalado, al menos, estas librerías de desarrollo con una versión igual o superior a la 3.2. La versión del compilador gcc ha de ser también ésta, como mínimo.

Cada uno de los componentes tiene dependencias de sus propias librerías y dependerá de la distribución de Linux que usemos, para saber el nombre del paquete que deberemos instalar antes de poder realizar la compilación.

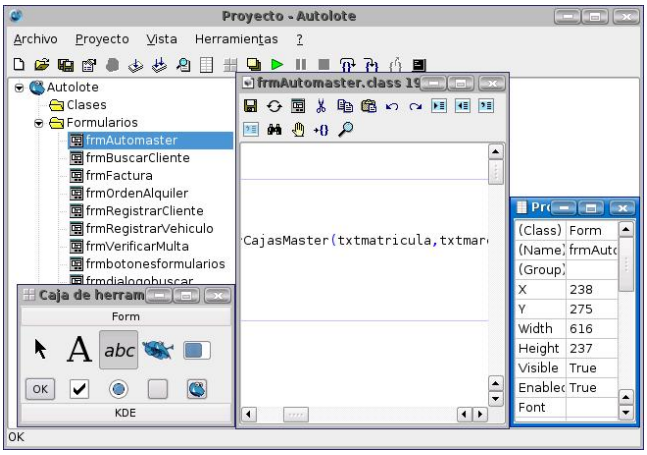
6.3 Familiarizarse con el IDE

Cuando se arranca Gambas, lo primero que nos aparece es la ventana de bienvenida.

Aquí se nos ofrece la opción de comenzar un nuevo proyecto o aplicación, abrir un proyecto del que tengamos sus archivos disponibles, abrir uno usado recientemente o uno de los numerosos ejemplos que están incluidos en la ayuda de Gambas.

Antes de elegir cualquiera de estas opciones es necesario saber que todos los códigos fuentes de una aplicación hecha en gambas es lo que se denomina *proyecto.* El proyecto esta formado por una serie de archivos que en Gambas están SIEMPRE situados dentro de un único directorio. En el puede haber, a gusto del desarrollador, distintos subdirectorios y organizar todo como se desee, pero cualquier grafico, texto y código que forme parte de la aplicación estará dentro de el. Por ello, si elegimos en esta ventana la opción **Nuevo Proyecto…,** el asistente siempre creara un nuevo directorio con el nombre del proyecto y ahí ira introduciendo todos los archivos necesarios para el desarrollo de la aplicación. Así, para enviar a alguien el código fuente de una aplicación hecha en Gambas o cambiarla de ordenador o disco, solo hay que transportar el directorio con el nombre del proyecto, sin tener que preocuparse de otros archivos. Del mismo modo, si desde el entorno de desarrollo escogemos un archivo o un grafico para integrarlo en nuestro trabajo, el archivo será copiado automáticamente al directorio del proyecto.

Aunque un programa en Gambas se podría hacer perfectamente usando un editor de texto cualquiera, sería un desperdicio no aprovechar uno de los mayores atractivos que el lenguaje tiene: su IDE o entorno de desarrollo. El IDE de Gambas ahorra al programador buena parte del trabajo más tedioso, le proporciona herramientas que hacen mucho más fácil su tarea, con utilidades de ayuda, de diseño de interfaces, auto completado de instrucciones, traducción de programas, etc. En la ventana siguiente podemos ver algunas de las ventanas más importantes del entorno, que se usan durante el desarrollo de una aplicación.



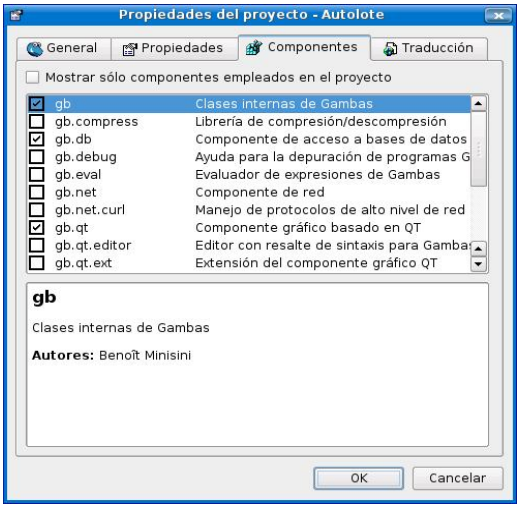
6.4. Sistema de componentes.

La interfaz desarrollada por Benoit para su programación hace que hayan sido varios los programadores que han colocado con él, desarrollando nuevos componentes que se van añadiendo a las distintas versiones de Gambas en cada nueva aplicación. En la versión 1.0 estable de Gambas sólo se podían desarrollar en C y C++, pero desde la versión 1.9.4 de la rama de desarrollo también se pueden escribir componentes en Gambas, lo que abre numerosas posibilidades futuras ya que es mucho más sencillo hacerlo que en C.

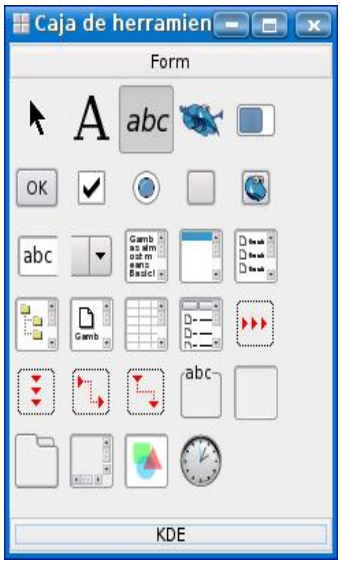
El listado de componentes disponibles es amplio y se aumenta continuamente en la versión de desarrollo.

Estos componentes pueden verse en la pestaña **Componentes**, accesibles a través del menú **Proyecto | Propiedades.** Cada uno de los componentes se corresponde a un paquete compilado en la distribución, de forma que si se añade al proyecto, por ejemplo, el componente gb.sdl, ha de instalarse el paquete gambas-gb-sdl en los ordenadores que se quiere ejecutar la aplicación compilada.

Al hacer clic sobre cada uno de los componentes aparece una pequeña descripción de sus funciones, el nombre del autor o autores del componente y un listado de los controles que están disponibles para el desarrollador si selecciona ese componente.



Los controles son clases para crear objetos útiles en la programación. Los objetos creados pueden ser visuales (como pestañas, editores de texto, etc.) u objetos de código (como servidores de red o conexiones a bases de datos). Si el componente tiene objetos visuales, estos se incorporan a algunas de las pestañas de la **caja de herramientas** del entorno de desarrollo.



En las imágenes anteriores podemos ver algunos de los objetos gráficos que están disponibles al seleccionar los distintos componentes.

Cada componente tiene su propia documentación y se encuentra en la ayuda de Gambas. En la rama estable esta documentación está siempre disponible, en la rama de desarrollo solo esta disponible si ha sido seleccionado par su uso en el proyecto.

Todas las cosas que se pueden hacer con gambas y no son parte del propio lenguaje BASIC, se programan mediante el uso de componentes. Esto significa que, por ejemplo, para hacer una aplicación de bases de datos es necesario seleccionar el componente gb.db o no estarán disponibles los objetos de conexión a bases de datos. Lo mismo ocurre con las conexiones de red, captura de video, etc. Estos objetos no son parte del lenguaje BASIC.

VI. METODOLOGIA DEL TRABAJO

El método de desarrollo que hemos seleccionado para la elaboración de la aplicación**:** Diseño de Guías de Laboratorios para Base de Datos I bajo plataforma Linux, es el método **ciclo de vida clásico** o **en cascada** ya que resulta conveniente en la identificación de las actividades ha seguir en la elaboración de la aplicación.

Las etapas a desarrollar durante el proceso investigativo son:

1. Investigación preliminar o ingeniería en sistemas
2. Análisis de requisitos del software
3. Diseño del sistema
4. Codificación
5. Prueba
6. Implementación y evaluación

Cada una de las etapas lleva asociada una serie de tareas que deberán realizarse y una serie de documentos que serán las salidas de cada una de estas fases y que servirán de entrada a la siguiente fase, de esta manera se logra progresar a través del análisis, diseño, codificación, prueba e implementación.

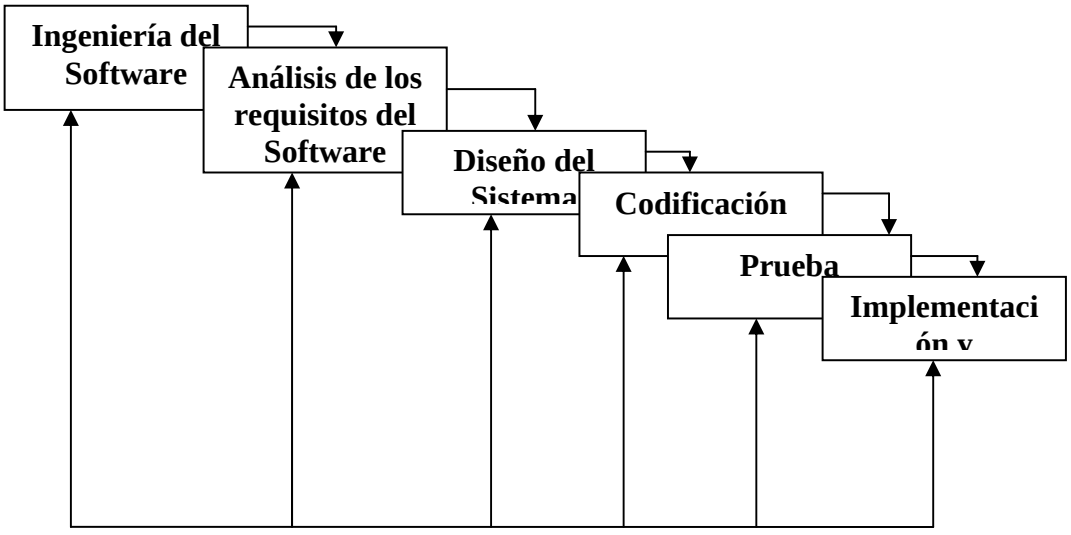


Figura No. 1

1. **Investigación preliminar o ingeniería del sistema:** en esta fase se realiza el estudio de factibilidad técnica, económica y operacional de la aplicación.

2. **Análisis de requisitos del software:** se debe comprender la funcionalidad que debe tener la aplicación, así como su rendimiento y la interfaz requerida.

3. **Diseño del sistema:** se diseñan procedimientos precisos para especificar la interfaz que utilizara la aplicación.

4. **Codificación:** el diseño se traduce de forma precisa a la computadora, utilizando un lenguaje de programación de alto nivel, en nuestro caso BASIC. Se debe codificar los ejercicios resueltos y propuestos, además de generar la interfaz grafica de cada uno de ellos.

5. **Prueba:** una vez generado el código respectivo para cada una de las guias, comienza la fase de prueba de cada una de ellas, confirmando que se puede ejecutar cada una de las prácticas propuestas.

6. **Implementación y evaluación:** una vez realizada la fase de prueba y después de haber hechos las correcciones en caso de ser necesario, se procede a la publicacion de las guias.

VII. RECURSOS DISPONIBLES Y NECESARIOS

1. Recursos Hardware

Disponible: Para la elaboración de nuestro trabajo monografico hemos utilizado una PC con las siguientes caracteristicas:

- Procesador: Intel pentium IV de 2.4 GHz
- Disco Duro: Seagate de 40 GB
- Memoria Principal: 256 MB, DDR
- Unidad de CD/ ROM de 56x
- Unidad de Disco de 3 ½

Requerido: para el desarrollo de la practica 0 se requiere una PC con conexion a internet y como minimo debe tener 256 MB de memoria RAM.

Para el desarrollo de las otras guias propuestas se requiere una PC con caracteristicas semejantes o superiores a las de la PC utilizadas para elaborar dichas guias.

2. Recursos Software

a. Kubuntu Linux versión 6.06: Sistema operativo

b. Gambas : Entorno de desarrollo para KDE-Linux

c. DIA (Editor de diagrama): Herramienta utilizada para elaboración diagramas E-R

d. MYSQL 5.0: Sistema Gestor de Base de Datos

e. Visual Basic: Entorno de desarrollo para Windows

VIII. FASE DE ANALISIS

1. Introducción.

1.1 Propósito.

La definición del conjunto de especificaciones de los requisitos del software que debe cumplir la aplicación “Prácticas de laboratorio para Base de Datos I bajo entorno Linux”, consiste en la elaboración de una serie de ejercicios resueltos y propuestos para los estudiantes que llevan el curso de Base de Datos I, diseñados bajo plataforma Linux y presentados a través de una pequeña pagina Web.

1.2 Alcance.

El nombre de la aplicación será: Diseño de prácticas de laboratorios para Base de Datos I bajo entorno Linux.

La aplicación de dichas prácticas permitirá:

- Conocer una nueva plataforma de trabajo (Linux)
- Realizar la esquematización del diagrama E-R en un potente sistema gestor de bases de datos como lo es MySQL.
- Realizar consultas MySQL.
- Conocer un nuevo entorno para diseño de tablas en plataforma Linux (DIA).
- Conocer un nuevo entorno de desarrollo para aplicaciones con acceso a bases de datos en plataforma Linux (Gambas).
- Realizar aplicaciones con acceso a una Base de Datos MySQL.
- Generación de Informes

1.3 Referencias.

Abraham Silberschatz. “Fundamentos de Bases de Datos”. Tercera Edición.

Sitio oficial de Ubuntu con enlaces a otros sitios de soporte:
http://www.ubuntu.com/.

Pagina oficial de Gambas: http://www.gambassourceforge.net

Ceballos, Francisco Javier Enciclopedia de Visual Basic.

Manual de referencia de MySQL.: http://dev.mysql.com/doc/.

www.mailxmail.com/curso/informatica/datareport

1.4 Visión General.

Primeramente se realizará una descripción general de la aplicación y luego se abordará cada uno de los requisitos específicos.

2. Descripción General.

2.1 Relaciones de la aplicación.

Las prácticas de laboratorio se presentaran a través de una pequeña página Web a la cual se tendrá acceso desde el sitio Web del docente encargado de impartir el curso de Base de Datos I.

2.2 Funciones de la aplicación.

El diseño de cada una de las guías de laboratorio esta hecho de acuerdo a cada uno de los conocimientos teóricos que el alumno recibirá en las clases teóricas impartidas por el docente encargado de la asignatura.

a) Instalación del Sistema Operativo: Kubuntu 6.06.
b) Instalación del entorno de desarrollo y Sistema Gestor de BD (Gambas y MySQL).
c) DDL de MySQL: Modelo Entidad – Relación.
d) DML en MySQL: Consultas y Modificación de Datos.
e) Componentes del entorno de desarrollo Gambas.
f) Diseño de Aplicaciones con acceso a Bases de Datos.
g) Generación de Informes.

2.3 Características del usuario.

Los usuarios finales de la aplicación serán los estudiantes de Ingeniería en Sistemas de Información e Ingeniería Telemática que llevan el curso de Base de Datos I.

2.4 Restricciones generales.

a) El lenguaje a utilizar será BASIC
b) El sistema gestor de Base de Datos será MySQL
c) El estudiante solo tendrá acceso a las guías de laboratorios desde el sitio Web del profesor.

3. Requisitos específicos.

3.1 Requisitos funcionales.

3.1.1 Instalación del sistema Operativo Kubuntu 6.06.

3.1.1.1 Especificación.

3.1.1.1.1 Introducción.

Los requerimientos mínimos de hardware para instalar este sistema son por lo menos 256 de RAM y un buen procesador.

Recomendación: Realizar la instalación sin conexión a Internet.

3.1.1.1.2 Entradas.

Insertar en la unidad de CD el CD de instalación de Kubuntu.

3.1.1.1.3 Proceso.

- Iniciar Kubuntu para cargar los archivos y arrancar en modo Live.
- Clic en la opción instalar o install.
- Selección del Idioma: Spanish.
- Configurar ciudad, país y zona horaria.
- Distribución del teclado.
- Registrar Usuario.
- Selección del Disco.
- Particionamiento del disco.
- Preparando puntos de montaje.

3.1.1.1.4 Salidas.

Aparecerá una ventana, que dirá *La instalación se ha completado,* si la instalación tuvo éxito, en caso contrario pueden darse algunos errores que no permitirán completar la instalación.

3.1.1.2 Interfaces Externas.

3.1.1.2.1 Interfaces de Usuario.

La captura de la configuración del idioma, teclado, zona horaria, registro de usuario y Particionamiento de disco.

3.1.1.2.2 Interfaces Hardware.

Se podrá utilizar cualquier ordenador que tenga los requerimientos mínimos de hardware para la instalación.

3.1.1.2.3 Interfaces Software.

El proceso interactúa directamente con el Ordenador.

3.1.2 DDL de MySQL.

3.1.2.1 Especificación.

3.1.2.1.1 Introducción.

Una vez instalado el sistema gestor de bases de datos (MYSQL), se procederá a realizar la creación de tablas y bases de datos**.**

3.1.2.1.2 Entradas.

Iniciar sesión en MySQL desde una terminal

3.1.2.1.3 Proceso.

Aplicar los comandos necesarios para la creación de tablas y bases de datos.

3.1.2.1.4 Salidas.

Almacenamiento de las bases de datos con sus respectivas tablas en el Sistema Gestor MySQL.

3.1.2.2 Interfaces Externas.

3.1.2.2.1 Interfaces de usuario.

Una terminal KDE.

3.1.2.2.2 Interfaces Hardware.

Se podrá utilizar cualquier ordenador que tenga instalado el sistema gestor de Base de Datos MySQL.

3.1.2.2.3 Interfaces Software.

El proceso interactuará directamente con el sistema gestor MySQL.

3.1.3 DML de MySQL.

3.1.3.1 Especificación.

3.1.3.1.1 Introducción.

Una vez creada una base de datos con sus respectivas tablas se podrán efectuar consultas MySQL y modificación de los datos contenido en las mismas.

3.1.3.1.2 Entradas.

Iniciar sesión en MySQL desde una terminal.

3.1.3.1.3 Proceso.

Aplicar las sentencias, cláusulas y funciones necesarias para realizar las consultas y modificación de datos.

3.1.3.1.4 Salidas.

Almacenamiento de las modificaciones en el sistema gestor y visualización de los datos requeridos en las consultas a través de la terminal.

3.1.3.2 Interfaces Externas.

3.1.3.2.1 Interfaces de usuario.

Un terminal KDE.

3.1.3.2.2 Interfaces Hardware.

Se podrá utilizar cualquier ordenador que tenga instalado el sistema gestor de Base de Datos MySQL.

3.1.3.2.3 Interfaces Software.

El proceso interactuará directamente con el sistema gestor MySQL.

3.1.4 Componentes del entorno de desarrollo Gambas.

3.1.4.1 Especificación.

3.1.4.1.1 Introducción.

Aunque un programa en Gambas se podría hacer perfectamente usando un editor de texto cualquiera, sería un desperdicio no aprovechar uno de los mayores atractivos que el lenguaje tiene: su IDE o entorno de desarrollo.

3.1.4.1.2 Entradas.

Conjunto de herramientas que hacen mucho más fácil la tarea del programador, con utilidades de ayuda, de diseño de interfaces, auto completado de instrucciones, traducción de programas, etc.

3.1.4.1.3 Proceso.

Creación de aplicaciones sencillas utilizando las herramientas del entorno.

3.1.4.1.4 Salidas.

Interfaces gráficas creadas como: formularios, informes, etc.

3.1.4.2 Interfaces Externas.

3.1.4.2.1 Interfaces de usuario.

Herramientas del entorno de desarrollo Gambas.

3.1.4.2.2 Interfaces Hardware.

Se podrá utilizar cualquier ordenador que tenga instalado el sistema gestor de Base de Datos MySQL y el entorno de desarrollo Gambas.

3.1.4.2.3 Interfaces Software.

El proceso interactuará directamente con el entorno de desarrollo Gambas.

3.1.5 Diseño de Aplicaciones con acceso a Bases de Datos.

3.1.5.1 Especificación.

3.1.5.1.1 Introducción.

MySQL es uno de los más potentes sistemas gestores de bases de datos que existen actualmente, por lo cual lo utilizaremos como gestor en aquellas aplicaciones que necesiten acceder a una base de datos MySQL.

3.1.5.1.2 Entradas.

Realización de la interfaz grafica de la aplicación con las herramientas propias del entorno de desarrollo Gambas.

3.1.5.1.3 Proceso.

Establecimiento de la conexión con la base de datos de la aplicación y dar funcionalidad a cada uno de los componentes u objetos de la interfaz gráfica.

3.1.5.1.4 Salidas.

Interfaces graficas como: formularios, cajas de dialogo y mensajes las cuales contienen datos o información requerida por el usuario.

3.2 Requisitos de funcionamiento

Requisitos estáticos: el número máximo de terminales o de usuarios trabajando de forma simultánea con la aplicación estará en función al número de estudiantes que deseen visitar el sitio Web donde se encuentran las prácticas de laboratorios.

3.3 Restricciones de diseño

3.3.1 Atributos

3.3.1.1 Seguridad

Las guías de laboratorios que se encontraran disponibles en el sitio Web no tendrán ningún tipo de seguridad. Por tanto se omiten tareas como: caídas de la red, fallo en el sistema eléctrico, etc.

IX. CODIFICACION

PRACTICA 0:
INSTALANDO EL SISTEMA OPERATIVO
KUBUNTU VER. 6.06

Introducción

Kubuntu es una de las tantas distribuciones Linux, la cual se diferencia de las demás, por algunas razones, como por ejemplo de Ubuntu 5.10, por su instalación con entorno grafico amigable y no el típico sistema de instalación fuente rígido con múltiples pantallas de letras y fondo poco amigable a nuestra vista.

Los Requerimientos mínimos de Hardware son 256 de RAM y un buen procesador.

Arranque

Insertar en la unidad de CD el CD de instalación de Kubuntu y configurar nuestro boot desde la Bios para poder arrancar desde la unidad de CD si no lo tenemos configurado.

Al arrancar el CD, nos aparece en la primera pantalla, las siguientes opciones:

Iniciar Kubuntu
Iniciar Modo grafico Seguro
Verificar si el CD contiene errores
Test de Memoria
Arrancar desde el primer Disco Duro

Seleccionar la primera opción, ***Iniciar Kubuntu*** para que se carguen los archivos y arrancar en modo Live.

Una vez cargados todos los archivos, se muestra un entorno grafico parecido al escritorio de Windows como se muestra en la siguiente gráfica:

- **Instalación**

Entonces dar clic en la opción instalar o install que se encuentra en el escritorio.

2.1. Selección del Idioma: Spanish

Luego se desplegará una ventana de Bienvenida para seleccionar el idioma, como se muestra a continuación:

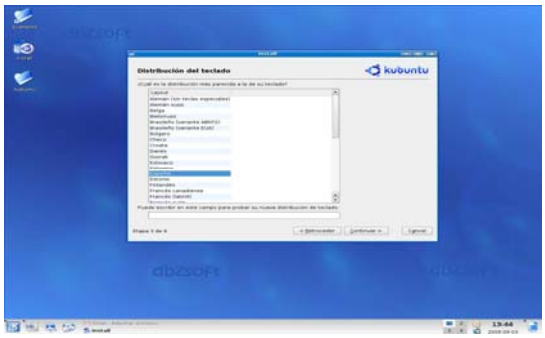
Una vez seleccionado el idioma dar clic en continuar

2.2. Ubicación

Aquí hay que configurar ciudad, país y zona horaria.

2.3. Distribución del teclado

Escoger la configuración de idioma del teclado

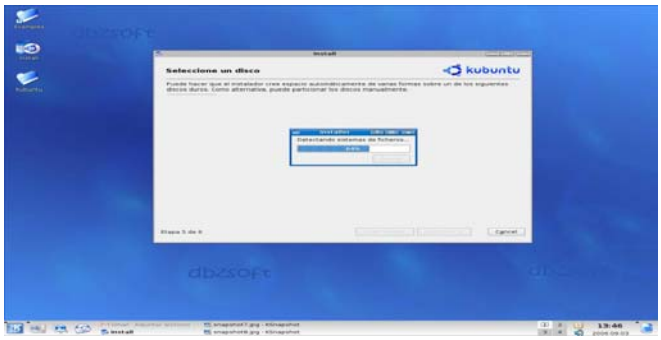


2.4. Registrar Usuario

Una vez seleccionado el idioma de teclado dar clic en continuar. Aparecerá una ventana para colocar tu nombre, Nombre de inicio de sesión (este debe ser en minúsculas) y tu contraseña de sesión.

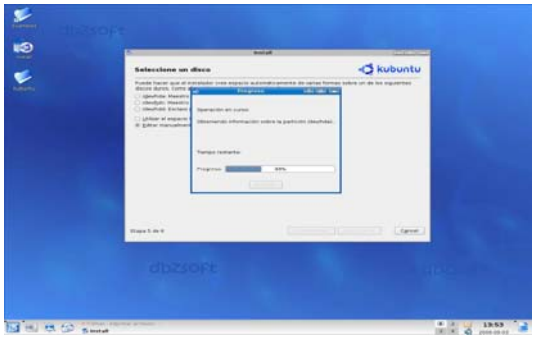
2.5. Selección del Disco

Seleccionar el disco donde se va a instalar el sistema operativo y realizar las particiones.

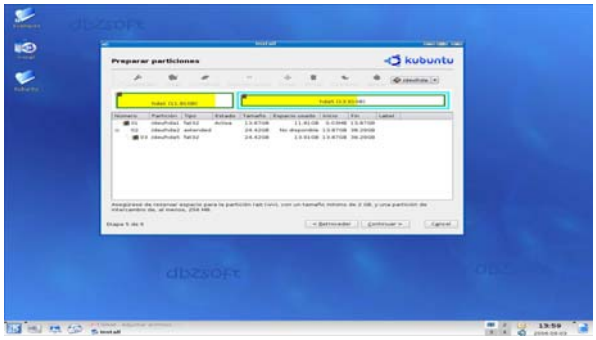


En la segunda pantalla, hay que tener cuidado de escoger las opciones correctas, en este caso seleccionar la última opción que es: ***editar manualmente la tabla de particiones*** y dar clic en continuar.

Aparecerá una ventanita mostrando la operación en curso como la siguiente:



Después de verificar el o los discos, aparecerá una ventana en donde se pueden observar las particiones que tenemos en cada uno de ellos, como se ve en la pantalla siguiente:

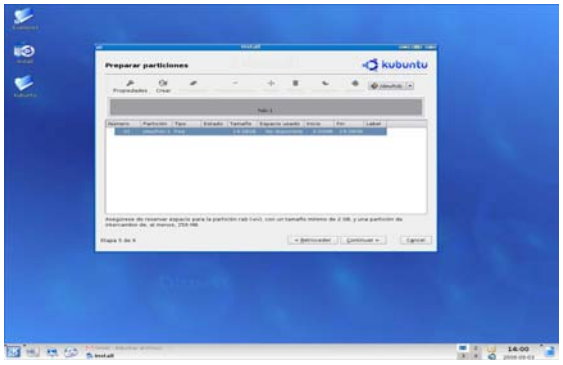


Nota: En la parte superior derecha aparece un desplegable que permite escoger el disco duro en el cual queremos instalar nuestro sistema.

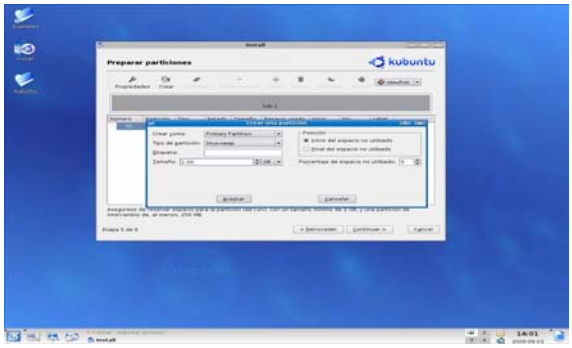
Nota 2: En este caso se instalará el sistema desde un disco duro totalmente desocupado pero para instalarlo desde un equipo con Windows, primero hay que redimensionar la partición, luego, seguir los pasos que mostramos a continuación dentro del nuevo espacio libre que crearon al redimensionar.

2.6. Particionamiento del disco

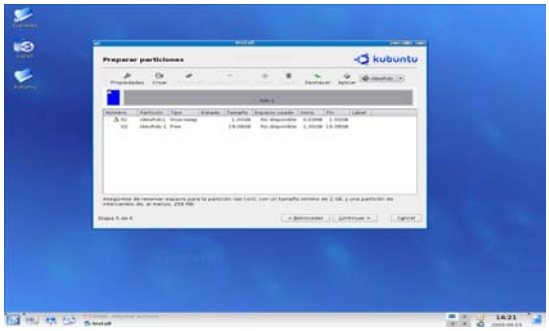
Una vez elegido el disco duro (o el espacio libre que se creó al redimensionar la partición original), seleccionar la partición para activar la opción ***crear*** que es parecida a una hojita, la cual permitirá crear las particiones.



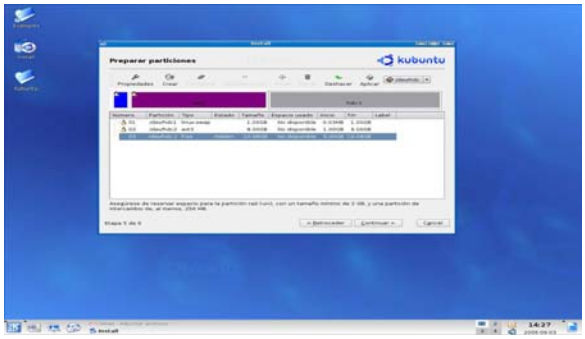
Primero hay que crear una partición primaria tipo Linux-Swap con un *tamaño del doble* de capacidad que tenga la PC de memoria RAM, Ej. Si tiene 256Mb, crear la partición con 512Mb.



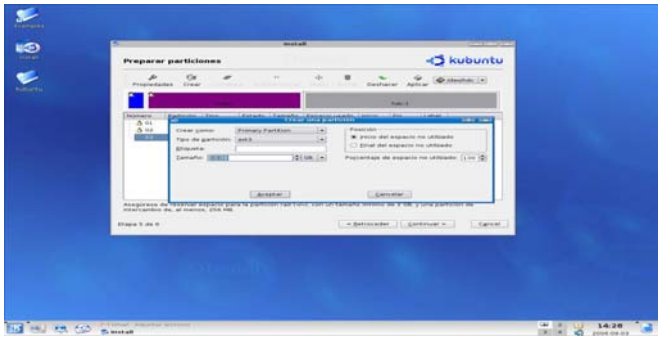
Al terminar dar clic en aceptar para ver la primera partición casi creada.



Luego hay que crear una segunda partición primaria tipo **Ext2,** cuyo tamaño estará en función del espacio que se tenga disponible para instalar el sistema y los programas que se vayan a instalar.

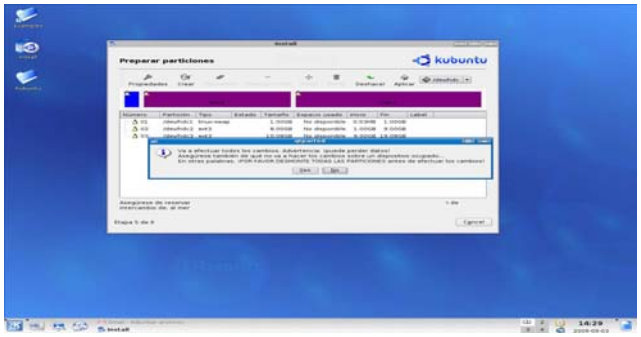


Por ultimo pueden crear una tercera partición primaria tipo **Ext3** para /home , la cual servirá para guardar todos sus archivos, algo por defecto en Linux (videos, mp3, documentos, etc). Para ella utilizamos todo el espacio restante.

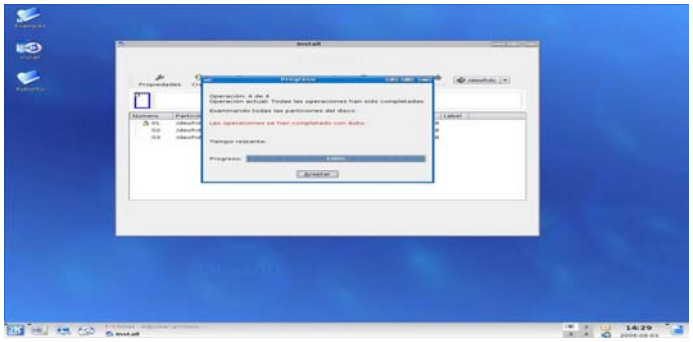


Esta partición no es obligatoria, pero recomendable por si hay inconvenientes con el sistema operativo y se requiere borrar la partición, de tal manera que los archivos no se borren junto con el sistema.

Ya finalizadas las dos o tres particiones, hacer clic en continuar y aparecerá una advertencia para no borrar una partición ocupada, pero como estamos seguros hacemos clic en si.

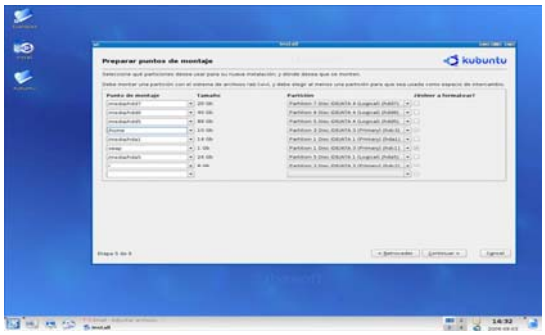


Al dar clic en **SI** el sistema empieza a realizar los cambios y aparecerá una pantalla la cual dirá: *Las operaciones se han realizado con éxito; entonces dar clic en aceptar.*

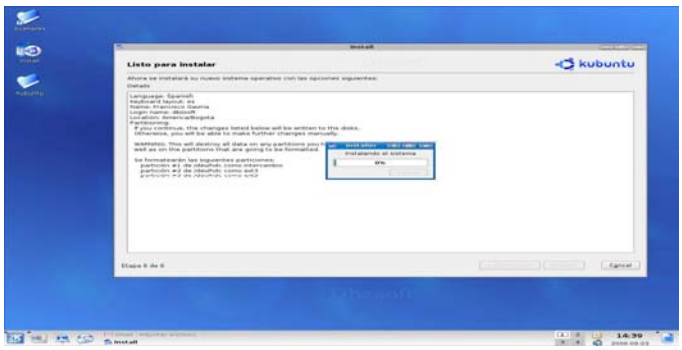


2.7. Preparando puntos de montaje

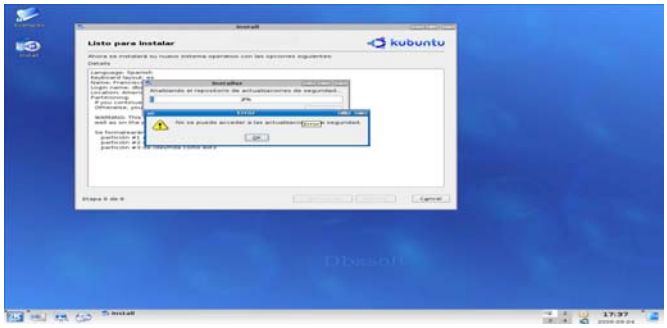
Luego de crear las particiones, hay que elegir las particiones en donde se montará el sistema de archivo raíz “/” (segunda partición creada), el de espacio de intercambio que es la partición Swap (primera partición creada) y /home (tercera partición creada).



Luego de haber seleccionado las particiones a usar, dar clic en continuar y el sistema empieza la detección de ficheros. Al terminar, muestra una ventana en donde aparecerán las opciones que escogimos para instalar el sistema operativo.

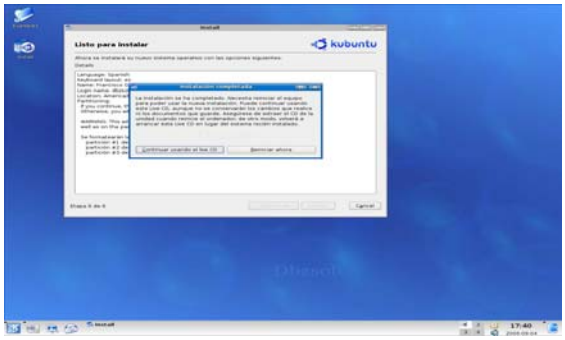


Luego aparecerá un mensaje de error (por no estar conectado a Internet), el cual dice que no se puede acceder a las actualizaciones de seguridad. Hacemos clic en ***OK***.



2.8. Finalizando la instalación

Por último aparecerá una ventana, que dirá: "*La instalación se ha completado, necesita reiniciar el equipo para poder usar la nueva instalación*", y hacemos clic en **"Reiniciar Ahora".**



3. Posibles errores

En la instalación de kubuntu es preferible desconectar la conexión a Internet ya que puede ocurrir lo siguiente:

- Que la instalación no prosiga y se quede en una ventana que dice "Escaneando el espejo" con la barra de un 1% indefinidamente.

- Que el sistema empiece a descargar los paquetes de idiomas y se demore mucho tiempo.

Y si se intenta cancelar la actualización, aparecerá una ventana con varios errores los cuales no permitirán continuar con la instalación.

Instalación de Paquetes

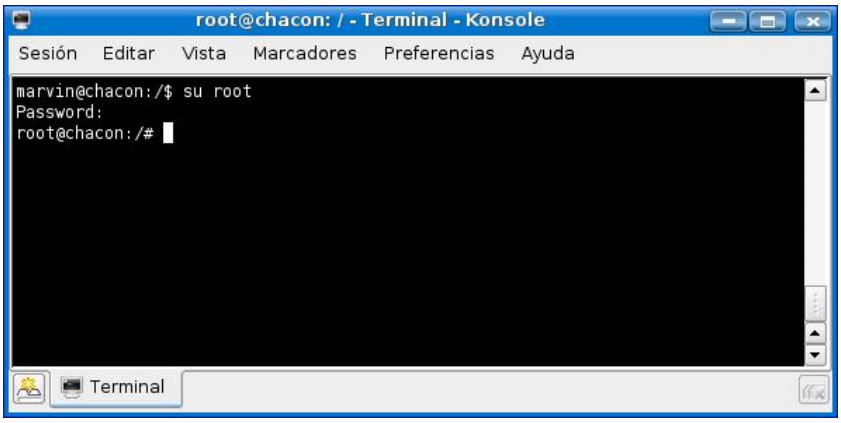
Gambas versión 1.0.13

MySQL 5.0.21

Editor de Diagramas DIA

Instalación de Gambas.

Para comenzar, tenemos que abrir una terminal y entrar como root, lo hacemos como se indica en la figura.



Después de esto, nos cambiamos al directorio apt ubicado en la carpeta etc como se muestra a continuación:

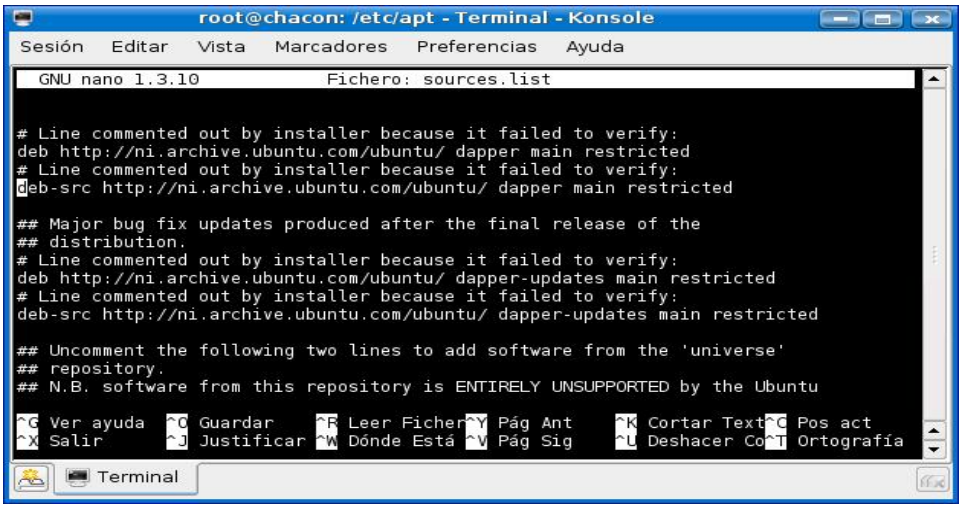
```
root@chacon:/# cd /etc/apt
root@chacon:/etc/apt#
```

Una vez que estemos dentro del directorio apt visualizamos el fichero sources.list, este fichero contiene una lista de repositorios necesarios para las descargas de paquetes.

La visualización se hace así:

```
root@chacon:/etc/apt# nano sources.list
```

El fichero se mostrará como en la figura siguiente:



Por razones de seguridad los repositorios dentro del fichero están comentados, entonces debemos descomentar los siguientes:

deb http://ni.archive.ubuntu.com/ubuntu/ dapper main restricted

deb-src http://ni.archive.ubuntu.com/ubuntu/ dapper main restricted

deb http://ni.archive.ubuntu.com/ubuntu/ dapper-updates main restricted

deb-src http://ni.archive.ubuntu.com/ubuntu/ dapper-updates main restricted

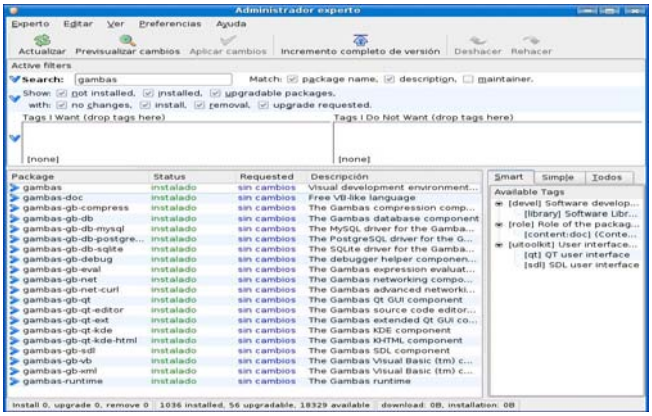
deb http://ni.archive.ubuntu.com/ubuntu/ dapper universe

deb-src http://ni.archive.ubuntu.com/ubuntu/ dapper universe

deb http://security.ubuntu.com/ubuntu/ dapper-security universe

deb-src http://security.ubuntu.com/ubuntu/ dapper-security universe

Luego, guardamos las modificaciones, salimos del fichero y cerramos el terminal. Ahora visualice el administrador de paquetes (Adept) e introduzca la contraseña de inicio de sesión para poder descargar los paquetes necesarios.



Instalación de MySQL

Para descargar los paquetes necesarios de MySql escriba en la pestaña Search del Administrador de paquetes: MySql, luego seleccione los paquetes siguientes:

1. libmysqlclient15off
2. mysql-client
3. mysql-client-5.0
4. mysql-common
5. mysql-server
6. mysql-server-5.0
7. php5-mysql

Instalación del Editor de Diagramas DIA

Para descargar los paquetes necesarios del DIA escriba en la pestaña Search del Administrador de paquetes: *dia*, luego seleccione los paquetes siguientes:

- dia
- dia-common
- dia-gnome
- dia-libs

Una vez descargados los paquetes de Gambas, MySQL y DIA asegúrense que la pestaña Status muestre la opción "*instalado",* de lo contrario habrá ocurrido algún error en la instalación y deberán volver a realizar todo de nuevo.

Nota: Para realizar esta práctica es necesaria la conexión a Internet para poder descargar los paquetes.

PRÁCTICA 1
CONSTRUCCIÓN DE DIAGRAMAS E-R

Objetivo: Aprender a identificar cada uno de los elementos (entidades, relaciones y atributos) que forman un diagrama E-R en sistemas reales sencillos y realizar el diseño del mismo utilizando como herramienta el editor de diagramas DIA.

Introducción

La estructura lógica de una base de datos se puede expresar gráficamente mediante un diagrama E-R. Tal diagrama consta de los siguientes componentes:

Rectángulos: representan un conjunto de entidades.
Rectángulos Dobles: representan entidades débiles.
Elipses: representan atributos.

Rombos: representan relaciones.

Líneas: unen atributos a conjuntos de entidades y conjuntos de entidades a conjuntos de relaciones.

Elipses dobles: representan atributos multivalorados.

Elipses discontinuas: denotan atributos derivados

Líneas dobles: indican participación total de una entidad en un conjunto de relaciones.

Conceptos fundamentales

Entidad: es una cosa u objeto en el mundo real que es distinguible de todos los demás objetos.

Relación: es una asociación entre diferentes entidades.

Atributos: Características que pueden formar parte de una entidad. Pueden ser: simples, compuestos, multivalorados, nulos y derivados. Los más utilizados son los tres primeros.

Atributos Simples: No están divididos en subpartes. Por Ej.: el sexo podría ser un atributo simple de una entidad llamada Persona.

Atributos Compuestos: Están divididos en subpartes. Por ej: el nombre para la misma entidad Persona puede estar dividido en: nombre, primer apellido, segundo apellido, por tanto puede ser considerado como un atributo compuesto.

Atributos multivalorados: son atributos que tienen un conjunto de valores para una entidad específica. Por Ej. el teléfono de una Persona es un atributo multivalorado en el sentido en que una persona puede tener varios números de teléfono.

Entidades débiles: es un conjunto de entidades que no tiene suficientes atributos para formar una llave primaria. Se indica en los diagramas E-R mediante un rectángulo dibujado con una línea doble y la correspondiente relación de identificación mediante un rombo dibujado con línea doble.

Discriminante: conjunto de atributos que en conjunto con la llave primaria de la entidad fuerte forman una entidad diferente dentro del conjunto de entidades débiles.

Llave primaria: es aquella que diferencia una entidad de otra dentro del conjunto de entidades.

Correspondencia de cardinalidades: expresa el número de entidades a las que otra entidad puede estar asociada vía un conjunto relaciones (uno a uno, uno a varios, varios a uno, varios a varios).

Pasos generales para el diseño de un diagrama E-R

- Comprensión del problema.
- Identificación de los conjuntos entidades.
- Identificación de los conjuntos relaciones.

- Diseño o esquematización de los conjuntos entidades y conjuntos relaciones.
- Asignación de cardinalidades.

Con ayuda del editor de diagrama ***DIA*** realizaremos algunos ejercicios sobre diseño de diagramas E–R, poniendo en práctica los conceptos anteriormente mencionados.

Ejercicios

- Una empresa necesita llevar un control de sus proveedores, clientes, productos y ventas. Las características del sistema son:

    - Un proveedor puede atender a varios clientes y realizar muchas ventas o ninguna. Un proveedor tiene un id, nombre, dirección, teléfono y pagina Web.

    - Un cliente puede ser atendido por muchos proveedores y obtener muchos productos o ningunos. Un cliente tiene un id, nombre, dirección pero puede tener varios teléfonos. La dirección se entiende por calle, número y ciudad.

    - Un producto puede ser ofrecido por uno o varios proveedores. Un producto tiene un id, nombre, precio actual, nombre del proveedor. Además se organizan en categorías, una categoría tiene un id, nombre y descripción.

    - La venta tiene un id, fecha, cliente, descuento, monto de la venta, cantidad vendida y monto total por el producto.

Para realizar este ejercicio seguiremos los pasos generales para el diseño de un diagrama E–R.

- **Comprensión del problema.**

El ejercicio consiste en un sistema de ventas en donde se desea llevar un control de sus proveedores, clientes, productos y ventas, para lo cual se necesita identificar el conjunto de identidades y relaciones con sus respectivos atributos y establecer las cardinalidades respectivas.

- **Identificación del conjunto de entidades y sus respectivos atributos.**

  Entidad: *Proveedores.*
  Atributos: *IdProv (key), Nombre, dirección, teléfono y pagWeb.*

  Entidad: *Clientes.*
  Atributos: *IdCliente (key), Nombre, dirección (calle, numero, ciudad), teléfonos.*

  Entidad: *Ventas.*
  Atributos: *IdVentas (key), NombreCliente, descuento, fecha, cantidadVendida, MontoVenta, MontoProd.*

  Entidad: *Producto.*
  Atributos: *IdProd (key), NombreProd, NombreProv, PrecioActual.*

  Entidad débil: *Categoría.*
  Atributos: *IdCateg (weak key), NombreCateg, descripción.*

- **Identificación del conjunto de relaciones.**

  Entidades: *Proveedores y Clientes.*
  Relación: *atienden*.
  Atributos de la relación: *IdProv, IdCliente.*

Entidades: *Proveedores y Producto.*

Relación: *ofrecen.*

Atributos de la relación: *IdProv, IdProd.*

Entidades: *Clientes y Producto.*

Relación: *obtienen.*

Atributos de la relación: *IdProd, IdCliente.*

Entidades: *Proveedores y Ventas.*

Relación: *realiza.*

Atributos de la relación: *IdProv, IdVentas.*

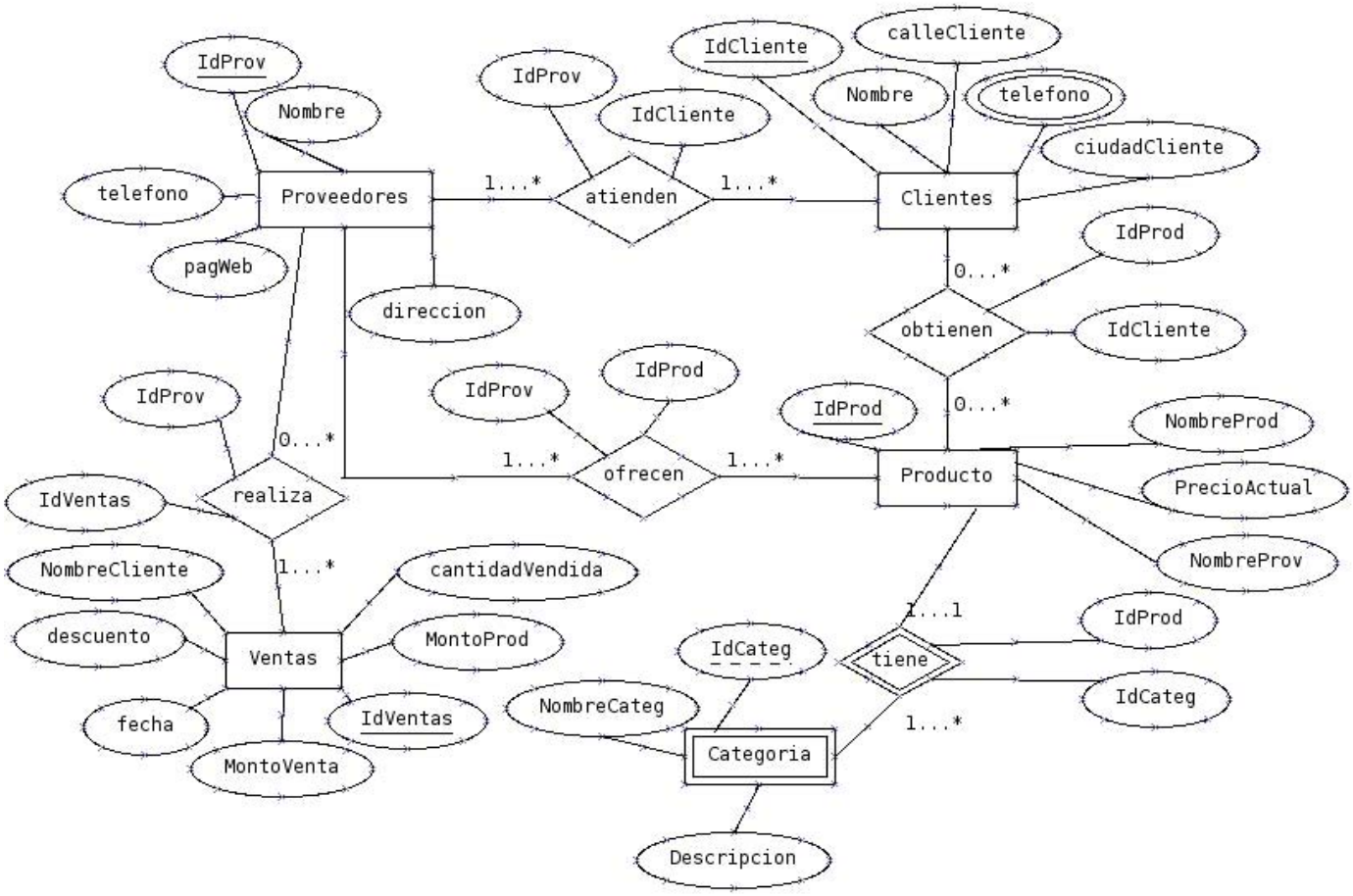
Entidades: *Producto y Categoría.*

Relación: *tiene.*

Atributos de la relación: *IdProd.*

4. Diseño o esquematización de los conjuntos entidades y conjuntos relaciones (Diagrama E – R) y asignación de cardinalidades.

Utilice el editor de diagramas DIA para realizar el diagrama E-R y la asignación de cardinalidades.



2. Una agencia de vuelo desea llevar un control de cada uno de los vuelos realizados en su empresa, las características del sistema son:

- De cada aeropuerto se conoce su código, nombre, ciudad y país. En cada aeropuerto pueden tomar tierra diversos modelos de aviones, el modelo de un avión determina su capacidad, es decir, el número de plazas.

- En cada aeropuerto existe una colección de programas de vuelos que tiene el número de vuelos, línea aérea y días de la semana. Cada programa de vuelo despega de un aeropuerto y aterriza en otro.

- Los números de vuelos son únicos en todo el mundo. En cada aeropuerto hay múltiples aterrizajes y despegues. En todos los aeropuertos tiene algún aterrizaje o algún despegue.

- Cada vuelo pertenece a un programa, para cada vuelo se requiere conocer su fecha, plaza vacía y el modelo del avión.

- **Comprensión del problema.**

El ejercicio consiste en llevar un control de cada uno de los vuelos realizados en una agencia de vuelos, para ello se debe identificar sus respectivas entidades con sus atributos, relaciones y establecer cardinalidades.

- **Identificación de entidades y atributos.**
  - **Entidad débil:** *Despegue.*
  - **Atributos:** *HoraDespegue, lugar.*
  - **Entidad débil:** *Aterriza.*
  - **Atributos:** *Lugar, HoraAterrizaje.*
  - **Entidad:** *Aeropuerto.*
  - **Atributos:** *Código (key), Nombre, Ciudad, País.*
  - **Entidad:** *Aviones.*
  - **Atributos:** *Modelo (key), Capacidad.*
  - **Entidad:** *ProgramasVuelos.*
  - **Atributos:** *NoVuelo (key), LíneaAerea, DiasSemana, AeropuertoOrigen, AeropuertoDeystino.*

- **Entidad:** *Vuelos.*
- **Atributos:** *Fecha, PlazaVacía, ModeloAvión.*

- **Identificación del conjunto de relaciones.**

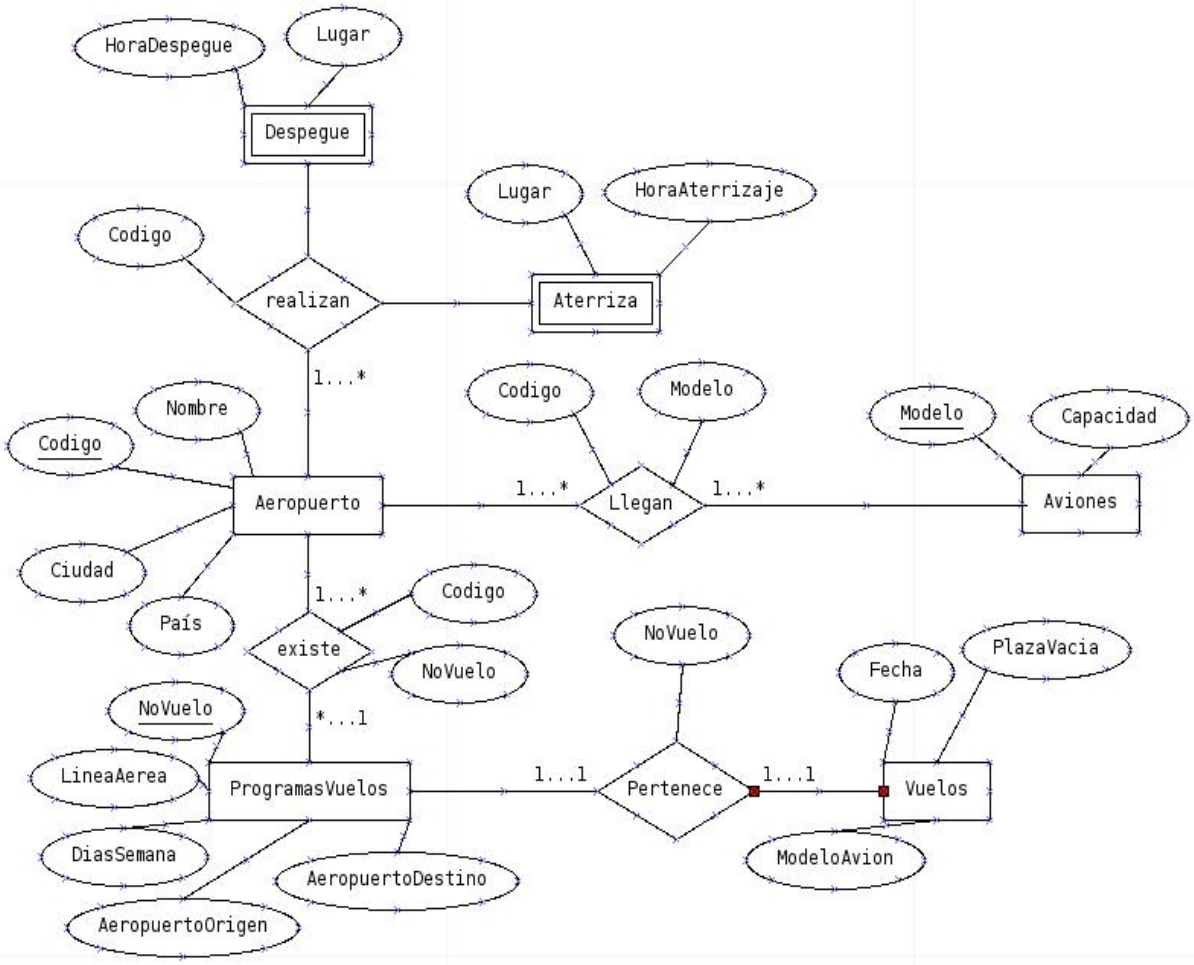
  - **Entidades:** *Despegue, Aterriza, Aeropuerto.*
  - **Relación:** *realizan.*
  - **Atributos de la relación:** *Código.*

  - **Entidades:** *Aeropuerto, Aviones.*
  - **Relación:** *llegan*
  - **Atributos de la relación:** *Código, Modelo.*

  - **Entidades:** *Aeropuerto, ProgramasVuelos.*
  - **Relación:** *existe.*
  - **Atributos de la relación:** *Código, NoVuelo.*

  - **Entidades:** *ProgramasVuelos, Vuelos.*
  - **Relación:** *pertenece.*
  - **Atributos de la relación:** *NoVuelo.*

- **Diseño o esquematización de los conjuntos entidades y conjuntos relaciones (Diagrama E -R) y asignación de cardinalidades.**



Ejercicios Propuestos

Realice el diagrama E-R y asigne las cardinalidades respectivas para cada uno de los siguientes ejercicios.

V. Se desea crear información referente a películas en cartel de salas de cine para un sitio Web.

De cada película se almacena una ficha con el título de distribución, subtítulo original, su género, el idioma, subtítulo, países de origen, el año de producción, la URL del sitio Web de la película, duración (horas, minutos). La clasificación (acta a todo público, mayores de 9 a 15 años, mayores de 18 años) y fecha de estreno.

De cada película interesa conocer la lista de directores y el reparto, es decir, para cada actor que trabaja, el nombre de todos los personajes que interpreta. Además interesa información sobre los directores y actores que trabajan, de ambos se conoce su nombre y su nacionalidad. Se desea conocer la cantidad de películas en las que actuaron y dirigieron.

Para cada función se conoce el día de la semana y la hora de inicio, la sala y la película que exhibe.

De cada sala se sabe el nombre, un número que la identifica y el número de sillas que posee. De cada cine se conoce su nombre, dirección y teléfono.

Además interesa registrar la opinión de las personas, de ésta se conoce el nombre de la persona, la edad, fecha de la opinión, la calificación (obra maestra, buena, muy buena, y mala) a cada opinión se le asigna un número que la identifica respecto a la película que opina.

2. Una base de Datos para una empresa debe tener información acerca de: cliente, artículo y pedido.

Para cada cliente, número de identificación, nombre del cliente, direcciones de envío, saldo, límite de crédito (depende del cliente, pero en ningún caso debe ser superior a los tres millones).

Para cada artículo, número de artículo, fábrica que lo distribuye, existencia de ese artículo, la descripción del artículo.

Cada pedido tiene una cabecera y el cuerpo del pedido, la cabecera está formada por el número del cliente, dirección de envío y fecha de pedido.

El cuerpo del pedido son varias líneas, en cada línea se especifican el número del artículo, pedido y la cantidad.

Se debe almacenar la información de la fábrica, para ellos tenemos el número de fábrica, teléfono y se desea saber cuántos artículos provee la fábrica.

6. **Se desea llevar el control de estadística de los partidos de béisbol del campeonato “Supercampeones”. Se desea:**

   - Información de los equipos que juegan o han jugado en un determinado año.
   - Estadística por juego de los pitcheres y bateadores.
   - Una plantilla de planificación de los juegos de un determinado campeonato.
   - El esquema debe ser capaz de obtener información de los campeonatos ganados por un determinado equipo.
   - Los equipos participantes en cada uno de los juegos de un determinado rango o período de fecha.
   - El cuerpo técnico que pertenece o ha pertenecido a un determinado equipo.

Identifique el conjunto de entidades y conjunto de relaciones para realizar el diseño del diagrama E-R y asigne las correspondientes cardinalidades.

PRÁCTICA 2:

DDL DE MYSQL

Objetivo:

Aprender a crear y manipular una base de datos con sus respectivas tablas usando como sistema gestor **MySQL.**

1. Creación de Bases de datos.

La creación de una Base de Datos en MySQL es una tarea muy simple, al momento de crearla, la base de datos estará vacía, es decir, no contendrá ninguna tabla.

Para familiarizarse con la forma de trabajo de MySQL crearán y manipularán una pequeña Base de Datos, la cual se llamará *"prueba"*.

Para crear una base de datos en MySQL se usa la sentencia CREATE DATABASE:

```
Sintaxis: CREATE DATABASE nombre_de_basededatos;

Por ej:

mysql>CREATE DATABASE prueba;
```

Para averiguar cuántas bases de datos existen en nuestro sistema se usa la sentencia SHOW DATABASES.

Por ej:

```
mysql> SHOW DATABASES;
+--------------------+
|Database            |
+--------------------+
| mysql              |
| prueba             |
| test               |
+----------------- --+
3 rows in set (0.00 sec)
```

Con esto debería aparecer la base de datos prueba que acabamos de crear.

Para seleccionar una base de datos se usa el comando USE, que no es exactamente una sentencia SQL, sino más bien una opción de **MySQL**:

```
Sintaxis: USE nombre_de_basededatos;
mysql> USE prueba;
Database changed
```

2. Creación de Tablas

La sentencia CREATE TABLE sirve para crear tablas.

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

En su forma más simple, la sentencia CREATE TABLE creará una tabla con las columnas que indiquemos. Deberemos indicar el nombre de la tabla y los nombres y tipos de las columnas.

```
Sintaxis: CREATE TABLE nombre_de_tabla (columna1 tipodedato, columna2
tipodedato, columnaN tipodedatoN);

Por ej:

mysql> USE prueba
Database changed
mysql>CREATE TABLE gente (nombre VARCHAR(40), fecha DATE);
Query OK, 0 rows affected (0.53 sec)

mysql>
```

Con esto se ha creado una tabla llamada ***"gente"*** con dos columnas: ***"nombre"*** que puede contener cadenas de hasta 40 caracteres y ***"fecha"*** de tipo fecha.

Para consultar cuántas tablas y qué nombres tienen en una base de datos, se usa la sentencia SHOW TABLES*.*

Por ej:

```
mysql>SHOW TABLES;
+------------------+
| Tables_in_prueba |
+------------------+
| gente            |
+------------------+
1 row in set (0.01 sec)
```

Pero existen muchas más opciones a la hora de definir columnas. Además del tipo y el nombre, se pueden definir valores por defecto, permitir o no que contengan valores nulos, crear una clave primaria, indexar, etc.

Valores nulos

Al definir cada columna se puede decidir si podrá o no contener valores nulos. Deben recordar que, aquellas columnas que son o forman parte de una clave primaria no pueden contener valores nulos.

Verán que, si definen una columna como clave primaria, automáticamente se impide que pueda contener valores nulos, pero este no es el único caso en que puede ser interesante impedir la asignación de valores nulos para una columna.

La opción por defecto es que se permitan valores nulos, ***NULL***, y para que no se permitan, se usa ***NOT NULL***. Por ejemplo:

```
mysql> CREATE TABLE ciudad1 (nombre CHAR(20) NOT NULL, poblacion
INT NULL);
Query OK, 0 rows affected (0.98 sec)
```

Con esto se ha creado la tabla ciudad1, la cual contiene los atributos *nombre* cuyo valor no puede ser un valor nulo (***NOT NULL***), y *población* el cual si permite valores nulos (***NULL***).

Valores por defecto

Para cada columna también se puede definir, opcionalmente, un valor por defecto. El valor por defecto se asignará de forma automática a una columna cuando no se especifique un valor determinado al añadir filas.

Si una columna puede tener un valor nulo, y no se especifica un valor por defecto, se usará NULL como valor por defecto. En el ejemplo anterior, el valor por defecto para *población* es NULL.

Por ejemplo, si queremos que el valor por defecto para *población* sea 5000, podemos crear la tabla como:

```
mysql>CREATE TABLE ciudad2 (nombre CHAR(20) NOT NULL,
    -> poblacion INT NULL DEFAULT 5000);
Query OK, 0 rows affected (0.09 sec)
```

Claves primarias

También se puede definir una clave primaria sobre una columna, usando la palabra clave ***KEY*** o ***PRIMARY KEY***.

Sólo puede existir una clave primaria en cada tabla, y la columna sobre la que se define una clave primaria no puede tener valores *NULL*. Si esto no se especifica de forma explícita, **MySQL** lo hará de forma automática.

Por ejemplo, si queremos crear un índice en la columna *nombre* de la tabla de ciudades, crearemos la tabla así:

```
mysql> CREATE TABLE ciudad3 (nombre CHAR(20) NOT NULL PRIMARY KEY,
    -> poblacion INT NULL DEFAULT 5000);
Query OK, 0 rows affected (0.20 sec)
```

Usar ***NOT NULL PRIMARY KEY*** equivale a ***PRIMARY KEY***, ***NOT NULL KEY*** o sencillamente ***KEY***

Columnas autoincrementadas

En **MySQL** existe 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 autoincrementada 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.

Generalmente, estas columnas se usan como claves primarias 'artificiales'. **MySQL** está optimizado para usar valores enteros como claves primarias, de modo que la combinación de clave primaria, que sea entera y autoincrementada es ideal para usarla como clave primaria artificial:

```
Por ejemplo:
mysql> CREATE TABLE ciudad5 (clave INT AUTO_INCREMENT PRIMARY KEY,
    -> nombre CHAR(20) NOT NULL,
    -> poblacion INT NULL DEFAULT 5000);
Query OK, 0 rows affected (0.11 sec)
```

Ejercicio Propuesto

Introduzca en una BD llamada ***"Escuela"*** las siguientes tablas con las características que se muestran a continuación. Luego compruebe si la base de datos y las tablas han sido creadas correctamente.

Tabla Alumnos	
Atributos	**Tipos de datos**
Carnet_Alumno	varchar(10), not null primary key.
NombreAlumno	varchar(40)
Edad	Int
CursoActual	Int

Tabla Aulas	
Atributos	**Tipos de datos**
NoAula	varchar(6), not null primary key
Capacidad	Int
Tabla Horarios	

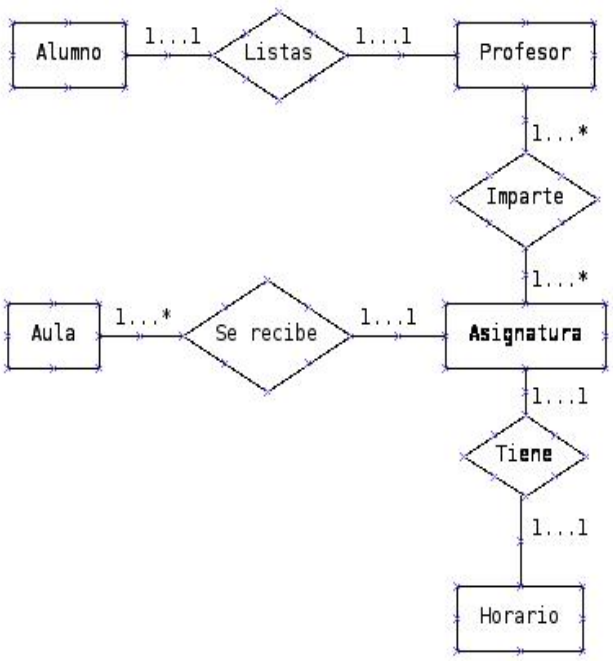
Tabla Profesores	
Atributos	**Tipos de datos**
Clave_Profesor	varchar(4), not null primary key
NombreProfesor	varchar(40)
EstadoCivil	Varchar(10)

Tabla Asignaturas	
Atributos	**Tipos de datos**
Código_Asignatura	bigint, not null primary key
NoAula	varchar(6)
NombreAsignatura	varchar(25)
Curso	int

Atributos	**Tipos de datos**
Código_Asignatura	bigint, not null
Horario_inicio	Time
Horario_fin	Time
Día	Date

Tabla Listas	
Atributos	**Tipos de datos**
Carnet_Alumno	varchar(10), not null
Clave_Profesor	varchar(4), not null

DIAGRAMA ENTIDAD-RELACION



- A traves de consultas SQL realice el esquema relacional

PRÁCTICA 3

DML DE MYSQL

Objetivo:

Aprender a realizar consultas en MySQL. Poniendo en práctica cada una de las sentencias (SELECT, INSERT, UPDATE, DELETE, DROP, ALTER TABLE), cláusulas (ORDER BY, HAVING, WHERE) y funciones de agregación, funciones de fecha/hora y funciones de control de flujo.

1. Selección de datos.

Para extraer datos de una base de datos se usa la sentencia ***SELECT***.

La **sintaxis** de ***SELECT*** es compleja. Una forma más general consiste en la siguiente:

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

Forma incondicional

Consiste en pedir todas las columnas y no especificar condiciones.

```
Sintaxis:
SELECT * FROM nombre_tabla;
```

Limitar las columnas: proyección

Una de las operaciones del álgebra relacional es la **proyección**, que consiste 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. En la sintaxis mostrada, la selección de columnas corresponde con la parte ***"expresion_select"***.

En el ejemplo anterior hemos usado ***'*'***, que quiere decir que se muestran todas las columnas.

Pero se puede usar una lista de columnas, y de ese modo sólo se mostrarán esas columnas:

```
Sintaxis:
SELECT atributo1, atributo2, atributoN FROM nombre_tabla;
```

Las ***expresiones_select*** no se limitan a nombres de columnas de tablas, pueden ser otras expresiones, incluso aunque no correspondan a ninguna tabla:

Las consultas anteriores se refieren sólo a una tabla, pero también es posible hacer consultas usando varias tablas en la misma sentencia ***SELECT***.

Esto nos permite realizar otras dos operaciones del álgebra relacional: ***el producto cartesiano*** y **los inner *join***.

Producto cartesiano

El producto cartesiano de dos tablas son todas las combinaciones de todas las filas de las dos tablas. Usando una sentencia **SELECT** se hace proyectando todos los atributos de ambas tablas. Los nombres de las tablas se indican en la cláusula *FROM* separados con comas:

```
Sintaxis:
SELECT * FROM tabla1, tabla2;
```

2. Inserción de datos.

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 se debe 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.

```
Sintaxis:
INSERT INTO nombre_tabla VALUES (valor1,valor2,valorN);
```

Otra opción consiste en indicar una lista de columnas para las que se van a suministrar valores. A las columnas que no se nombren en esa lista se les asigna el valor por defecto.

```
Sintaxis:
INSERT INTO nombre_tabla (atributo1,atributo2,atributoN) VALUES
(valor1,valor2,valorN);
```

Existe otra sintaxis alternativa, que consiste en indicar el valor para cada columna:

```
Sintaxis: INSERT INTO nombre_tabla
  SET atributo1 = valor, atributo2 = valor;
```

Una vez más, a las columnas para las que no se indiquen valores se les asignarán sus valores por defecto. Esto también se pede hacer usando el valor *DEFAULT*.

Para las sintaxis que lo permiten, se puede observar que cuando se inserta más de una fila en una única sentencia, se obtiene un mensaje desde **MySQL** que indica el número de filas afectadas, el número de filas duplicadas y el número de avisos.

Para que una fila se considere duplicada debe tener el mismo valor que una fila existente para una clave principal o para una clave única. En tablas en las que no exista clave primaria ni índice de clave única no tiene sentido hablar de filas duplicadas. Es más, en esas tablas es perfectamente posible que existan filas con los mismos valores para todas las columnas.

Observe lo que pasa en este ejemplo, existe una tabla llamada *mitabla5* que tiene como atributos *id* y *nombre,* nombre es la clave primaria de la tabla.

```
mysql> INSERT INTO mitabla5 (id, nombre) VALUES
    -> (1, 'Carlos'),
    -> (2, 'Felipe'),
    -> (3, 'Antonio'),
    -> (4, 'Carlos'),
    -> (5, 'Juan');
ERROR 1062 (23000): Duplicate entry 'Carlos' for key 1
```

Si se intenta insertar dos filas con el mismo valor de la clave única se produce un error y la sentencia no se ejecuta. Pero existe una opción que podemos usar para los casos de claves duplicadas: *ON DUPLICATE KEY UPDATE*. En este caso podemos indicar a **MySQL** qué debe hacer si se intenta insertar una fila que ya existe en la tabla.

Las opciones son limitadas: no podemos insertar la nueva fila, sino únicamente modificar la que ya existe.

Sintaxis:

```
INSERT INTO nombre_tabla (atributo1,atributo2,atributoN) VALUES
(valor1,valor2,valorN)
ON DUPLICATE KEY UPDATE atributo = VALUES(atributo);
```

3. 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.

```
Sintaxis: REPLACE INTO nombre_tabla (atributo1, atributo2,atributoN)
          VALUES(valor1,valor2,valorN);
```

Las mismas sintaxis que existen para **INSERT**, están disponibles para **REPLACE**.

Borrado: Sentencia DELETE y DROP

Sentencia Delete.

El borrado se expresa de igual modo que una consulta. Se pueden borrar solo filas completas, es decir, no se pueden borrar valores de atributos concretos.

```
Sintaxis: delete from r where p;
r: representa una relación.
p: representa un predicado.
```

La declaración ***delete*** selecciona primero todas las filas en r para las que p es cierto y a continuación las borra de r. Si se omite la cláusula ***where*** se borran todas las filas de ***r***.

La orden ***delete*** opera solo sobre una relación. Si se desea borrar filas de varias relaciones, se deberá utilizar una orden ***delete*** por cada relación. El predicado de la cláusula ***where*** puede ser tan complicado como otro ***where*** con cláusula ***select***, o tan simple como una cláusula ***where*** vacía.

Sentencia Drop.

Se utiliza para borrar una relación de una base de datos. Dicha orden borra de la base de datos toda la información sobre la relación eliminada.

Sintaxis: *drop table* ***r;***
r: representa la relación a eliminar.

Esta instrucción tiene una repercusión más drástica que la instrucción: ***delete from r****.*

La diferencia en ambas instrucciones es que en la primera (drop) se borran todas las filas de la relación y la relación misma, en cambio, en la segunda (delete) solo se borran todas las filas, es decir, se conserva la relación.

- **Sentencia Alter Table.**

Se utiliza para añadir atributos a una relación existente.

Sintaxis: *alter table* ***r*** *add* ***A D;***
r: representa el nombre de la relación existente.
A: representa el nombre del atributo que se desea añadir.
D: representa el dominio del atributo A.

Además de añadir atributos a una relación se pueden eliminar, utilizando la orden:

alter table r drop A;

donde:

r: representa el nombre de la relación.
A: representa el nombre de un atributo de la relación.

Ejemplo: Crear la tabla persona con atributos numero (int primary key), nombre (varchar (25)), dirección (varchar (25)), e insertar un registro y luego añadir a la tabla persona el atributo teléfono.

```
mysql> create table persona (numero int primary key, nombre varchar(25),
direccion varchar (25));
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> insert into persona
    -> set numero = 1, nombre= "mario", direccion = "abc";
Query OK, 1 row affected (0.03 sec)
```

```
mysql> alter table persona add telefono varchar(8);
Query OK, 1 row affected (0.03 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

6. Actualización: Sentencia Update.

En determinadas situaciones puede ser deseable cambiar un valor dentro de una fila, sin cambiar todos los valores de la misma. Para este tipo de situaciones se utiliza la instrucción ***update***. Al igual que ocurre con ***insert*** y ***delete***, se pueden elegir las filas que van a ser actualizadas mediante una consulta.

Sintaxis: update ***r*** set ***o*** where ***p***
r: representa una relación.
o: operación de actualización.
p: representa un predicado.

7. Funciones.

7.1 Funciones de control de flujo.

Sentencia IF

Sintaxis: IF(*expr1*,*expr2*,*expr3*)

Si *expr1* es TRUE (*expr1* <> 0 and *expr1* <> NULL) entonces IF() retorna *expr2*; de otro modo retorna *expr3*. La función IF() retorna un valor numérico o cadena de caracteres, en función del contexto en que se usa.

Si expr2 o expr3 es explícitamente NULL, el tipo del resultado de la función IF() es el mismo tipo que la expresión no NULL.

Si expr2 y expr3 son cadenas de caracteres, el resultado es sensible a mayúsculas si alguna de las cadenas lo es.

Sentencia IFNULL

Sintaxis: IFNULL (*expr1*, *expr2*)

Si *expr1* no es NULL, IFNULL() retorna *expr1*, de otro modo retorna *expr2*. La función IFNULL() retorna un valor numérico o de cadena de caracteres, en función del contexto en que se usa.

Sentencia NULLIF

Sintaxis: NULLIF (*expr1*, *expr2*)

Retorna NULL si *expr1* = *expr2*, de otro modo retorna *expr1*.

7.2 Funciones de cadenas de caracteres

Sentencia CHAR_LENGTH

Sintaxis: CHAR_LENGTH(*str*)

Retorna la longitud de la cadena de caracteres *str*, medida en caracteres.

Sentencia CONCAT

Sintaxis: CONCAT(*str1*,*str2*,...)

Retorna la cadena resultado de concatenar los argumentos. Retorna NULL si algún argumento es NULL. Puede tener uno o más argumentos.

Sentencia LEFT

Sintaxis: LEFT(*str*,*len*)

Retorna los *len* caracteres empezando por la izquierda de la cadena *str*

Sentencia LOWER

Sintaxis: LOWER(*str*)

Retorna la cadena *str* con todos los caracteres cambiados a minúsculas

Sentencia UPPER

Sintaxis: UPPER(*str*)

Retorna la cadena *str* con todos los caracteres cambiados a mayúsculas.

Sentencia REPEAT

Sintaxis: REPEAT(*str*,*count*)

Retorna una cadena consistente de la cadena *str* repetida *count* veces. Si *count* <= 0, retorna una cadena vacía. Retorna NULL si *str* o *count* son NULL.

Sentencia REVERSE
Sintaxis: REVERSE (*str*)
Retorna la cadena *str* con el orden de los caracteres invertido.

Sentencia RIGHT
Sintaxis: RIGHT(*str*,*len*)
Retorna los *len* caracteres de la derecha de la cadena *str*.

Sentencia STRCMP
Sintaxis: STRCMP (*expr1*, *expr2*)
Retorna 0 si las cadenas son idénticas, -1 si el primer argumento es menor que el segundo según el orden actual, y 1 en cualquier otro caso.

7.3 Funciones de fecha y hora

Sentencia CURDATE
Sintaxis: CURDATE()
Retorna la fecha horaria como valor en formato 'YYYY-MM-DD'

Sentencia CURTIME
Sintaxis: CURTIME()
Retorna la hora actual como valor en formato 'HH:MM:SS'

Sentencia DATE
Sintaxis: DATE (*expr*)
Extrae la parte de fecha de la expresión de fecha o fecha y hora *expr*.

Sentencia DATE_FORMAT
Sintaxis: DATE_FORMAT(*date*,*format*)
Formatea el valor *date* según la cadena *format*.

Sentencia NOW
Sintaxis: NOW()
Retorna la fecha y hora actual como valor en formato 'YYYY-MM-DD HH:MM:SS'

Sentencia TIME
Sintaxis: TIME (*expr*)
Extrae la parte de hora de la expresión hora o fecha/hora *expr*.

Sentencia TIME_FORMAT

Sintaxis: TIME_FORMAT(*time*,*format*)

Se usa como la función DATE_FORMAT() pero la cadena *format* puede contener sólo los especificadores de formato que tratan horas, minutos y segundos.

Ejercicios Resueltos

1. Crear una base de datos llamada animales que contenga las siguientes tablas:

Tabla mascotas con atributos: nombre varchar(20), dueño varchar(20), TipoAnimal varchar(20), sexo char(1), FechaNac date, FechaMuerte date.

Tabla eventos con atributos: nombre varchar(20), fecha date, TipoEvento varchar(15), descripcion varchar(255).

2. Inserción de registros en la tabla mascotas:

```
mysql> insert into mascotas
    -> set nombre ="kaiser", duenyo = "Manuel", sexo ="m", FechaNac = "2004-02-
1
2",especie = "perro";
Query OK, 1 row affected (0.01 sec)
```

```
mysql> insert into mascotas
    -> set nombre ="abelito", duenyo = "Abel", sexo ="m", FechaNac = "2006-12-
24
",especie = "chocoyo";
Query OK, 1 row affected (0.03 sec)
```

```
mysql> insert into mascotas
    -> set nombre ="marita", duenyo = "Karla", sexo ="f", FechaNac = "1989-05-
13
",especie = "gallina";
Query OK, 1 row affected (0.05 sec)
mysql> insert into mascotas
    -> set nombre ="toro", duenyo = "Mario", sexo ="m", FechaNac = "1979-08-
31",
FechaMuerte = "1995-07-29",especie = "caballo";
Query OK, 1 row affected (0.03 sec)
```

```
mysql> insert into mascotas
    -> set nombre ="mañosa", duenyo = "Ernesto", sexo ="f", FechaNac = "1998-
09-
11",especie = "vaca";
Query OK, 1 row affected (0.02 sec)
```

```
mysql> insert into mascotas
    -> set nombre ="Homero", duenyo = "Juan", sexo ="m", FechaNac = "2000-12-
09"
,especie = "gato";
Query OK, 1 row affected (0.03 sec)
```

```
mysql> insert into mascotas
    -> set nombre ="romeo", duenyo = "Julieta", sexo ="m", FechaNac = "2005-03-
1
4",especie = "pajaro";
Query OK, 1 row affected (0.03 sec)
```

```
mysql> insert into mascotas
    -> set nombre ="chispi", duenyo = "Marvin", sexo ="f", FechaNac = "2005-05-
1
0",especie = "perra";
Query OK, 1 row affected (0.03 sec)
```

3. Seleccionar de la tabla mascotas todos los registros en donde TipoAnimal sea perro.

```
mysql> select * from mascotas where especie = "perro";
```

4. Seleccionar aquellas mascotas que han nacido después de la fecha 2000-02-12.

```
mysql> select * from mascotas where FechaNac > "2000-02-12";
```

5. Seleccionar aquellas mascotas donde especie sea chocoyo o pájaro, ordenados por fecha de nacimiento.

```
mysql> select * from mascotas where especie = "chocoyo" or especie = "pajaro"
order by FechaNac;
```

6. Seleccionar el nombre, sexo y TipoAnimal de aquellas mascotas cuyo TipoAnimal sea perro y sexo sea femenino o la especie sea gato y el sexo sea masculino. Use los operadores and y or.

```
mysql> select nombre,sexo,especie from mascotas where (especie = "perro" and
sexo = "f") or (especie ="gato" and sexo = "m");
```

7. Mostrar el nombre, fecha de nacimiento, fecha actual y la edad en años de cada mascota. Use las funciones CURDATE(), YEAR() Y RIGHT()

```
mysql> select nombre,FechaNac,CURDATE() AS FechaActual,
    -> (YEAR (CURDATE())-YEAR(FechaNac))
    -> - (RIGHT(CURDATE(),5) < RIGHT(FechaNac,5))
    -> AS Edad
    -> FROM mascotas;
```

Nota: Para determinar la edad en años de cada mascota, hay que calcular la diferencia entre el año de la fecha actual y el de la fecha de nacimiento y luego restar 1 al resultado si el día y mes actuales son anteriores al día y mes indicados por la fecha de nacimiento.

8. Determinar la edad, a la fecha de muerte, de los animales verificando si el valor de FechaMuerte es Null ordenados por edad.

```
mysql> select Nombre,FechaNac,FechaMuerte,
    -> (year(FechaMuerte)-year(FechaNac)) - Right(FechaMuerte,5)<Right(FechaMue
rte,5))
    -> as edad
    -> from mascotas where FechaMuerte is not null order by edad;
```

9. Encontrar los animales que cumplen años en el mes siguiente del mes actual.

```
mysql> select Nombre,FechaNac from mascotas where month(FechaNac)= 6;
```

Nota: En este caso el ejercicio se hizo en el mes de Mayo por lo cual 6 representa el mes siguiente (Junio)

10. Determinar a que edad fueron llevados los animales al veterinario

```
mysql> select mascotas.Nombre,
    -> (year(Fecha)- year(FechaNac)) - (right(Fecha,5) < right(FechaNac,5)) as
edad,
    -> descripcion
    -> from mascotas, eventos
    -> where mascotas.Nombre=eventos.Nombre and eventos.TipoEvento="veterina
rio";
```

Nota: Para realizar esta consulta se requiere el uso de ambas tablas, ya que la fecha en que cada mascota fue llevada al veterinario está en la tabla eventos, pero para calcular su edad, se necesita su fecha de nacimiento, la cual esta localizada en la tabla mascotas. Observe como se accede a los atributos de cada tabla.

11. Obtener animales machos y hembras que pertenezcan a un mismo tipo de animal.

```
mysql> SELECT p1.Nombre, p1.Sexo, p2.Nombre, p2.Sexo, p1.TipoAnimal
 -> from mascotas as p1, mascotas as p2
 -> where p1.TipoAnimal=p2.TipoAnimal and p1.Sexo="f" and p2.Sexo = "m";
```

Nota: En la consulta anterior se especificaron alias (p1 y p2) para la tabla con el fin de indicar a qué instancia de la tabla pertenece cada columna referenciada.

12. Seleccionar el nombre de aquellas mascotas cuya edad sea mayor o igual a 1 año, en caso contrario concatenar el nombre y la fecha de nacimiento. Verificar si la mascota aun no ha fallecido, en caso positivo no mostrar ese registro.

```
mysql> select if ((year(curdate()) - year(FechaNac)) - (right(curdate(),5) <
right(FechaNac,5)) >= 1, mascotas.Nombre, concat(Nombre, " Edad inferior..",
FechaNac)) from mascotas where FechaMuerte is NULL;
```

13. Seleccionar de la tabla eventos todas las mascotas (concatenando el nombre, el mensaje “ha ido al veterinario” y la fecha que ocurrió el evento) que han ido al veterinario y cuyo nombre tenga una longitud mayor o igual a 5, en caso contrario mostrar el nombre y el mensaje “no ha ido al veterinario”

```
mysql>select
if(strcmp("veterinario",eventos.TipoEvento),concat(eventos.Nombre,"  No ha ido
al veterinario   "),concat(eventos.Nombre,"   ha ido al veterinario el -->
",eventos.Fecha)) from eventos where char_length(eventos.Nombre) >= 5;
```

14. Verificar si la longitud del nombre de la mascota es igual a la longitud del nombre de su dueño, si es así, convertir a mayúsculas el nombre de la mascota y si no invertir el nombre del dueño

```
mysql> select if(char_length(Nombre)=char_length(Duenyo),upper(Nombre),
reverse(Duenyo)) from mascotas;
```

Ejercicios Propuestos

- Insertar los siguientes registros en cada una de las tablas creadas en la Práctica 2 (Tablas: Alumnos, Profesores, Listas, Aulas, Asignaturas y Horarios).

Tabla alumnos

('01-00010-0','Juan Tórrez',16,1), ('01-00012-0','Marvin Baldizón',17,1),
('01-00018-0','Laura Corrales',19,2), ('01-00056-0','Juan Antón',23,5),
('01-00034-0','Luis López',20,3), ('01-00045-0','Luisa Parada',21,3),
('01-00078-0','Martha Rosales',20,3), ('01-00200-0','Miriam Rostran',21,3)
('01-00013-0','Mario López', 15, 2), ('01-00022-0','Maria Laguna', 22, 5)
('01-00023-0','José Parada', 22, 5), ('01-00025-0','Armando Hoyos', 23, 4)
('01-00079-0','Lucia Aráuz', 18, 3).

Tabla profesores

('0002','Aldo Martínez','casado'), ('0006','Álvaro Altamirano','soltero'),
('0008','Rina Aráuz','casada'), ('0010','Johana Hernández','casada'),
('0345','Ana Maria Salgado','casada'), ('0654','Máximo Guido','soltero'),
('0444','Julio Rojas','casado'), ('0987','Ernesto Espinosa','casado')

Tabla Listas

(‘01-00010-0’,’0987’), (‘01-00012-0’,’0654’), (‘01-00034-0’,’0345’), (‘01-00045-0’,‘0010’), (‘01-00078-0’,’0006’), (‘01-00079-0’,’4321’), (‘01-00200-0’,’0002’), (‘01-00056-0’,’0008’), (‘01-00018-0’,’0444’), (‘01-00025-0’,’0444’), (‘01-00022-0’,’0345’), (‘01-00023-0’,’0987’), (’01-00013-0’,’4321’)

Tabla aulas

(‘aula01’,80),(‘aula02’,70), (‘aula03’,52), (‘aula04’,72), (‘aula05’,40),(‘aula06’,60), (‘aula07’,56),(‘aula08’,25)

Tabla asignaturas

(1,’aula08’,’informatica’,1), (2,’aula07’,’Base de Datos’,3), (3,’aula06’,’programacion en c’,1), (4,’aula05’,’Circuitos Lógicos’,3), (5,’aula04’,’ Redes I’,5), (6,’aula03’,’Ingenieria del software’,5), (7,’aula02’,’ Redes II’,5), (8,’aula01’,’Programacion en C++’,4)

Tabla horarios

(1,’07:00:00’,’09:00:00’,’2007-04-16’), (2,’07:00:00’,’09:00:00’,’2007-04-17’), (3,’09:00:00’,’11:00:00’,’2007-04-18’), (4,’11:00:00’,’01:00:00’,’2007-04-18’), (5,’02:00:00’,’04:00:00’,’2007-04-19’), (6,’11:00:00’,’01:00:00’,’2007-04-19’), (7,’07:00:00’,’09:00:00’,’2007-04-19’), (8,’04:00:00’,’06:00:00’,’2007-04-20’)

- Obtener la capacidad del aula donde se imparte la asignatura de Base de Datos.

- Obtener los horarios y días en los que se imparte la asignatura de programación.

- Seleccione de la tabla alumnos el nombre de los alumnos, donde el curso actual sea 3 y la edad sea mayor o igual a 18. A continuación seleccione los registros de la tabla alumnos en donde la edad este entre 15 y 22 años y el nombre empiece con la letra M.

- Seleccionar los nombres (ordenados alfabéticamente) y curso actual de los alumnos cuyo nombre lleve la vocal u.

- Seleccionar aquellas asignaturas que comiencen entre las 7:00 am. y las 9:00 am. ordenadas por la hora de inicio.

- Obtener los alumnos cuyo profesor es Ernesto Espinoza.

- Encontrar los Alumnos cuyo curso actual sea 3 y el nombre del profesor que imparta dicho curso.

- Obtener el número de alumnos cuya edad sea 20 años, luego la edad del mayor de los alumnos.

- Aumentar en un 5% la capacidad de las aulas, excepto aquella donde se imparte la asignatura de informática.

- Añadir a la tabla alumnos el atributo p*rocedencia* y actualizar los registros insertando valores para dicho atributo.

- Seleccionar todos los alumnos (ordenados por edad) cuya procedencia sea Chinandega y que no reciba clase con el profesor Julio Rojas.

- Determinar que día de la semana y en que aula se imparte la asignatura Redes I y II.

- Crear la tabla becas con atributos: *NoCarnet varchar(10)*, *TipoBeca varchar (15), Monto int,* e insertar los registros: (’01-00010-0’,interna’,1200), (’01-00007-0’,’externa’,500), (‘01-00013-0’,’residencia’,1500), (‘01-00022-0’,‘externa’,500), (‘01-00079-0’, ’interna’,1200) A continuación borre los registros donde *TipoBeca* sea externa, luego borre toda la tabla.

- Aumentar las becas internas en un 5%, las externas en un 3% y las de residencia en un 8%.

- Asignar becas externas con monto de 500 a todos los estudiantes no becados de León.

- Concatenar el aula y la asignatura que se imparte en dicha aula, si la capacidad del aula es mayor a 50, en caso contrario, concatenar el aula y el mensaje “Su capacidad es inferior a 50”.

- Seleccionar las aulas en las que se está impartiendo clases en este momento y la asignatura impartida, en caso que en ninguna se imparta clase mostrar el aula con un mensaje que diga “no hay clases en este momento”.

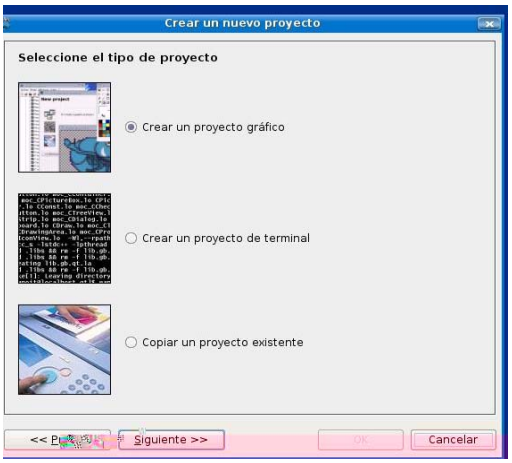
PRACTICA 4.
COMPONENTES DEL ENTORNO DE DESARROLLO GAMBAS

Objetivo: conocer cada una de las herramientas y facilidades que proporciona el entorno, desarrollar habilidades en el diseño de interfaces y aprender a realizar aplicaciones sencillas con ayuda de dichas herramientas.

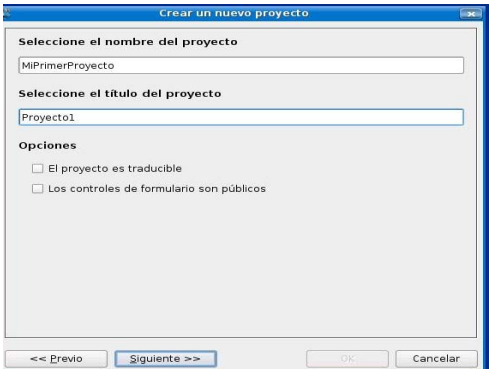
Como iniciar un proyecto en Gambas

Inicio del asistente de creación de proyectos de Gambas

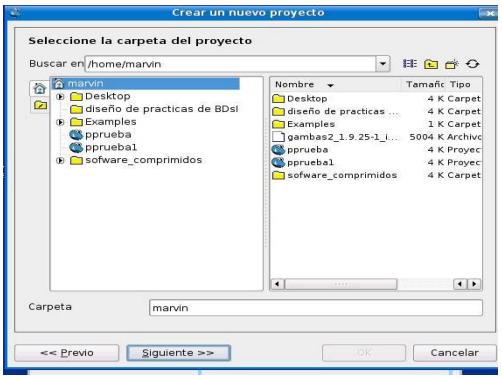
3. Seleccionar el tipo del proyecto**.**



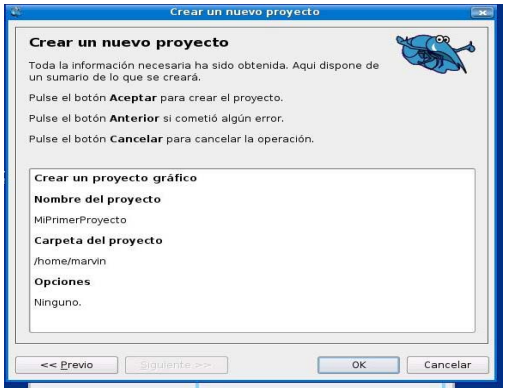
- Dar un nombre al proyecto.



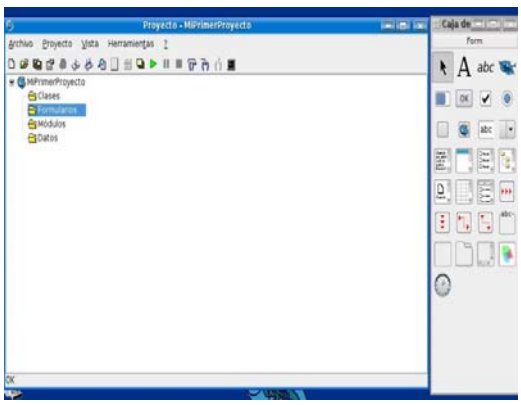
- Seleccionar la ubicación del proyecto



- Finalizar proyecto

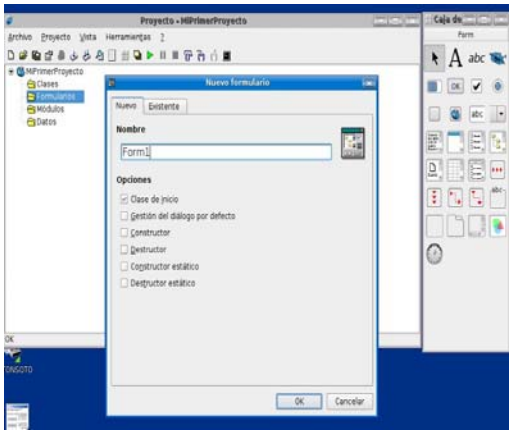


- Herramientas de desarrollo.

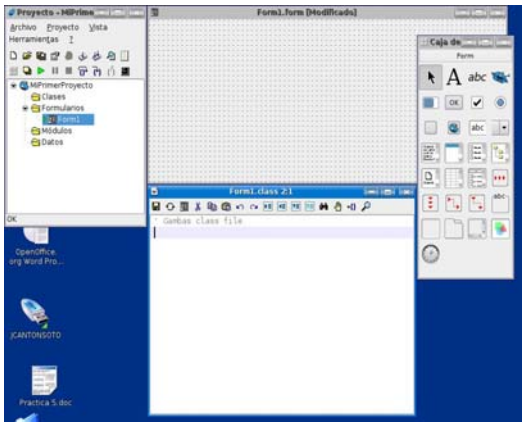


8. Creación de un nuevo formulario

Para la creación de un nuevo formulario, ubíquese en la carpeta Formularios y a continuación de clic derecho sobre ella, escoja la opción Nuevo y luego Formulario..., aparecerá el asistente para crear formulario, allí, ingrese un nombre para su formulario y escoja una de las opciones que se muestran debajo, una vez que esté seguro de lo hecho de clic en OK.



IDE ó Entorno de Desarrollo.



Interfaz gráfica de usuario

Select: Se utiliza para seleccionar cualquier componente.

Label: Se utiliza para crear una etiqueta sobre un formulario.

PictureBox: Muestra una imagen jpg, png.

Button: Crea un botón con el cual desencadenamos eventos que accionan una determinada rutina.

CheckBox: Control para opciones incluyentes (Casilla de verificación).

RadioButton: Control para opciones excluyentes (botón de radio).

TextBox: Pone sobre el formulario una caja de texto.

ComboBox: Muestra valores en una lista desplegable.

TextArea: Pone sobre el formulario una caja de texto en la cual se permite escribir varias líneas con la diferencia del TextBox que solo permite escribir una.

ListBox: Muestra valores en una lista.

Frame: Grupo de opciones como botones de radio.

Nota: Para más información sobre las demás herramientas consulte la ayuda de Gambas.

Creación de formularios cuya fuente de datos es una sola tabla.

En esta parte vamos a crear los formularios correspondientes a las tablas que fueron creadas en la practica 2.

Formulario FrmAlumno

1. Crear un formulario llamado FrmAlumnos.

- Agregar cuatro etiquetas con sus respectivas cajas de texto.

3 Para insertar las etiquetas con sus cajas de texto, ubiquese en la caja de herramientas y de clic en *Label* (Etiqueta) y dibujela sobre el formulario, para agregarle un nombre solo ubiquese en las propiedades de ésta y vaya a *Text*, de esta forma hará para la inserción de sus respectivas cajas de texto (*TextBox)*.

Formato para el formulario FrmAlumnos

Herramientas	***Propiedades***
Label1	**Text:** *No. Carnet,* **Font:** *TSCu_Comic,14* **Border:** *Sunken*
Label2	**Text:** *Nombre del Alumno,* **Font:** *TSCu_Comic,12* **Border:** *Sunken*
Label3	**Text:** *Edad,* **Font:** *TSCu_Comic,14* **Border:** *Sunken*
Label4	**Text:** *Curso Actual,* Font: *TSCu_Comic,12* **Border:** *Sunken*
TextBox	**Name:** *txtNoCarnet* **Group:** *grupoEC* **Text:** **ToolTip:** *Solo es permitido 10 digitos separados por guiones* **Font:** *Century Schoolbook L,10,Bold* **Foreground:** *&HFF0000& RGB(255,0,0)* **MaxLength:** *10*
TextBox	**Name:** *txtNombreAlumno* **Text:** **Font:** *Century Schoolbook L,10,Bold* **Foreground:** *&HFF0000& RGB(255,0,0)*
TextBox	**Name:** *txtEdad* **Group:** *grupoEC* **Text:** **Font:** *Century Schoolbook L,10,Bold* **Foreground:** *&HFF0000& RGB(255,0,0* **MaxLength:** *2)*
TextBox	**Name:** *txtCursoActual* **Group:** *grupoEC* **Text:** **Font:** *Century Schoolbook L,10,Bold* **Foreground:** *&HFF0000& RGB(255,0,0)* **MaxLength:** *1*
Form	**Name:** *FrmAlumnos* **Text:** *Inserción de datos de alumnos*

Resultado del formulario FrmAlumnos

Formulario FrmProfesores

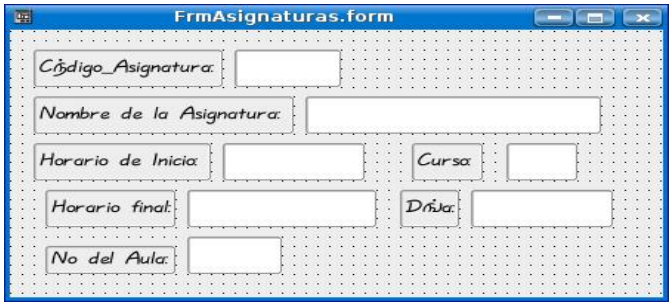
De igual manera que hizo con el formulario FrmAlumnos haga con el formulario FrmProfesores, agregue las siguientes herramientas con sus respectivas propiedades.

Herramientas	***Propiedades***
Label1	**Text:** Código: **Font:** TSCu_Comic,14 **Border:** Sunken
Label2	**Text:** Nombre: **Font:** TSCu_Comic,14 **Border:** Sunken
Label3	**Text:** Estado Civil: **Font:** TSCu_Comic,14 **Border:** Sunken
TextBox	**Name:** txtCodigo **Text:** **ToolTip:** Inserte 4 dígitos **Font:** Century Schoolbook L,10,Bold **Foreground:** &HFF0000& RGB(255,0,0) **MaxLength:** *4*
TextBox	**Name:** txtEstadoCivil **Group:** *grupoCajas* **Text:** **Font:** Century Schoolbook L,10,Bold **Foreground:** &HFF0000& RGB(255,0,0)
Form	**Name:** FrmProfesores **Text:** Inserción de datos de profesores

Resultado del formulario FrmProfesores

Formulario FrmAsignaturas

Herramientas	Propiedades
Label1	**Text:** Código_Asignatura: **Font:** TSCu_Comic,14 **Border:** Sunken
Label2	**Text:** Nombre de la Asignatura: **Font:** TSCu_Comic,14 **Border:** Sunken
Label3	**Text:** No del Aula: **Font:** TSCu_Comic,14 **Border:** Sunken
Label4	**Text:** Curso **Font:** TSCu_Comic,14 **Border:** Sunken
TextBox	**Name:** txtCodigoAsig **Text:** **Font:** Century Schoolbook L,10,Bold **Foreground:** &HFF0000& RGB(255,0,0)
TextBox	**Name:** txtNombreAsig **Text:** **Font:** Century Schoolbook L,10,Bold **Foreground:** &HFF0000& RGB(255,0,0)
TextBox	**Name:** txtNoAula **Text:** **Font:** Century Schoolbook L,10,Bold **Foreground:** &HFF0000& RGB(255,0,0)
TextBox	**Name:** txtCurso **Text:** **Font:** Century Schoolbook L,10,Bold **Foreground:** &HFF0000& RGB(255,0,0)
TextBox	**Name:** txtHInicio **Text:** **Font:** Century Schoolbook L,10,Bold **Foreground:** &HFF0000& RGB(255,0,0)
TextBox	**Name:** txtHFinal **Text:** **Font:** Century Schoolbook L,10,Bold **Foreground:** &HFF0000& RGB(255,0,0)
Form	**Name:** FrmAsignaturas **Text:** Visualizacion de registros de asignaturas



Resultado del formulario FrmAsignaturas

PRACTICA 4.1
CONTINUACIÓN DE LA PRÁCTICA ANTERIOR**.**

Creación de aplicaciones sencillas utilizando el entorno de desarrollo Gambas.

- Vamos a crear nuestro primer proyecto llamado “holamundo”, para el cual realice los siguientes pasos:

1) Arrancar el entorno de desarrollo Gambas.
2) Crear un Nuevo Proyecto.
3) Seleccionar la opción Crear un Proyecto Gráfico.
4) Dar nombre al proyecto, en este caso *holamundo.*
5) En la carpeta **formularios** crear uno nuevo (Nuevo-Formulario-OK).
6) Agregar un botón al formulario (Caja de Herramientas/clic en button/arrastrarlo al formulario). En la propiedad *Name* del Botón cambiar el texto a **Btn*Pulsame*** y en *Text* escribir **Pulsame.**

7) Dar doble clic sobre el botón y aparecerá la siguiente instrucción.

```
PUBLIC SUB BtnPulsame_Click()
END
```

8) Escribir dentro de la instrucción anterior lo siguiente:

```
Message.info (“Hola mundo”, “OK”)
```

9) Ubicar el ratón en la pestaña ***Proyecto*** de la barra de menú y escoger la opción ***compilar todo***.

10) Ejecutar el proyecto dando clic en el icono , ubicado en la barra de botones del proyecto

Una vez ejecutado el proyecto deberá aparecer la siguiente figura:

Elaborar el siguiente proyecto llamado Temperatura, el cual consiste en la conversión de grados Celsius a grados Farenheit y viceversa.

4 Crear un proyecto llamado *Temperatura.*

5 Crear un nuevo formulario llamado *Conversion de temperaturas* y agregar los siguientes componentes: 2 etiquetas con sus cajas de texto y dos botones.

Componente	***Propiedades***
Label1	Text: *Grados Celsius*
TextBox	Name: *txtGCelsius* Text:
Label2	Text: *Grados Farenheit*
TextBox	Name: *txtGFar* Text:
Button1	Name: *BtnConvertir* Text: *Convertir*
Button2	Name: *BtnLimpiarCajas* Text: *Limpiar Cajas*

6 Dar doble clic sobre el botón *Convertir* y en el interior de ***PUBLIC SUB BtnConvertir_Click()*** introducir el código siguiente:

```
PUBLIC SUB BtnConvertir_Click()

DIM far AS Float        // se declaran las variables far y celsius de
tipo float
 DIM celsius AS Float

        IF txtGFar.Text <> 0  THEN  // si la caja de texto txtGFar.text
es != 0 entonces
             far = CFloat(txtGFar.Text)    // capturar el contenido de la
caja de texto txtGFar en la variable far
             txtGCelsius.Text = CString((far + 32) * (5/9))    //
almacenar el resultado de la conversion en la
                              siguiente caja de texto txtGCelsius
       END IF                 // fin del If
```

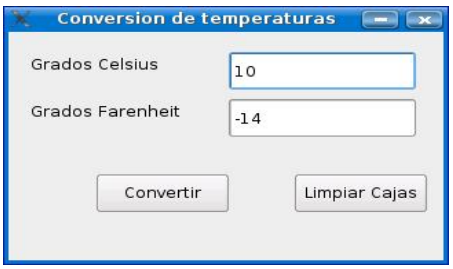
```
// La misma lógica se utiliza cuando la caja de texto txtGCelsius es != 0

IF txtGCelsius.Text <> 0 THEN
           celsius = CFloat(txtGCelsius.Text)
           txtGFar.Text = String((celsius * (9/5)) - 32)
      END IF
END
```

7 Dar doble clic sobre el botón *LimpiarCajas* y ponga el siguiente código como lo hizo en el inciso anterior.

```
PUBLIC SUB BtnLimpiar_Click()

    txtGCelsius.Text = ""
    txtGFar.Text = ""

END
```

Compilar y ejecutar el proyecto.

8 Resultado de una ejecución del proyecto:



Nota: La clase Cfloat convierte una expresión a un número de coma flotante, su sintaxis es la siguiente:

Float = Cfloat (Expresion)

La clase Cstring convierte una expresion en una Cadena de caracteres, su respectiva sintaxis es la siguiente:

String = CString (Expresion)

- Elaborar un proyecto llamado “pgallopinto” el cual consiste en una comideria donde el cliente decide el tipo de comida y de bebida que desea, y según la opcion elegida sera el total a pagar. Con este proyecto se pondra en practica los controles: RadioButton, CheckBox, TextBox y Frame.

- Crear un nuevo proyecto
- Crear un Formulario de nombre Frmgallopinto y agregue los siguientes componentes:

Las características de la interfaz son las siguientes:

Herramienta	**Propiedades**
Frame	Text: *Comidas*
Frame	Text: *Bebidas*
CheckBox	Name: *bisteck* Text: *Bisteck C$ 30*
CheckBox	Name: *carneasada* Text: *Carne Asada C$ 20*
CheckBox	Name: *pollofrito* Text: *Pollo Frito C$ 25*
CheckBox	Name: *gallopinto* Text: *Gallo Pinto C$ 10*
RadioButton	Name: *cocacola* Text: *Coca Cola C$ 10*
RadioButton	Name: *refresco* Text: *Refresco Natural C$ 8*
RadioButton	Name: *jugo* Text: *Jugo C$ 15*
RadioButton	Name: *agua* Text: *Agua C$ 7*
TextBox	Name: *txtpreciocomida* Text: Read Only: *true* Alignment: *center*
TextBox	Name: *txtpreciobebida* Text: Read Only: *true* Alignment: *center*
TextBox	Name: *txttotal* Text: Read Only: *true* Alignment: *center*
Label1	Text: *Precio Comida C$*
Label2	Text: *Precio Bebida C$*
Label3	Text: *Total a Pagar C$*

Nota: El control Frame sirve para agrupar los CheckBox y los RadioButton

3. Dar funcionalidad a los checkbox y radiobutton

3.1 Dar doble clic sobre el checkbox Bisteck e insertar el siguiente código

```
   PUBLIC SUB bisteck_Click()

         IF bisteck.Value = TRUE THEN
              txtpreciocomida.Text = Str(Val(txtpreciocomida.Text) + 30)

              ELSE
                  txtpreciocomida.Text = Str(Val(txtpreciocomida.Text) - 30)

         END IF

         txttotal.Text = Str(Val(txtpreciocomida.Text) +
Val(txtpreciobebida.Text))

   END
```

Nota: Completar el código para los demás CheckBox siguiendo la misma lógica anterior

3.2 Dar doble clic sobre el radiobutton Coca Cola e insertar el siguiente código

```
PUBLIC SUB cocacola_Click()
       txtpreciobebida.Text = "10"
       txttotal.Text = Str(Val(txtpreciocomida.Text) +
Val(txtpreciobebida.Text))
END
```

Nota: Completar el código para los demás RadioButton siguiendo la misma lógica anterior

3.3 Al cargar el formulario inicializar a cero las cajas de texto: precio comida y preciobebida para esto, de doble clic sobre el formulario e inserte el siguiente código:

```
PUBLIC SUB Form_Open()
 txtpreciocomida.Text = Val("0")
txtpreciobebida.Text = Val("0")
 END
```

4. Al finalizar la codificación, compile y ejecute el proyecto

5. Una corrida del proyecto se puede ver en la siguiente figura:

- Elaborar un proyecto llamado CombosListas y agregar el formulario FrmCombosListas, el objetivo del proyecto consiste en la manipulación de combos, listas y botones de opción.

Las características de la interfaz del formulario serán:

Herramienta	Propiedades
Button	Name: *BtnInsCombo* Text: *Insertar un elemento del Combo en la lista*
Button	Name: *BtnBorrarLista* Text: *Borrar de la lista*
Button	Name: *BtnInsUsr* Text: *Insertar un usuario*
ComboBox	Name: *ComboBox1*
ListBox	Name: *ListBox1*

En la función que carga un formulario (*PUBLIC SUB Form_Open()),* declare un arreglo de elementos de tamaño aleatorio, el cual aparecerá en el ComboBox, para ello de doble clic sobre el formulario y escriba el siguiente código:

```
PUBLIC SUB Form_Open()

   DIM Cadena[6] AS String              //arreglo de 6 elementos de tipo
cadena
   DIM i AS Integer
   DIM elemaleatorio AS Integer
    DIM MAXELEM AS Integer
      Cadena[0] = "Marvin"
      Cadena[1] = "Juan"
      Cadena[2] = "Allan"
      Cadena[3] = "Abel"
     Cadena[4] = "Martha"
     Cadena[5] = "Mario"

     Randomize  // iniciar la semilla para generar numeros aleatorios
```

```
 FOR i = 1 TO 100
       elemaleatorio = Int(Rnd(0,5))   //elemaleatorio almacena la
posicion de                                        //uno de los elementos
del arreglo                                             //generada de forma
aleatoria
       ComboBox1.Add(Cadena[elemaleatorio])  // añadir al combo el elemento
                                                  //correspondiente a la
posición                                                      //generada por
elemaleatorio
    NEXT

END
```

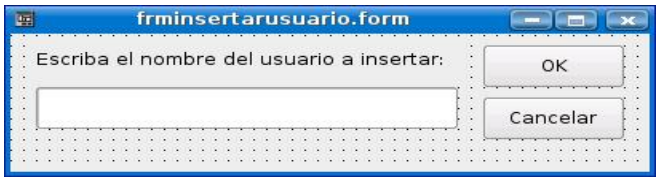
Luego, añada al botón *Insertar un elemento del Combo en la lista*, el siguiente código.

```
PUBLIC SUB BtnInsCombo_Click()

    IF combobox1.Index < 0 THEN
      Message.Info ("No hay ningun elemento seleccionado")

       ELSE
         Listbox1.Add(combobox1.Text)      // insertar un elemento del
combo en                                          //la lista
    END IF

END
```

Añada al botón *Borrar de la lista* el código que se muestra a continuación.

```
PUBLIC SUB BtnBorrarLista_Click()

  DIM var AS Integer
  DIM aux AS Integer

     var = ListBox1.Count                // var almacena la cantidad de
                                         // elementos de la lista

       IF var = 0 THEN                    //Si no hay ningun elemento en la
                                          //      lista
         Message.Info ("No hay elementos en la lista","OK")  // mostrar
                                                               mensaje

         ELSE

          aux = ListBox1.Index         //aux almacena la posicion del
                                            elemento seleccionado

           IF aux < 0 THEN              //si no hay ningun elemento
                                        //  seleccionado
             Message.Info ("No hay ningun elemento seleccionado","OK")
            ELSE                                // de lo contrario

           ListBox1.Remove(aux)  //borrar de la lista el elemento
                                 seleccionado

           END IF
      END IF
  END
```

En el botón *Insertar un usuario* se debe visualizar un cuadro de diálogo que pida el usuario a insertar, para esto se tiene que crear otro formulario llamado *frminsertarusuario*, este formulario debe quedar de la siguiente manera:



Un Label con ***Name:*** Label1 y ***Text:*** Escriba el nombre del usuario a insertar.
Un TextBox con ***Name:*** txtnombreusuario y la propiedad Text la dejamos en blanco.
Un Button con ***Name:*** btnok y ***Text:*** OK.
Un Button con ***Name:*** btnCancel y ***Text:*** Cancelar.

Una vez hecho esto, para poder enlazar este cuadrito de dialogo con el formulario principal (FrmCombosListas), dé un clic derecho sobre el formulario frminsertarusuario y en sus propiedades ponga a ***True*** la propiedad ***Persistent***, Luego, en el botón OK añada el siguiente código:

```
PUBLIC SUB btnok_Click()

     IF txtnombreusuario.Text  = " "  THEN
         Message.Info ("Debe escribir un nombre","OK")
     ELSE
          var = txtnombreusuario.Text  //var es una variable de tipo String
                                       //declarada de forma global
          ME.Close()                             //cerrar el Dialogo
Modal
     END IF

END
```

Y en el botón Cancelar

```
PUBLIC SUB btncancel_Click()
      frminsertarusuario.Close    //cerrar el formulario frminsertarusuario
END
```

Ahora dentro del boton "Insertar un usuario" del formulario principal introduzca lo siguiente:

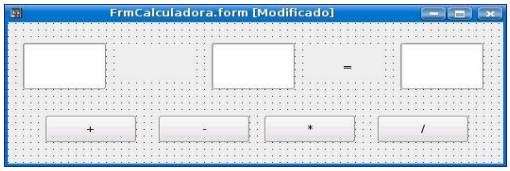
```
PUBLIC SUB BtnInsUsr_Click()

     frminsertarusuario.ShowDialog()  //la funcion ShowDialog() muestra un
                                           //dialogo modal

     IF frminsertarusuario.var <> " " THEN      //Si se ha introducido un
nombre
       ListBox1.Add(frminsertarusuario.var)       //Agregarlo a la lista
     END IF

     frminsertarusuario.Delete()        //destruir el dialogo modal

END
```

Ejercicios propuestos

- Realizar una pequeña aplicación que consiste en una calculadora elemental que permite hacer las cuatro operaciones aritméticas. Agregar un nuevo formulario llamado FrmCalculadora, los controles a utilizar se muestran en el siguiente gráfico**:**

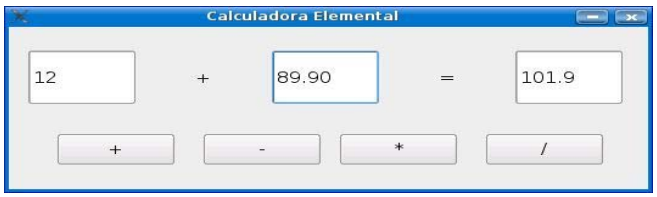
Herramienta	***Propiedades***
TextBox	Name: *txtOper1* Text:
TextBox	Name: *txtOper2* Text:
TextBox	Name: *txtResult* Text:
Label	Name: *lblOp* Text: Alignment: *Center*
Label	Name: *lblIgual* Text: = Alignment: *Center*
Button	Name: *cmdSuma* Text: +
Button	Name: *cmdResta* Text: -
Button	Name: *cmdProd* Text: *
Button	Name: *cmdDiv* Text: /

La interfaz en tiempo de diseño se verá de la siguiente manera**:**



El usuario debe de introducir los operandos en las dos primeras cajas de texto, al dar clic en cualquiera de los botones debe aparecer entre ambas cajas de texto la operación correspondiente a dicho botón y visualizar el resultado de la operación en la última caja de texto. Realice la codificación de cada uno de los botones.

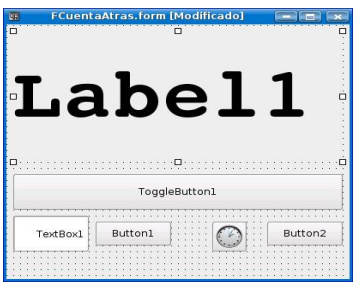
Una ejecución de la aplicación podría ser como se muestra en la figura:



- Realizar una aplicación, la cual consiste en un cronómetro con una cuenta atrás en pantalla. Se podrá cambiar el tiempo del cronometro, detenerlo y arrancarlo cuando queramos. Los controles que se utilizaran para la aplicación son: un Label, un TextBox, un Timer, dos Button y ToggleButton. Modifique únicamente las propiedades que se especifican.

Herramientas	***Propiedades***
Label	Name: lblContador Font: Courier Bold,72
ToggleButton	Name: tglFuncionando
TextBox	Name: txtsegundos Alignment: Right
Timer	Name: clkMiReloj
Button	Name: cmdPonerSegundos
Button	Name: cmdSalir

El formulario en tiempo de diseño se vera de la siguiente manera:



Para dar funcionalidad a la aplicación comience con la codificación de la misma:

Para ello antes de la función *PUBLIC SUB Form_Open()* que arranca un formulario defina las siguientes variables y procedimientos:

```
CONST fSegundoPorDefecto AS Float = 120.0  // Constante que contiene el número
de                                          //segundos por defecto de la
cuenta atrás.
 fSegundos AS Float     // Variable que contendrá los segundos de la cuenta
atrás.

PRIVATE SUB VerValores() y PRIVATE SUB VerActivarDesactivar()
```

En la función que carga el formulario llamar a las funciones VerValores y VerActivarDesactivar, poner en el Button1 el texto *Reiniciar* y en el Button2 el texto Salir y en la caja de texto visualizar los segundos de la cuenta atrás.

La Función *VerValores* presentará en la etiqueta el tiempo en minutos, segundos y milisegundos del cronometro (en este caso debe aparecer por defecto 2:00.0 al ejecutar por primera vez la aplicación) y la función *VerActivarDesactivar* debe presentar en el ToggleButton el texto Arrancar o Detener según sea el caso.

ToggleButton: Al dar clic en este botón se debe activar el timer para iniciar el cronometro y llamar a la función VerActivarDesactivar.

Botón Salir: Al dar clic el usuario en este botón se deberá salir de la aplicación

Botón Reiniciar: Al dar clic en este botón se debe capturar el contenido de la caja de texto de tal manera que vuelva a empezar la cuenta atrás del cronometro y llamar a la función *VerValores.*

Objeto Timer: Establece el intervalo para recibir eventos del reloj, en nuestro caso la precisión del cronómetro para recibir un evento será cada décima de segundo. Para ello en las propiedades del objeto Timer cambiar el valor de Delay que contiene 1000ms a 100.

En la definición de la función del Objeto timer se deberá calcular los segundos de la cuenta atrás, es decir disminuirlos en una décima y llamar a la función VerValores.

Si el cronometro finaliza la cuenta atrás (llega a cero) sin detenerlo, el togglebutton debe cambiar el texto de Detener a Arrancar.

Al ejecutar por primera vez la aplicación, deberá aparecer la siguiente interfaz:

3. Elaborar un proyecto llamado Países, el cual consiste en un menú con una lista de países (Estados Unidos, Canadá, México, Alemania, Panamá, Rusia, España y Venezuela), por defecto debe aparecer la hora nacional (capturar la del sistema) en una caja de texto. Al seleccionar un país del menú se mostrara su correspondiente hora en otra caja de texto.

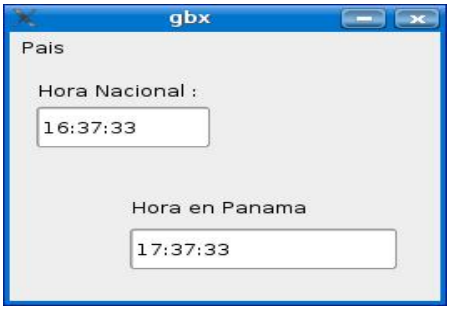
Para calcular la hora de cada país hay que sumarle a la hora nacional la diferencia horaria según el país elegido. Ambos tiempos deben ser dinámicos, es decir, actualizarse según la hora del sistema.

Controles a utilizar: Menu, Dos Label, Dos timer, Dos TextBox

El formulario en tiempo de diseño se vera de la siguiente manera:

Al seleccionar un país del menú, en la etiqueta “Hora en” debe aparecer el nombre del País elegido, por Ej.: si seleccionamos del menú el país Panamá, deberá aparecer: Hora en Panamá

Una ejecución del programa seria:



En este caso se selecciono el país Panamá, como se puede observar aparece tanto la hora nacional como la hora correspondiente a Panamá según la diferencia horaria, que en este caso la diferencia es de una hora. No olvidar que ambos tiempos deben cambiar continuamente según la hora del sistema

PRACTICA 5.
DISEÑO DE APLICACIONES CON ACCESO A BASE DE DATOS

Objetivo: Con esta práctica se pretende que el alumno aprenda a establecer la conexión y a trabajar con una base de datos MySQL desde el entorno de desarrollo Gambas.

1. Asignar privilegios de usuario para manipular una Base de Datos.

Antes de poder hacer la conexión de nuestra Base de Datos con el gestor MySQL tenemos que crearnos un usuario que tenga privilegios en la base de datos que queremos acceder:

Sintaxis:

```
GRANT ALL ON *.* TO usr IDENTIFIED BY 'xxxxx'
```

GRANT: crea un usuario y le concede los privilegios especificados a continuación.
ALL: concede todos los privilegios (select, create, update, delete, drop, insert).
ON *.*: concedemos privilegios de nivel de tabla para todas las tablas y bases de datos.
TO *usr* ***IDENTIFIED BY*** *'xxxxx':* es el usuario a quien se le concede los privilegios con su respectiva contraseña.

Un ejemplo de lo anterior sería:

```
Grant all on *.* to marvin identified by 'marvin123'
```

2. Como conectarnos con una base de Datos MySQL

Antes de empezar con la conexión hay que asegurarse que el componente de acceso a bases de datos esta activo, para ello estando dentro del proyecto que contiene los formularios con los que vamos a trabajar, ubicarse en la pestaña proyecto y seleccionar la opcion propiedades, luego seleccionar la pestaña componentes, aparecerá una lista de todos los componentes del entorno en la cual debemos activar el componente gb.db para poder establecer la conexión con cualquier base de datos.

2.1 Crear una variable publica de tipo Connection

```
Ej.: Public con As Connection
```

2.2 Crear una función en la cual especificaremos el tipo y nombre de la base de datos a la cual queremos conectarnos, el host o dirección IP de nuestro ordenador, el usuario y contraseña con la cual iniciamos sesión en MySQL. Todos los parámetros mencionados se obtienen a través de la variable publica (**con**) creada anteriormente.

```
PUBLIC SUB Conectar()
con.Type = Tipo de base de datos (mysql,postgree,sqlite, etc)
con.Host = Host o dirección IP del ordenador
con.Name = Nombre de la base de datos
con.Login = usuario
con.Password = contraseña
con.Open     //abrir la base de datos
END
```

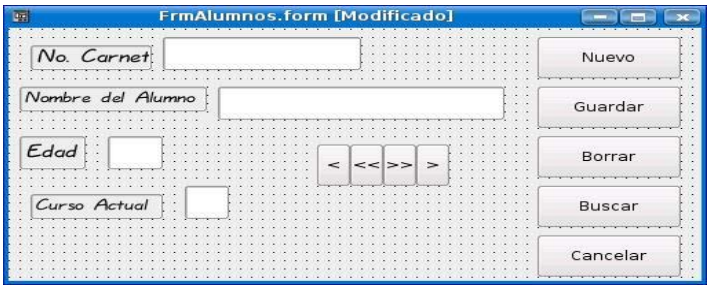
Para poner en práctica los pasos anteriormente explicados vamos a trabajar con la base de datos “escuela”, las tablas creadas en la práctica 2 y los formularios creados en la práctica 4. Para ello realice los siguientes ejercicios paso a paso para comprender mejor cómo se trabaja con una base de datos MySQL.

Formulario FrmAlumnos:

Agregar los siguientes controles al formulario FrmAlumnos.

Herramientas	*Propiedades*
Button	Name: *btnguardar* Text: *Guardar*
Button	Name: *btnborrar* Text: *Borrar*
Button	Name: *btnbuscar* Text: *Buscar*
Button	Name: *btncancelar* Text: *Cancelar*
Button	Name: *btnprimero* Text: < ToolTip: *primero*
Button	Name: *btnanterior* Text: << ToolTip: *anterior*
Button	Name: *btnsiguiente* Text: >> ToolTip: *siguiente*
Button	Name: *btnultimo* Text: >> ToolTip: *último*

El formulario en vista de diseño se muestra a continuación:



Establecer la conexión con la Base de Datos "escuela".

Antes de establecer la conexión con la base de datos asegurarse que el componente de acceso a bases de datos este activo.

Luego crear un modulo llamado "funciones" e incluir el siguiente código, el cual contiene todas las funciones necesarias para el manejo de la base de datos y para dar funcionalidad a los controles del formulario FrmAlumno.

Modulo Funciones

```
' Gambas module file
PUBLIC con AS NEW Connection  //con es una variable publica tipo Connection con
                              //la cual realizamos la conexión con la base de
                              //datos
PUBLIC rs AS Result    //rs es un objeto tipo Result que se utiliza para
                       //almacenar el resultado de una consulta MySQL.

PUBLIC SUB Conectar()  //Función que realiza la conexión con la base de datos
  con.Type = "mysql"
  con.Host = "localhost"
  con.Name = "escuela"
  con.Login = "marvin"
  con.Password = "marvin123"
  con.Open

END
```

//Función para limpiar las cajas de texto cuando sea necesario

```
PUBLIC SUB LimpiarCajas(c1 AS TextBox, c2 AS TextBox, c3 AS TextBox, c4 AS
TextBox)
  c1.Text = ""
  c2.Text = ""
  c3.Text = ""
  c4.Text = ""
END
```

//Función para bloquear y desbloquear las cajas de texto según sea el caso.

```
PUBLIC SUB BloquearDesbloquearCajas(c1 AS TextBox, c2 AS TextBox, c3 AS
TextBox, c4 AS TextBox, b AS Boolean)

  IF b = TRUE THEN
    'bloquear las cajas
      c1.Enabled = FALSE
      c2.Enabled = FALSE
      c3.Enabled = FALSE
      c4.Enabled = FALSE

     ELSE
     'desbloquear las cajas
        c1.Enabled = TRUE
        c2.Enabled = TRUE
        c3.Enabled = TRUE
        c4.Enabled = TRUE
  END IF
END
```

//Función para bloquear y desbloquear los botones según sea el caso

```
PUBLIC SUB BloquearDesbloquearBoton(boton AS Button, v AS Boolean)

    IF v = TRUE THEN
      'bloquear el boton
      boton.Enabled = FALSE

      ELSE
       'desbloquear el boton
       boton.Enabled = TRUE

    END IF
END
```

//Función para guardar en la base de datos el registro contenido en las cajas de texto

```
PUBLIC SUB GuardarenBD(c1 AS TextBox, c2 AS TextBox, c3 AS TextBox, c4 AS
TextBox)
DIM v1 AS Integer
DIM v2 AS Integer

  v1 = Val(c3.Text)
  v2 = Val(c4.Text)

//sentencia MySQL que realiza la inserción de los datos contenidos en las cajas
de texto
  con.Exec("insert into alumnos values (&1,&2,&3,&4)", c1.Text, c2.Text, v1,
v2)

END
```

//Función para borrar el registro actual que se encuentre en las cajas de texto

```
PUBLIC SUB BorrardeDB(c1 AS TextBox, c2 AS TextBox, c3 AS TextBox, c4 AS
TextBox)

   DIM v1 AS Integer
   DIM v2 AS Integer

   v1 = Val(c3.Text)
   v2 = Val(c4.Text)

////sentencia MySQL que realiza el borrado del registro actual mostrado en las
//cajas de texto

   con.Exec("delete from alumnos where carnet_alumno = &1",c1.Text)

END
```

3. Codificación de los controles del formulario FrmAlumnos

- Al arrancar el formulario tiene que mostrarse el primer registro de la tabla “alumnos” en los controles respectivos. Para ello dé doble clic sobre el formulario y edite el siguiente código.

```
PUBLIC SUB Form_Open()
'conectarnos con la BD escuela
funciones.Conectar()       //llamamos a las funciones a traves del nombre del
modulo

'bloquear las cajas de texto
funciones.BloquearDesbloquearCajas(txtNoCarnet,txtNombreAlumno,txtEdad,txtCurso
Actual, TRUE)

'bloquear el boton guardar y cancelar
funciones.BloquearDesbloquearBoton(btnguardar, TRUE)
funciones.BloquearDesbloquearBoton(btncancelar, TRUE)

'guardar todos los registros de la tabla alumnos en la variable de 'tipo Result
funciones.rs = funciones.con.Exec("select * from alumnos")

'Visualizar en las cajas de texto el primer registro de la tabla
txtNoCarnet.Text = funciones.rs["carnet_alumno"]
txtNombreAlumno.Text = funciones.rs["nombre_alumno"]
txtEdad.Text = funciones.rs["edad"]
txtCursoActual.Text = funciones.rs["curso_actual"]

END
```

El **botón Guardar** tiene la funcionalidad de almacenar los datos insertados en las cajas de texto a la tabla alumnos de la Base de Datos “escuela”. Dé doble clic sobre el botón Guardar y edite el siguiente código.

```
PUBLIC SUB btnguardar_Click()

  'verificar si hay datos en las cajas de texto
    IF txtNoCarnet.Text = "" OR txtNombreAlumno.Text = "" OR txtEdad.Text = ""
OR txtCursoActual.Text = "" THEN

      Message.Error("Introducir datos", "Ok")

        ELSE
         'bloquear las cajas
 funciones.BloquearDesbloquearCajas(txtNoCarnet,txtNombreAlumno,txtEdad,
txtCursoActual, TRUE)

    'desbloquear el boton nuevo,borrar y botones de navegación
     funciones.BloquearDesbloquearBoton(btnnuevo, FALSE)
     funciones.BloquearDesbloquearBoton(btnborrar, FALSE)
     funciones.BloquearDesbloquearBoton(btnprimero, FALSE)
     funciones.BloquearDesbloquearBoton(btnsiguiente, FALSE)
     funciones.BloquearDesbloquearBoton(btnanterior,FALSE)
     funciones.BloquearDesbloquearBoton(btnultimo, FALSE)
     funciones.BloquearDesbloquearBoton(btnbuscar, FALSE)
```

```
'bloquear el boton guardar y cancelar
funciones.BloquearDesbloquearBoton(btnguardar,TRUE)
funciones.BloquearDesbloquearBoton(btncancelar, TRUE)

     'guardar el registro en la BD
    funciones.GuardarenBD(txtNoCarnet,txtNombreAlumno,txtEdad, txtCursoActual)

 'seleccionar los registros incluyendo el nuevo, almacenados en el objeto result
    funciones.rs = funciones.con.Exec("select * from alumnos")

    'limpiar cajas
    funciones.LimpiarCajas(txtNoCarnet,txtNombreAlumno,txtEdad, txtCursoActual)

    'mostrar el ultimo registro
     btnultimo_Click()

     Message.Info("Registro guardado")

   END IF
END
```

- El **botón Borrar**, borrará el registro actual, pero antes de hacerlo debe preguntarse si deseamos borrar dicho registro de la Base de Datos a través de un mensaje.

```
PUBLIC SUB btnborrar_Click()

   IF  Message.Question("Esta  seguro  que  desea  borrar  el  registro
actual?", "Aceptar", "Cancelar") = 1 THEN

 //llamada a la función Borrar
     funciones.BorrardeDB(txtNoCarnet)

     funciones.rs = funciones.con.Exec("select * from alumnos")

     'colocar el cursor en el primer registro
     btnprimero_Click()

     Message.Info("Registro Borrado")

  END IF

END
```

- El **botón Buscar**, abrirá una caja de diálogo modal llamada frmBuscarAlumno en la cual deberemos introducir el número de carnet del registro a buscar, al dar clic en aceptar de la caja de diálogo se deberá mostrar el registro correspondiente en las cajas de texto respectivas. Dé doble clic sobre el botón Buscar y edite el siguiente código.

Nota: Consideramos no necesaria la explicación de cómo trabajar con las cajas de diálogo puesto que ya fue explicado en prácticas anteriores por lo cual damos por hecho que el estudiante ya sabe como hacerlo.

```
PUBLIC SUB btnbuscar_Click()
DIM i AS Integer
DIM nreg AS Integer

frmBuscar.ShowDialog()

'seleccionar todos los registros de la tabla alumnos
funciones.rs = funciones.con.Exec("select * from alumnos")
```

```
'contar el numero de registros en la tabla alumnos
nreg = funciones.rs.Count

FOR i=0 TO  nreg

IF frmbuscar.var = funciones.rs["carnet_alumno"] THEN


  txtNoCarnet.Text = funciones.rs["carnet_alumno"]
  txtNombreAlumno.Text = funciones.rs["nombre_alumno"]
  txtEdad.Text = funciones.rs["edad"]
  txtCursoActual.Text = funciones.rs["curso_actual"]

ELSE IF i = nreg - 1 THEN
  Message.Info("Registro no encontrado","Ok")

  ELSE
     funciones.rs.MoveNext

END IF
NEXT
END
```

codificación de la caja de dialogo frmBuscarAlumno

```
' Gambas class file
PUBLIC var AS String 'variable global
```

```
PUBLIC SUB btnaceptar_Click()

  IF txtncarnetb.Text = "" THEN
      Message.Info("Debe escribir un numero de carnet","OK")

      ELSE
        var = CString(txtncarnetb.Text)

        ME.Close()
  END IF

END
```

```
PUBLIC SUB Form_Show()

  txtncarnetb.Clear
END
```

```
PUBLIC SUB txtncarnetb_KeyPress()

 IF (NOT ((key.Code >= Asc("0") AND key.Code <= Asc("9")) OR key.Code =
key.Delete OR key.code = key.Left OR key.Code = key.Right OR key.Code =
key.BackSpace)) THEN

    STOP EVENT

  END IF

END
```

```
PUBLIC SUB btncancelar_Click()
 frmBuscarAlumno.Close
END
```

El **botón Nuevo**, permitirá ingresar un nuevo registro en la base de datos.

```
PUBLIC SUB btnnuevo_Click()

'limpiar cajas
funciones.LimpiarCajas(txtNocarnet,txtNombreAlumno,txtEdad,
txtCursoActual)

'desbloquear las cajas
funciones.BloquearDesbloquearCajas(txtNocarnet,txtNombreAlumno,txtEdad,txtCurso
Actual, FALSE)

'establecer el foco a la caja de texto txtNoCarnet
txtNoCarnet.SetFocus()

'desbloquear el boton guardar y cancelar
funciones.BloquearDesbloquearBoton(btnguardar, FALSE)
funciones.BloquearDesbloquearBoton(btncancelar, FALSE)

'bloquear el boton nuevo y el boton borrar
funciones.BloquearDesbloquearBoton(btnnuevo, TRUE)
funciones.BloquearDesbloquearBoton(btnborrar, TRUE)

END
```

- El **Botón Cancelar**, permitirá cancelar alguna acción que en ese momento no se necesita.

```
PUBLIC SUB btncancelar_Click()

'bloquear las cajas
funciones.BloquearDesbloquearCajas(txtNoCarnet,    txtNombreAlumno,    txtEdad,
txtCursoActual, TRUE)

'bloquear el boton cancelar y guardar
funciones.BloquearDesbloquearBoton(btncancelar, TRUE)
funciones.BloquearDesbloquearBoton(btnguardar, TRUE)

'desbloquear los botones: nuevo, buscar, borrar y botones de 'navegacion
funciones.BloquearDesbloquearBoton(btnnuevo, FALSE)
funciones.BloquearDesbloquearBoton(btnborrar, FALSE)
funciones.BloquearDesbloquearBoton(btnbuscar, FALSE)
```

```
funciones.BloquearDesbloquearBoton(btnprimero, FALSE)
funciones.BloquearDesbloquearBoton(btnsiguiente, FALSE)
funciones.BloquearDesbloquearBoton(btnanterior, FALSE)
funciones.BloquearDesbloquearBoton(btnultimo, FALSE)

'colocar el cursor en el 1er registro
btnprimero_Click()

END
```

- Los **botones Primero, Siguiente, Anterior y Ultimo** permitirán navegar por la base de datos para mostrar el registro correspondiente.

Botón Primero:

```
PUBLIC SUB btnprimero_Click()
funciones.rs.MoveFirst
'actualizar las cajas
actualizarcajas()
END
```

Botón Siguiente:

```
PUBLIC SUB btnsiguiente_Click()

IF funciones.rs.Index + 1 <> funciones.rs.Count THEN
'mover el cursor al siguiente registro de su posición actual
funciones.rs.MoveNext
'actualizar las cajas

actualizarcajas()
END IF

END
```

Botón Anterior:

```
PUBLIC SUB btnanterior_Click()

IF funciones.rs.Index <> 0 THEN
'mover el cursor al registro anterior de su posicion actual
funciones.rs.MovePrevious
'actualizar las cajas
actualizarcajas()
END IF

END
```

Botón Último:

```
PUBLIC SUB btnultimo_Click()
'mover el cursor al ultimo registro
funciones.rs.MoveLast
'actualizar las cajas
actualizarcajas()
END
```

- La **función actualizarcajas()** muestra en las cajas de texto el registro correspondiente al cual apunta el objeto Result.

```
PUBLIC SUB actualizarcajas()

txtNoCarnet.Text = funciones.rs["carnet_alumno"]
txtNombreAlumno.Text = funciones.rs["nombre_alumno"]
txtEdad.Text = funciones.rs["edad"]
txtCursoActual.Text = funciones.rs["curso_actual"]

END
```

El evento **KeyPress**

La función **grupoEC_KeyPress()** será activada cada vez que el usuario introduzca letras en vez de números en las cajas de texto No. Carnet, Edad, Curso Actual, también permitirá suprimir (delete) algún número o borrarlo (backspace), desplazarse con las direccionales izquierda (left) o derecha (right) o la facilidad de cambiarse de cajas de texto con tab.

El código de la función es de esta forma:

```
PUBLIC SUB grupoEC_KeyPress()

IF ( NOT ((key.Code >= Asc(“0”) AND key.Code <= Asc(“9”)) OR key.Code =
key.Delete OR key.Code = key.BackSpace or key.Code = key.Left OR key.Code =
key.Right OR key.Code = key.Tab OR )) THEN
  STOP EVENT
END IF

END
```

El NoCarnet permitirá introducir 10 números separados por guiones, el estudiante deberá codificar esto dentro de la función.

Para la caja de texto del nombre del estudiante se debe hacer casi lo mismo con lo que se hizo en la función grupoEC_KeyPress() se le dejará al estudiante hacer la codificación.

Ayuda: dé un clic derecho sobre la caja de texto txtNombreAlumno y escoja el evento KeyPress.

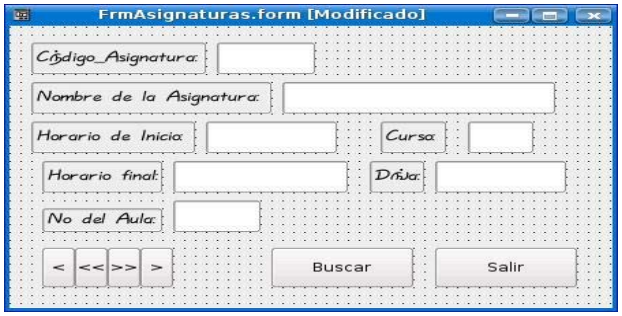
```
PUBLIC SUB txtNombreAlumno_KeyPress()
  ‘Escriba aquí el código para que no permita números
END
```

Formulario FrmAsignaturas:

1. Agregar los siguientes controles al formulario FrmAsignaturas.

Herramientas	***Propiedades***
Button	Name: btnbuscar Text: Buscar
Button	Name: btnsalir Text: Salir
Button	*Name: btnprimero* Text: < ToolTip: *primero*
Button	Name: *btnanterior* Text: < ToolTip: *anterior*
Button	Name: *btnsiguiente* Text: >> ToolTip: *siguiente*
Button	Name: *btnultimo* Text: >> ToolTip: *último*

El formulario en vista de diseño debe verse de la siguiente manera:



2. Codificación de los controles del formulario FrmAsignaturas.

- Al arrancar el formulario tiene que mostrarse el primer registro de la tabla “asignaturas” en los controles respectivos. Para ello dé doble clic sobre el formulario y edite el siguiente código.

```
PUBLIC SUB Form_Open()

funciones.Conectar()
'bloquear las cajas de texto
funciones.BloquearDesbloquearCajas(txtCodigoAsig,txtNombreAsig,
txtNoAula, txtCurso, TRUE)
'guardar todos los registros de asignaturas en la variable de tipo 'Result
funciones.rs = funciones.con.Exec("select * from asignaturas")
actualizarcajas()
END
```

- El **Botón Buscar:** Realiza la búsqueda de las asignaturas por su código, para lo cual se abre una caja de diálogo modal llamada frmBuscaAsig en la cual se debe introducir dicho código. Edite el siguiente código para ver la funcionalidad del botón.

```
PUBLIC SUB btnbuscar_Click()
DIM i AS Integer
DIM regtotal AS Integer

'abrir la caja de dialogo modal frmBuscarAsig
frmBuscarAsig.ShowDialog()
'seleccionar todos los registros de la tabla asignaturas y horarios
funciones.rs = funciones.con.Exec("select * from asignaturas,horarios")

‘contar el numero de registros en la tabla
regtotal = funciones.rs.Count

  FOR i=0 TO  regtotal

   IF frmBuscarAsig.var = funciones.rs["codigo_asignatura"] THEN

        txtCodigoAsig.Text = funciones.rs["codigo_asignatura"]
        txtNombreAsig.Text = funciones.rs["nombre_asignatura"]
        txtHInicio.Text = funciones.rs["horario_inicio"]
        txtCurso.Text = funciones.rs["curso"]
        txtHFinal.Text = funciones.rs["horario_fin"]
        txtHDia.Text = funciones.rs["dia"]
        txtNoAula.Text = funciones.rs["naula"]

     ELSE IF i = regtotal - 1 THEN
      Message.Info("Registro no encontrado","Ok")

       ELSE
        funciones.rs.MoveNext
    END IF
   NEXT
END
```

Codificación de la caja de dialogo frmBuscarAsig

```
' Gambas class file
PUBLIC var AS String
```

```
PUBLIC SUB btnOK_Click()

  IF txtCodAsig.Text = "" THEN
      Message.Info("Debe escribir un Codigo de asignatura","OK")

      ELSE
        var = CString(txtCodAsig.Text)

        ME.Close()
  END IF

END
```

```
PUBLIC SUB btnCancelar_Click()

  frmBuscarAsig.Close
END
PUBLIC SUB Form_Show()
  txtCodAsig.Clear
END
```

```
PUBLIC SUB txtCodAsig_KeyPress()

  IF (NOT ((key.Code >= Asc("0") AND key.Code <= Asc("9")) OR key.Code =
key.Delete OR key.code = key.Left OR key.Code = key.Right OR key.Code =
key.BackSpace)) THEN

    STOP EVENT

  END IF

END
```

Botón Salir: simplemente nos permitirá salir del formulario en caso de no realizar alguna acción.

```
PUBLIC SUB btnSalir_Click()
ME.close
END
```

La **función actualizarcajas()** muestra en las cajas de texto el registro correspondiente al cual apunta el objeto Result.

```
PUBLIC SUB actualizarcajas()
  txtCodigoAsig.Text = funciones.rs["codigo_asignatura"]
  txtNombreAsig.Text = funciones.rs["nombre_asignatura"]
  txtHInicio.Text = funciones.rs["horario_inicio"]
  txtCurso.Text = funciones.rs["curso"]
  txtHFinal.Text = funciones.rs["horario_fin"]
  txtHDia.Text = funciones.rs["dia"]
  txtNoAula.Text = funciones.rs["naula"]
END
```

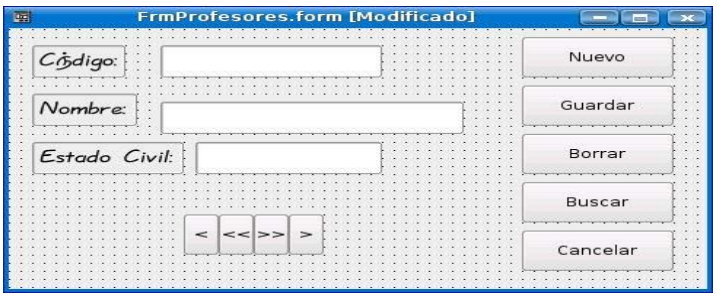
Los **botones primero, anterior, siguiente y último** realizan la misma función que en el formulario FrmAlumno por lo tanto se deja al estudiante realizar la codificación.

Formulario FrmProfesores:

1. Agregar los siguientes controles al formulario FrmProfesores**.**

Herramientas	***Propiedades***
Button	Name: *btnguardar* Text: *Guardar*
Button	Name: *btnborrar* Text: *Borrar*
Button	Name: *btnbuscar* Text: *Buscar*
Button	Name: *btncancelar* Text: *Cancelar*
Button	Name: *btnprimero* Text: < ToolTip: *primero*
Button	Name: *btnanterior* Text: << ToolTip: *anterior*
Button	Name: *btnsiguiente* Text: >> ToolTip: *siguiente*
Button	Name: *btnultimo* Text: > ToolTip: *último*

El formulario en vista de diseño debe verse de la siguiente manera:



2. Codificación de los controles del formulario FrmProfesores.

Como puede verse los controles son los mismos que en el formulario FrmAlumnos, por tanto la funcionalidad es la misma. Se deja al estudiante realizar la respectiva codificación, verifique si es necesario algún cambio y realícelo de tal manera que logre la funcionalidad de los controles, use el evento KeyPress en caso necesario.

Para finalizar, crear un formulario principal llamado **FrmPrincipal** el cual servirá para visualizar los formularios anteriores a través de botones de pulsación. Los controles del formulario principal se muestran a continuación.

Herramienta	***Propiedades***
Button	**Name:** *btnAlumnos* **Text:** *Alumnos* **Font:** *TSCu_Comic,20* **Background:** *&H5555FF& (RGB 85,85,255)*
Button	**Name:** *btnAsignaturas* **Text:** *Asignaturas* **Font:** *TSCu_Comic,20* **Background:** *&H5555FF& (RGB 85,85,255)*
Button	**Name:** *btnProfesores* **Text:** *Profesores* **Font:** *TSCu_Comic,20* **Background:** *&H5555FF& (RGB 85,85,255)*
Form	**Name:** *frmPrincipal* **Text:** *Bienvenidos a la Base de Datos escuela* **Background:** *&H000000&(RGB 0,0,0)*

El formulario en vista diseño se verá de la siguiente manera:

Codificación de los botones de pulsación

Botón Alumnos:

```
PUBLIC SUB btnAlumnos_Click()
    FrmAlumnos.Show  'abrir el formulario FrmAlumnos
END
```

Botón Asignaturas:

```
PUBLIC SUB btnAsignaturas_Click()
    FrmAsignaturas.Show  'abrir el formulario FrmAsignaturas
END
```

Botón Profesores:

```
PUBLIC SUB btnProfesores_Click()
   FrmProfesores.Show  'abrir al formulario FrmProfesores
END
```

PRACTICA 5.1

DISEÑO DE APLICACIONES CON ACCESO A BASE DE DATOS
CONTINUACIÓN DE LA PRÁCTICA ANTERIOR

Objetivo: Que el alumno ponga en práctica cada uno de los conocimientos adquiridos en la práctica anterior sobre la conexión y manipulación de una base de datos MySQL desde el entorno de programación Gambas.

Ejercicio Propuesto

Realizar una aplicación, la cual consiste en un Rent a Car para alquiler de vehículos. El objetivo de la aplicación es llevar un control del alquiler y devolución de los vehículos del Rent a Car, así mismo mantener la base de datos actualizada de tal manera que facilite la búsqueda y el registro de cada vehículo. Por cada vehículo alquilado se deberá tener una orden de alquiler y su correspondiente factura y además se deberá disminuir y aumentar el stock de vehículos por alquiler o devolución de los mismos.

Crear la base de datos llamada Rent_A_Car
Crear las siguientes tablas con sus respectivos atributos:

Tabla Automaster

Atributo	*tipo de dato*
Matricula	varchar(7) primary key
Marca	varchar(30)
Modelo	varchar(30)
Color	varchar(10)
Estado	varchar(20)
Precioxdia	Double
Multaxdia	Int

Tabla Cliente

Atributo	*tipo de dato*
Ncedula	varchar(16) primary key
Nombre	varchar(50)

Tabla Vehiculo

Atributo	*Tipo de dato*
Ncedula	varchar(16)
Matricula	varchar(7) primary key
Marca	Varchar(30)
Modelo	varchar(30)
Fechasalida	varchar(10)
fechaentrada	varchar(10)
Diasalquiler	Int
precioalquiler	Double
Deposito	Double

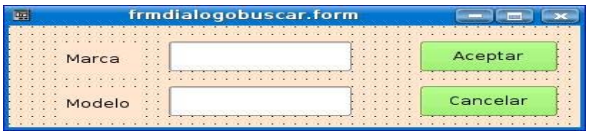
- Diseñar los siguientes formularios

frmAutomaster

El objetivo de este formulario es visualizar las características de cada vehículo solicitado por los clientes para su alquiler.

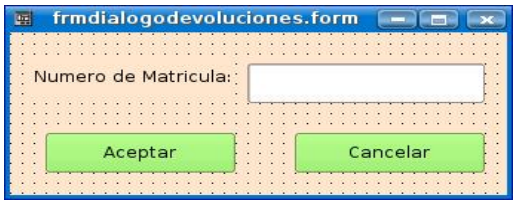
- Utiliza como fuente de datos la tabla Automaster, es decir las etiquetas contendrán los mismos nombres que los atributos de la tabla.
- Las cajas de texto asociadas a cada etiqueta tendrán el mismo nombre con el prefijo txt, es decir si una etiqueta se llama matricula, su caja de texto se llamará txtmatricula.
- Tendrá 4 botones llamados: btnbuscar, btnguardar, btnregistrardevoluciones, btnactualizarinventario y btnsalir.

Botón Buscar: buscará en la base de datos el vehículo requerido por el cliente para su alquiler y mostrará las características del mismo en las respectivas cajas de texto del formulario. La búsqueda deberá hacerse a través de una caja de diálogo en la cual se debe insertar la marca y modelo del vehículo. La caja de diálogo se llamará frmdialogobuscar y se verá como en la siguiente figura:



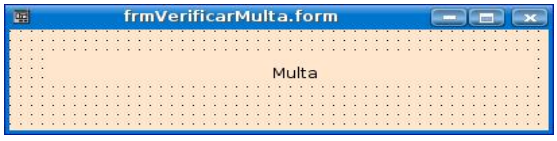
Botón Guardar: permitirá registrar en la base de datos (tabla Automaster) uno o tantos nuevos vehículos adquiera el Rent a Car.

Botón RegistrarDevoluciones: permitira actualizar la tabla Automaster, es decir debe cambiar el valor del atributo “estado” de **alquilado** a **disponible** del vehiculo que esta siendo devuelto en ese momento, dicha devolución debe hacerse a través de una caja de diálogo llamada frmdialogodevoluciones en la cual se debe introducir el número de matrícula del carro (si el número de matrícula introducido no corresponde a ningún vehículo del Rent a car mostrar un mensaje informando dicha situación), como se muestra en la siguiente figura:



Además se debe borrar dicho registro de la tabla Vehículo de tal manera que ya no cuente como alquilado.

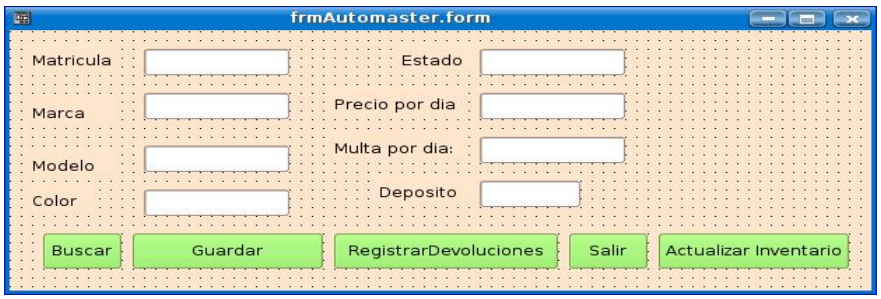
El Rent a Car tiene una política de multas por cada día de retraso de la fecha establecida para devolución del vehículo, o sea que si un cliente devuelve el vehículo después de la fecha establecida en el momento que alquiló el vehículo, pagará una multa por cada dia de retraso. Dicha multa será visualizada a través de una caja de diálogo en una etiqueta, la cual se muestra a continuación:



Botón ActualizarInventario: Este botón se utiliza para desbloquear las cajas de texto de tal manera que se pueda insertar datos en ellas (registrar un nuevo vehículo que adquiera el Rent a car), ya que inicialmente y después de cada registro guardado éstas deben permanecer bloqueadas.

Botón Salir: permitirá cerrar el formulario en caso de cancelar una operación y visualizar nuevamente el formulario frmbotonesformularios.

El formulario en vista de diseño se verá como la siguiente figura:



frmRegistrarCliente

El objetivo de este formulario es registrar un nuevo cliente en la base de datos.

9 Utiliza como fuente de datos la tabla Cliente.
10 Al igual que en el formulario anterior las cajas de texto se llamarán de igual manera que las etiquetas con el prefijo txt.
11 Tendrá dos botones: btnguardar y btncancelar.

Botón Guardar: verificará en la base de datos (tabla Cliente), por el número de cédula, si un cliente tiene o no un vehículo alquilado, en caso afirmativo emitirá un mensaje de información comunicando que ese cliente ya tiene alquilado un vehículo y que por política de la empresa, un cliente sólo puede tener alquilado un vehículo, de lo contrario lo registrará en la base de datos.

Botón Salir: cierra el formulario en caso de cancelar el registro del cliente y visualiza el formulario frmbotonesformularios.

El formulario en vista de diseño se verá como la siguiente figura:

frmBuscarCliente

El objetivo de este formulario es visualizar información sobre un cliente y el vehículo alquilado por este.

- Utiliza como fuente de datos las tablas Cliente y Vehículo.
- Añadir los siguientes componentes:

etiquetas: Nombres de Clientes, Modelo, Matrícula, Marca, Fecha de Devolución.
cajas de texto: txtmodelo, txtmatricula, txtmarca, txtfechadevolucion.
ComboBox: combonombreclientes
boton: btnsalir.

ComboBox: realiza la búsqueda por el nombre del cliente el cual será seleccionado del combo, en el combo solamente deben aparecer aquellos clientes que tienen alquilado algún vehículo. Al seleccionar un cliente en particular se debe visualizar en las cajas de texto la información referente al vehículo alquilado por este, por tanto las cajas de texto deben ser solamente de lectura.

Botón Salir: permitirá cerrar el formulario en caso de cancelar una operación y visualizar el formulario frmbotonesformularios.

El formulario en vista de diseño se verá como la siguiente figura:

frmRegistrarVehiculo

El objetivo de este formulario es capturar la información del vehículo solicitado por un cliente.

- Utiliza como fuente de datos la tabla Vehículo.
- Al igual que en el formulario frmRegistrarCliente las cajas de texto se llamarán de igual manera que las etiquetas (atributos de la tabla Vehículo) con el prefijo txt.
- Los objetos asociados a las etiquetas Matrícula y Días de alquiler deberán ser de tipo Combobox.
- Tendrá dos botones: btnguardar y btncancelar.
- Agregar un DataPicker (asegurarse que el componente gb.qt.kde este activo, Proyecto-Propiedades-Componentes).

Al cargar el formulario se debe inicializar el Combobox combomatricula con las matrículas de aquellos vehículos que se encuentran disponibles en el Rent a car y el Combobox combodias con números del 1 al 31 que corresponden al máximo número de días que tiene un mes, esto es debido a que el Rent a car tiene como política que el mínimo número de días para alquilar un vehículo es 1 y el máximo es 31. Ambos combos deben ser solamente de lectura.

Al seleccionar el número de matrícula correspondiente al vehículo requerido por el usuario se debe visualizar automáticamente la marca, el modelo y el depósito inicial de dicho vehículo en las respectivas cajas de texto, por tanto éstas deben ser de sólo lectura.

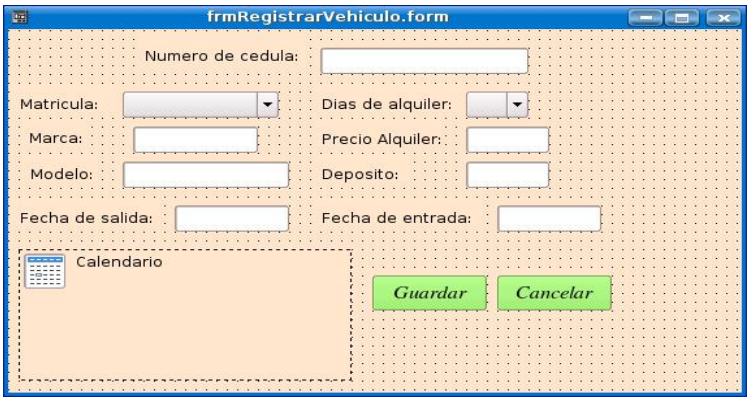
Al seleccionar del combo, el número de días de alquiler del vehículo, se debe visualizar automáticamente el precio total de alquiler del mismo (tomar en cuenta el precio por día de alquiler de cada vehículo) en la caja respectiva la cual debe ser de sólo lectura.

El objeto DataPicker es un calendario el cual se utilizará para seleccionar la fecha de salida del vehículo a través del evento Dblclick que será visualizada en la caja de texto respectiva y a la vez se debe visualizar la fecha de entrada en la caja respectiva(ambas cajas deben ser de sólo lectura), la cual será calculada en base a los días de alquiler seleccionados del combobox combodias. Ambas fechas deben mostrarse en el mismo formato y actualizarse según el orden en que se escojan los datos del DataPicker ó del combobox combodias.

Botón guardar: Una vez mostrado todos los datos en las cajas de texto, se verificará por el número de cédula si el cliente tiene alquilado algún vehículo, si es así se debe presentar un mensaje comunicando que ese cliente ya tiene alquilado un vehículo y que por política de la empresa un cliente sólo puede tener alquilado un vehículo, de lo contrario lo registrará en la base de datos.

Botón cancelar: cierra el formulario en caso de cancelar el registro del vehículo y debe visualizar el formulario frmbotonesformularios.

El formulario en vista de diseño se verá como la siguiente figura:



frmOrdenAlquiler

El objetivo de este formulario es visualizar e imprimir la orden de alquiler, la cual contiene los datos del vehículo solicitado por el cliente y deberá ser presentada al encargado de almacén para que éste entregue dicho vehículo.

- Utiliza como fuente de datos las tablas Cliente y Vehículo.
- Tendrá dos botones: btnImprimir y btnSalir.

Todas las cajas de texto de este formulario son de sólo lectura. La información que se debe mostrar en el formulario será obtenida al registrar el cliente y el vehículo alquilado por éste.

Botón Imprimir: al dar clic en él, se imprimirá la orden de alquiler correspondiente al cliente que en ese momento realice un alquiler de vehículo.

Botón Salir: cerrará el formulario.

El formulario en vista de diseño se verá como la siguiente figura:

frmFactura

El objetivo de este formulario es visualizar e imprimir la factura correspondiente al vehículo alquilado por un determinado cliente.

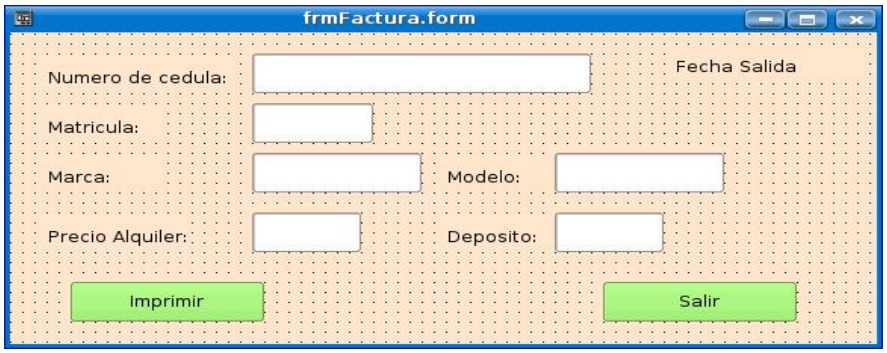
- Utiliza como fuente de datos las tablas Cliente y Vehículo.
- Tendrá dos botones: btnImprimir y btnSalir.

Todas las cajas de texto de este formulario son de sólo lectura. La información que se debe mostrar en el formulario será obtenida al registrar el cliente y el vehículo alquilado por éste.
La etiquieta Fecha Salida mostrará la fecha de salida del vehículo que se está alquilando.

Botón Imprimir: al dar clic en él, se imprimirá la factura con el número de cédula y datos del vehículo alquilado por éste.

Botón Salir: cerrará el formulario.

El formulario en vista de diseño se verá como la siguiente figura:



frmBotonesFormularios

En este formulario se deben añadir los siguientes componentes:

Botón Automaster: Al dar clic en el se debe cerrar el formulario frmbotonesformularios y visualizar el formulario frmAutomaster.

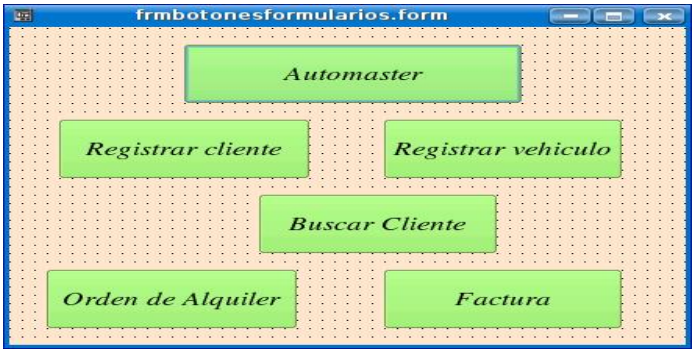
Botón Registrar Cliente: Al dar clic en el se debe cerrar el formulario frmbotonesformularios y visualizar el formulario frmRegistrarCliente.

Botón Registrar Vehiculo: Al dar clic en el se debe cerrar el formulario frmbotonesformularios y visualizar el formulario frmRegistrar Vehiculo.

Botón Buscar Cliente: Al dar clic en el se debe cerrar el formulario frmbotonesformularios y visualizar el formulario frmBuscarCliente.

Botón Orden Alquiler: Al dar clic en el se debe cerrar el formulario frmbotonesformularios y visualizar el formulario frmOrdenAlquiler.
Botón Factura: Al dar clic en el se debe cerrar el formulario frmbotonesformularios y visualizar el formulario frmFactura.

El formulario en vista de diseño se verá como la siguiente figura:



frmPrincipal

El objetivo de este formulario es que al arrancar la aplicación se visualice una imagen de un Carro o cualquier otra imagen que tenga relación con la aplicación. Al cabo de 5 segundos debe ocultarse dicho formulario y abrirse el formulario frmBotonesFormularios. Utilice un timer para controlar el tiempo de cierre y apertura de los formularios.

El formulario en vista de diseño se verá como la siguiente figura:

Practica 5.2 Elaboración de informes en OpenOffice 2.0 usando ODBC

Objetivo: El objetivo de esta práctica es que el estudiante conozca y aprenda a manejar OpenOffice 2.0 para la generación de informes accediendo a una base de datos MySQL.

Introducción

Los informes son herramientas utilizadas por el programador de la Base de Datos para brindar al usuario final datos en formato impreso. Los informes muestran la información desde listas simples de registros hasta agrupaciones con cálculos complejos sobre cada grupo de datos.
OpenOffice permite crear informes basados en tablas o consultas a través de un asistente, si la fuente de datos se encuentra almacenada en una tabla del sistema puede utilizar como fuente de datos la misma tabla; si los datos que desea mostrar se encuentran en distintas tablas cree una consulta con la información necesaria y dele formato al informe para mostrar la información dispuesta a su voluntad.

Como enlazar OpenOffice 2.0 a MySQL usando ODBC

Para poder utilizar el programa Database de OpenOffice es necesario instalar el driver ODBC,
para tener ODBC en nuestro sistema, instalamos el paquete **UNIXODBC** desde la consola através de la orden:

```
apt-get install unixodbc unixodbc-bin libmyodbc
```

o desde el Administrador de paquetes Adept a traves de Internet solicitamos la instalacion de estos tres paquetes (unixodbc, unixodbc-bin, libmyodbc).

El paquete **unixodbc** proporciona las librerías necesarias para crear el ODBC.
El paquete **unixodbc-bin** proporciona la interfaz gráfica para configurar el ODBC.
El paquete **libmyodbc** es el que instala el driver para MySQL.

Creación de los archivos .ini de ODBC

Ahora necesitamos configurar los archivos .ini utilizados por ODBC. Estos son archivos de texto que se pueden editar con algun editor de texto nano,pico, kate), estos archivos se encuentran en el directorio /etc. Estos son: odbcinst.ini y odbc.ini.

El archivo odbcinst.ini enumera los manejadores ODBC instalados en su sistema. Solamente necesitamos definir el manejador para MySQL de esta forma:

Desde la consola, ingrese como administrador y abra este archivo, en él, escriba lo siguiente:

```
[MySQL]
Driver            = /usr/lib/odbc/libmyodbc.so
Setup             = /usr/lib/odbc/libodbcmyS.so
FileUsage      = 1
```

El archivo odbc.ini define los vínculos que los usuarios utilizarán para conectarse a la base de datos a través del manejador.

Estando como administrador abra el archivo y edite lo siguiente:

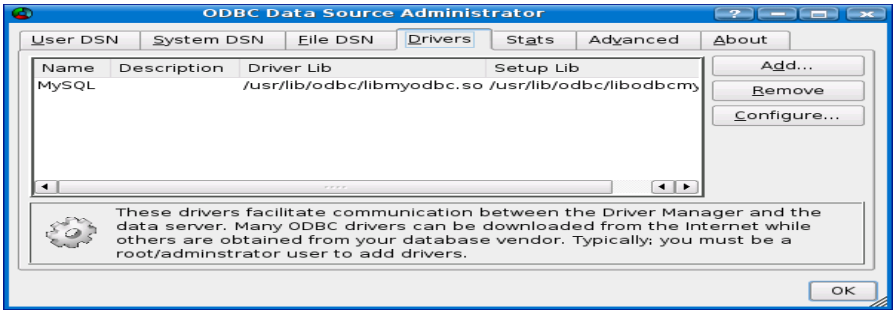
```
[nombre de la base de datos]
Driver             = MySQL
Server             = localhost
Database         = nombre de la base de datos a conectar
Port                 = 3306
```

una vez hecho esto se prueba el software de ODBC, utilizando un pequeño programa proporcionado por unixODBC llamado isql, estando como root debe escribir la siguiente orden:

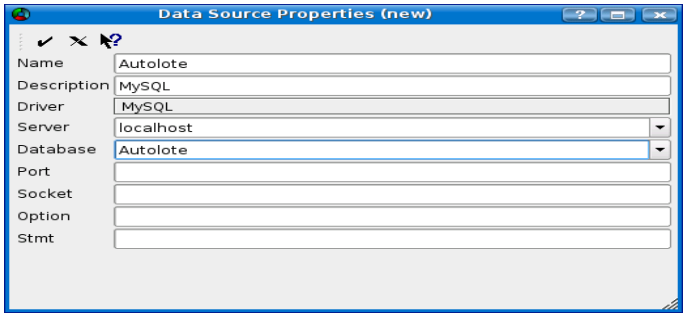
```
isql  nombre de la base de datos
```

Si esto no funciona, intente **isql nombre de la base de datos -v** para ver una lista completa de los errores.
El parámetro que se pasa a isql es pasado al programa de ODBC. Este busca el parámetro en el archivo de odbc.ini, y encuentra el manejador MySQL y la información de conexión. Después, busca MySQL en el archivo odbcinst.ini, y encuentra la localización de las librerías.

Ubíquese en el navegador Web (Konqueror), en la barra de dirección escriba locate:odbc y pulse enter, ahora escoja la carpeta (88 hits)/usr y dé clic en la aplicación ejecutable bin/ODBCConfig, se desplegará un cuadro de diálogo llamado **ODBC Data Source Administrator** éste contiene en la pestaña **Drivers** la información del archivo odbcinst.ini y en la pestaña **System DSN** el nombre y el Driver especificado en odbc.ini.



Ahora configuraremos el **User DSN**, dé clic en el botón Add... y seleccione el driver a usar, luego dé clic en el botón OK y escriba lo que se muestra en la figura siguiente, guarde esta configuración en el botón superior izquierdo representado con un check.



Ahora dé clic en el botón Ok de la ventana Administrador. Todo lo que se ha explicado anteriormente servirá para la configuración de ODBC desde OpenOffice.

Configuración de ODBC desde OpenOffice.org 2.0

OpenOffice buscará una librería denominada libodbc.so, Así que desde el usuario root cree el siguiente vínculo de esta manera:

```
ln –s /usr/lib/libodbc.so.1  /usr/lib/libodbc.so
```

Una vez que se haya hecho esto, estamos listos para configurar nuestro ODBC desde OpenOffice, para esto haga lo siguiente:

Escoja el Database de OpenOffice, Se abrirá el cuadro de diálogo Database Wizard, escoja la opción Conectar a una base de datos existente y seleccione MySQL, luego dé clic en el botón Next y escoja la opción conectar usando ODBC, dé clic en el botón Next, ahora debe escribir el nombre de la fuente de datos ODBC en su sistema o lo puede seleccionar del botón Browse y continúe dando clic en el botón Next.

Luego aparecerá un cuadro de texto donde debe especificarle el nombre y password correspondiente a la cuenta de usuario que se hizo para la conexión de Gambas con MySQL y pruebe la conexión en el botón **Test Connection**, una vez dando clic en éste se le preguntará el password del usuario especificado anteriormente, si el password es correcto aparecerá un mensaje indicando que la conexión fue establecida correctamente, y siga dando Next, como último paso dé en el botón **finish**.

Si todo esto funciona bien podrá acceder a su base de datos MySQL desde OpenOffice.

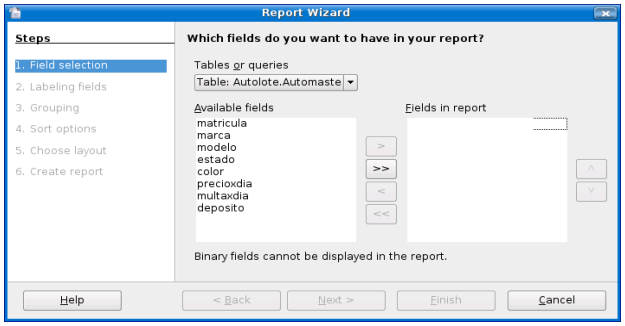
Familiarización con el entorno

Algunos ejemplos de informes a partir de los datos de la aplicación Rent a Car

Listado de todos los automoviles del Rent a Car

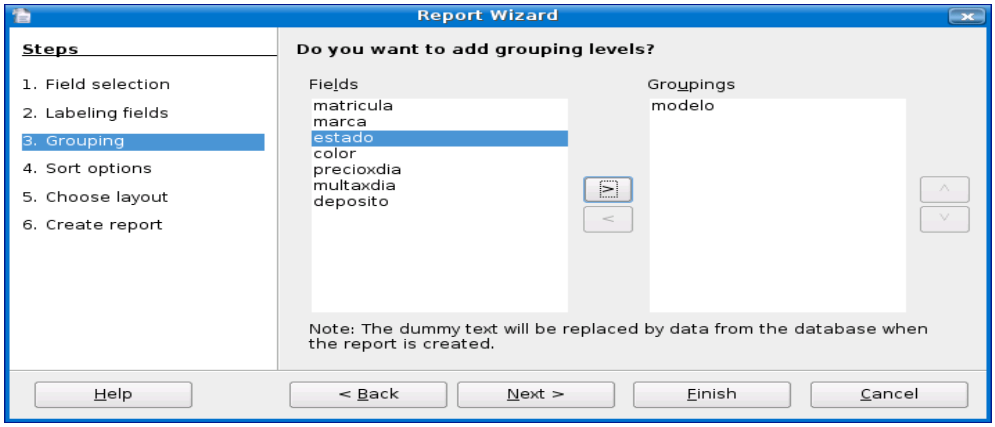
- Una vez realizada la configuracion del driver ODBC, abra el Database de OpenOffice y realice la conexión con la base de datos Autolote como se explico anteriormente.

- En este caso necesitamos generar un informe desde una misma tabla, por tanto seleccione el icono Report, luego escoja la opcion Use Wizard to Create Report.....
  A continuacion se desplegara el asistente de creacion de reportes y el writer de openoffice en el cual se visualizara el informe una vez creado. En el primer paso del asistente se muestran las tablas y atributos respectivos de la base de datos Autolote, seleccione la tabla Automaster y marque todos los atributos para generar el informe requerido (clic en >>), luego clic en next.



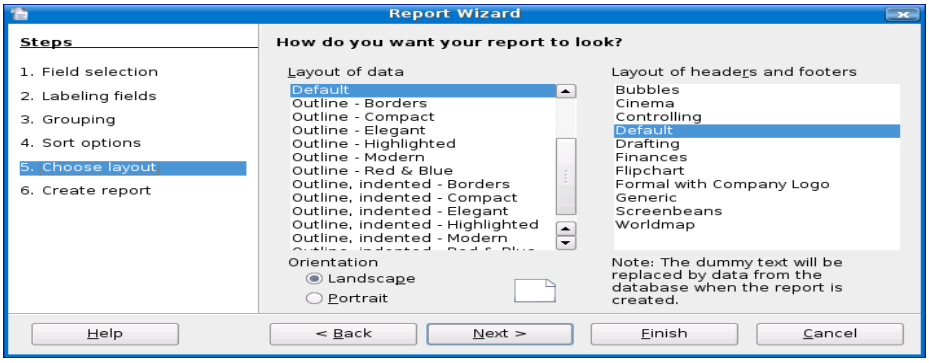
- En el segundo paso del asistente aparecen los campos seleccionados anteriormente con sus respectivas etiquetas, las cuales puede editar si lo desea, una vez editadas las etiquetas clic en next.

- A continuación el tercer paso es agrupar nuestra consulta por uno de los campos seleccionados, para esto de doble clic en el campo **modelo** y luego clic en next.

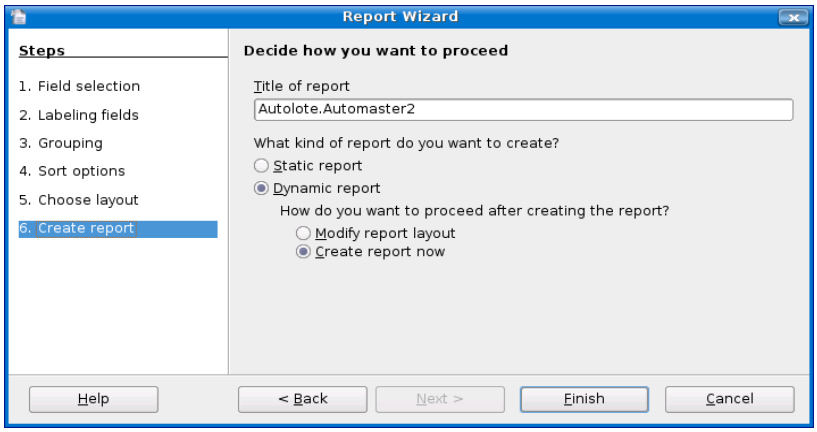


- Luego podemos ordenar el resultado de los datos en orden ascendente o descendente. En este caso dejamos el valor por defecto Ascendente, clic en next.

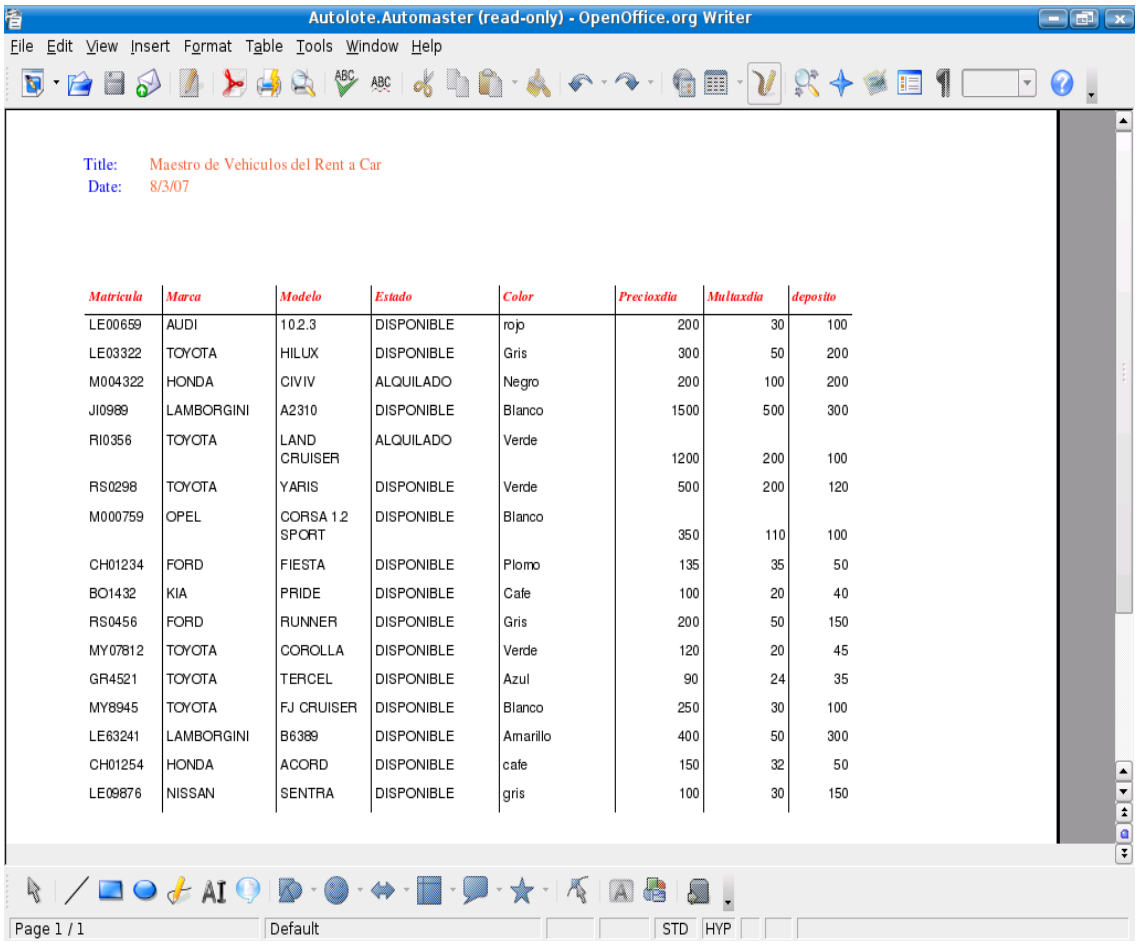
- A continuación podemos escoger el formato en el cual deseamos que aparezcan nuestros datos en el informe (en nuestro caso dejamos los valores por defecto), y clic en next.



- Por ultimo damos titulo a nuestro informe y clic en Finish.



- A continuación aparecerá el informe en el writer de openoffice con todos los vehículos del Rent a Car, como puede observar el formato no es muy agradable, por tanto edite el informe desde el Database de OpenOffice para darle formato a su gusto. Un ejemplo de salida para este informe se muestra en la siguiente figura.



Matricula	Marca	Modelo	Estado	Color	Precioxdia	Multaxdia	deposito
LE00659	AUDI	102.3	DISPONIBLE	rojo	200	30	100
LE03322	TOYOTA	HILUX	DISPONIBLE	Gris	300	50	200
M004322	HONDA	CIVIV	ALQUILADO	Negro	200	100	200
JI0989	LAMBORGINI	A2310	DISPONIBLE	Blanco	1500	500	300
RI0356	TOYOTA	LAND CRUISER	ALQUILADO	Verde	1200	200	100
RS0298	TOYOTA	YARIS	DISPONIBLE	Verde	500	200	120
M000759	OPEL	CORSA 1.2 SPORT	DISPONIBLE	Blanco	350	110	100
CH01234	FORD	FIESTA	DISPONIBLE	Plomo	135	35	50
BO1432	KIA	PRIDE	DISPONIBLE	Cafe	100	20	40
RS0456	FORD	RUNNER	DISPONIBLE	Gris	200	50	150
MY07812	TOYOTA	COROLLA	DISPONIBLE	Verde	120	20	45
GR4521	TOYOTA	TERCEL	DISPONIBLE	Azul	90	24	35
MY8945	TOYOTA	FJ CRUISER	DISPONIBLE	Blanco	250	30	100
LE63241	LAMBORGINI	B6389	DISPONIBLE	Amarillo	400	50	300
CH01254	HONDA	ACORD	DISPONIBLE	cafe	150	32	50
LE09876	NISSAN	SENTRA	DISPONIBLE	gris	100	30	150

Clientes que tienen algún vehículo alquilado con su fecha de devolución.

- Este informe involucra dos tablas para generar los datos requeridos, estas son: Vechiculo y Cliente, por tanto desde el OpenOffice seleccione el icono Query para crear la consulta, luego escoja la opcion Create Query in SQL View..... Se abrira la vista SQL en la cual debe editar la siguiente consulta:

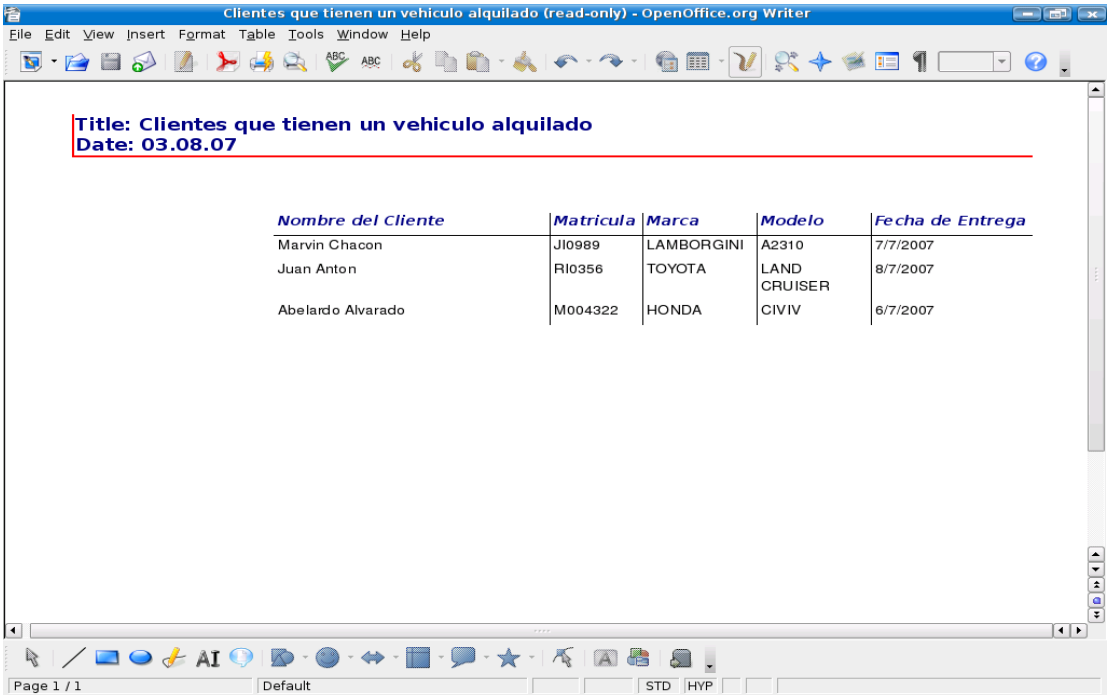
```
              Select
Cliente.NombreCliente,Vehiculo.matricula,Vehiculo.marca,Vehiculo.m
odelo,
     Vehiculo.FechaEntrada from Cliente,Vehiculo where
Cliente.NCedula = Vehiculo.NCedula
```

- Cierre la vista SQL, a continuación le pedirá un nombre para la consulta (deje el nombre por defecto). Para observar si los resultados son correctos, de doble clic sobre el nombre que dio a la consulta creada, si son correctos proceda a realizar el informe. Para ello dé clic derecho sobre la consulta creada y seleccione la opción Report Wizard, con lo cual se abrirá el asistente para crear el informe.

- En el paso uno del asistente aparece por defecto el nombre de la consulta creada y los atributos que desea mostrar en el informe, seleccione dicha consulta y siga los pasos del asistente a como se explico en el ejemplo anterior.

- Una vez creado el informe, dé formato al mismo y observe los resultados.

- Un ejemplo de salida se muestra en la siguiente figura.



Ejercicios Propuestos

- Generar un informe que refleje el monto total del alquiler de vehículos hasta la fecha de ejecución del informe. Se debe reflejar cada vehículo, el número de días de alquiler, el precio por día y el monto total de ese vehículo.

- Generar un informe con todos aquellos clientes que tienen una multa por incumplimiento en la entrega del vehículo en la fecha establecida con su respectivo monto.

PRACTICA 6

CREACIÓN DE APLICACIONES EN VISUAL BASIC CON ACCESO A UNA BASE DE DATOS MYSQL.

Objetivo:

Conocer y aprender a trabajar en una nueva plataforma (windows) con aplicaciones que necesiten acceso a una base de datos MySQL.

Introducción.

El desarrollo de aplicaciones con acceso a una base de datos MySQL en plataforma Windows es muy similar y a la vez sencillo como trabajar en plataforma Linux. El desarrollo de las aplicaciones realizadas en Linux se hizo utilizando un entorno de desarrollo muy parecido a Visual Basic y el gestor de bases de datos fue el mismo con el que vamos a trabajar en el desarrollo de esta practica, es por esa razón que nos limitaremos a darles solamente información básica sobre las diferencias mínimas en cuanto a modo de conexión y manipulación de la base de datos.

Cualquier ayuda adicional que se requiera para el desarrollo de dicha práctica puede consultarse la ayuda completa que trae Visual Basic, o la bibliografía mencionada en este trabajo.

Configuración del controlador ODBC.

El controlador MySQL con el que vamos a trabajar es el mysql-odbc-3.5.1, por tanto debe estar instalado en su ordenador para poder realizar esta practica.

Antes de establecer la conexión con la base de datos hay que configurar el controlador ODBC a través del cual podremos acceder a ella. Para ello realice los siguientes pasos:

1) Desde Inicio, escoja **Panel de control - Herramientas Administrativas - Origen de Datos ODBC.**

2) Luego de clic sobre el botón **Agregar,** aparecerá una lista de drivers, seleccione el driver **MySQL ODBC 3.5.1 driver,** y de clic en el botón finalizar.

3) A continuación aparecerá un cuadro de dialogo en el que se deben especificar los siguientes campos y luego clic en **OK**

Data Source Name	Cualquier nombre que haga referencia a la fuente de datos. Por Ej.: myodbc
Server	Localhost
User	Root
Password	Clave de acceso a mysql
DataBase	Base de datos a conectar

Con esto ya tenemos configurado y asignado un nombre para nuestro controlador ODBC.

Como establecer la conexión desde Visual Basic con MySQL.

```
1. Declarar un objeto de tipo Connection y uno de tipo Recordset.

Public con As ADODB.Connection
Public rt As ADODB.Recordset
```

Objeto tipo Connection: Sirve para establecer la conexión y apertura de la base de datos.

Objeto tipo Recordset: Sirve para acceder a una tabla específica de la base de datos y manipular sus registros.

Una forma de realizar la conexión es a través de una función la cual será llamada desde el espacio de trabajo que sea necesario. Por Ej.

```
Public Function conectar()
  Set con = New ADODB.Connection
  Set rt = New ADODB.Recordset
  con.CommandTimeout = 40
  con.CursorLocation = adUseClient
  con.Open
  "Server=localhost;Provider=MSDASQL;UserId=root;Password=mysql123; Data
Source=myodbcagenda;db=agenda"
End Function
```

Para poner en práctica todo lo comentado anteriormente vamos a realizar una aplicación con acceso a una base de datos MySQL, la cual les servirá para poder desarrollar posteriormente una aplicación propuesta.

Aplicación Mi Agenda Telefónica

El proyecto consiste en la creación de una Agenda Telefónica en la cual se pueda tener un registro de todos los contactos, modificarlos y borrarlos en caso necesario.

1. Crear una base de datos llamada Agenda con las siguientes tablas

   Tabla registros
   Atributos: nombre varchar (50), direccion varchar (50), telefono varchar (8), celular varchar (7), email varchar (20).

   Tabla clave
   Atributo: nclave varchar (10)

2. Creación de la Interfaz de la Aplicación

Para poder acceder a la aplicación se deberá mostrar un formulario en el cual se pedirá una clave de acceso, el formulario en tiempo de diseño se muestra a continuación:

Resultado del diseño del formulario frmclaveacceso

Una vez realizado el diseño del formulario procedemos a su codificación

En la carga del formulario añadir el siguiente código

```
Private Sub Form_Load()
 'conectarme con la BD_directorio
 conectar
 rt.Open "SELECT * FROM clave", con, adOpenDynamic, adLockOptimistic
End Sub
```

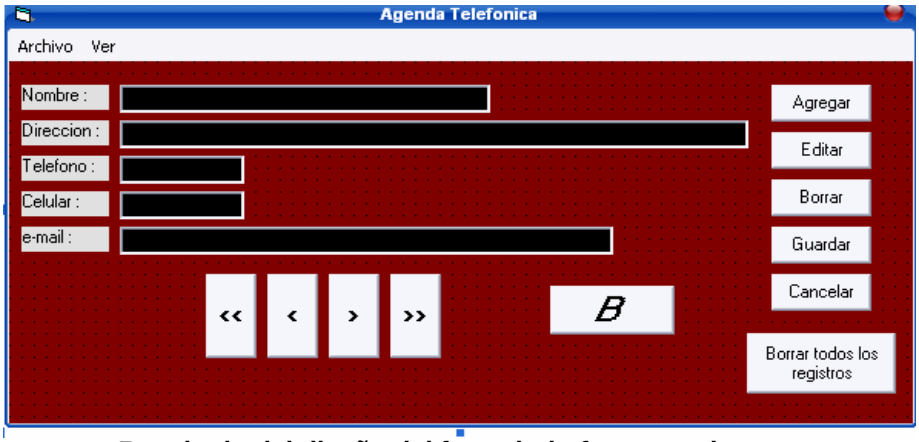
Boton Aceptar

```
Private Sub aceptar_Click()

    If rt!nclave = clave.Text Then
        frmclaveacceso.Hide
        frmgeneral.Show 1

         Else
           MsgBox "Clave invalida", vbCritical, "Mi Directorio"
           clave.Text = ""
           clave.SetFocus
    End If

End Sub
```

Boton Cancelar

```
Private Sub cancelar_Click()
    clave.Text = ""
    clave.SetFocus
End Sub
```

Al dar clic en Aceptar se oculta el formulario **frmclaveacceso** y se visualiza el formulario **frmgeneral**, cuyo diseño se muestra a continuación:



Resultado del diseño del formulario frmgeneral

Codificación

En la carga del formulario:

```
Private Sub Form_Load()

  VerificarEstadoRecordset
  rt.Open "SELECT * FROM registros", con, adOpenDynamic, adLockOptimistic

  If rt.RecordCount = 0 Then
     BD_Cajas txtnombre, txtdireccion, txttelf, txtcel, txtemail, True
     BD_Boton cmdeditar, True
     BD_Boton cmdguardar, True
     BD_Boton cmdborrar, True
     BD_Boton cmdcancelar, True
     BD_Boton borrar, True
     BD_Boton cmdbuscar, True
     BD_Botones_desplazamiento cmdprimero, cmdanterior, cmdsig, cmdultimo,
True
     MsgBox "Base de Datos vacia", vbInformation, "Mi directorio"
     Exit Sub
  End If

  'colocar el puntero en el primer registro
  rt.MoveFirst

  'visualizar el 1er reg en los respectivos controles
  txtnombre.Text = rt!nombre
  txtdireccion.Text = rt!direccion
  txttelf.Text = rt!telf
  txtcel.Text = rt!celular
  txtemail.Text = rt!email

  BD_Cajas txtnombre, txtdireccion, txttelf, txtcel, txtemail, True
  BD_Boton cmdcancelar, True
  BD_Boton cmdguardar, True
End Sub
```

Botón Agregar

```
Private Sub cmdagregar_Click()

   Limpiar_Cajas txtnombre, txtdireccion, txttelf, txtcel, txtemail
   BD_Boton cmdagregar, True
   BD_Botones_desplazamiento cmdprimero, cmdanterior, cmdsig,
cmdultimo,True
   BD_Boton cmdborrar, True
   BD_Boton borrar, True
   BD_Boton cmdbuscar, True
   BD_Boton cmdeditar, True
   BD_Boton cmdguardar, False
   BD_Boton cmdcancelar, False
   BD_Cajas txtnombre, txtdireccion, txttelf, txtcel, txtemail, False

   txtnombre.SetFocus
   Cambiar_Fondo txtnombre, txtdireccion, txttelf, txtcel, txtemail, True
 End Sub
```

Botón Editar

```
Private Sub cmdeditar_Click()

    editado = True
    BD_Cajas txtnombre, txtdireccion, txttelf, txtcel, txtemail, False
    BD_Boton cmdeditar, True
    BD_Boton cmdborrar, True
    BD_Boton cmdagregar, True
    BD_Boton cmdguardar, False
    BD_Boton cmdcancelar, False
    BD_Boton cmdbuscar, True
    BD_Boton borrar, True
    BD_Botones_desplazamiento cmdprimero, cmdanterior, cmdsig, cmdultimo,
True
    Cambiar_Fondo txtnombre, txtdireccion, txttelf, txtcel, txtemail, True

End Sub
```

Botón Borrar

```
Private Sub cmdborrar_Click()
Dim BDVacia As Boolean
Dim nreg As Integer
Dim ultimo As Integer
    If MsgBox("Esta seguro que desea borrar el registro actual ?", vbYesNo,
      "Borrar registro") = vbYes Then
    nreg = rt.AbsolutePosition
    ultimo = rt.RecordCount
            rt.Delete
            rt.Update

             'si la BD ha quedado vacia
             BDVacia = Not CBool(rt.RecordCount)
             If BDVacia Then
                MsgBox "Ha borrado el ultimo registro de la Base de Datos",
                  vbDefaultButton2, "Borrar registro"
     Limpiar_Cajas txtnombre, txtdireccion, txttelf, txtcel, txtemail
                Exit Sub
             End If

         If nreg = 1 Then
            rt.MoveNext

            ElseIf nreg = ultimo Then
              rt.MovePrevious

              Else
                rt.MoveFirst
          End If

        VerRegistroActual
        MsgBox "Registro borrado", vbInformation, "Borrar registro"
    End If
End Sub
```

Botón Guardar

```
Private Sub cmdguardar_Click()
Dim vacia As Boolean

    If txtnombre.Text = "" And txtdireccion.Text = "" And txttelf.Text = ""
      And txtcel.Text = "" And txtemail.Text = "" Then
        MsgBox "No ha introducido datos", vbInformation, "Guardar registro"
        txtnombre.SetFocus
        Exit Sub
    End If

      If txtnombre.Text = "" Or txttelf.Text = "" Then
         MsgBox "Nombre y telefono reguerido", vbCritical, "Guardar
      registro"
         txtnombre.SetFocus
         Exit Sub
      End If

    BD_Cajas txtnombre, txtdireccion, txttelf, txtcel, txtemail, True
    BD_Boton cmdguardar, True
    BD_Boton cmdcancelar, True
    BD_Boton cmdagregar, False
    BD_Boton cmdeditar, False
    BD_Boton cmdborrar, False
    BD_Boton cmdbuscar, False
    BD_Boton borrar, False

    BD_Botones_desplazamiento cmdprimero, cmdanterior, cmdsig, cmdultimo,
      False
    Cambiar_Fondo txtnombre, txtdireccion, txttelf, txtcel, txtemail, False

    vacia = Not CBool(rt.RecordCount)

    If vacia = True Then
        rt.AddNew
        rt!nombre = txtnombre.Text
        rt!direccion = txtdireccion.Text
        rt!telf = txttelf.Text
        rt!celular = txtcel.Text
        rt!email = txtemail.Text
       'actualizar el recordset
        rt.Update
        MsgBox "El registro ha sido agregado satisfactoriamente ",
        vbInformation, "Guardar registro"
        Exit Sub
    End If

       If editado = True Then
          rt!nombre = txtnombre.Text
          rt!direccion = txtdireccion.Text
          rt!telf = txttelf.Text
          rt!celular = txtcel.Text
          rt!email = txtemail.Text
          'actualizar el recordset
           rt.Update
           MsgBox "Registro actualizado", vbInformation, "Guardar registro"
           Exit Sub
       End If
    rt.AddNew
    rt!nombre = txtnombre.Text
    rt!direccion = txtdireccion.Text
```

```
    rt!telf = txttelf.Text
    rt!celular = txtcel.Text
    rt!email = txtemail.Text

    'actualizar el recordset
    rt.Update

    MsgBox "El registro ha sido agregado satisfactoriamente ",
      vbInformation, "Guardar registro"

    'visualizar el registro actualmente ingresado
    VerRegistroActual

End Sub
```

Botón Cancelar

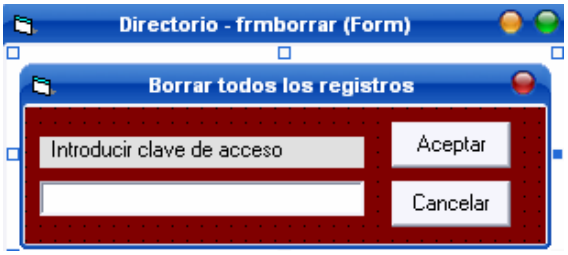
```
Private Sub cmdcancelar_Click()
   'si no hay ningun registro en la BD
    If rt.RecordCount = 0 Then
        'bloquear los botones correspondientes
         BD_Cajas txtnombre, txtdireccion, txttelf, txtcel, txtemail, True
         Cambiar_Fondo txtnombre, txtdireccion, txttelf, txtcel, txtemail,
         False
         BD_Boton cmdguardar, True
         BD_Boton cmdcancelar, True
         BD_Boton cmdagregar, False

        Else
            BD_Cajas txtnombre, txtdireccion, txttelf, txtcel, txtemail,
            True
            BD_Boton cmdguardar, True
            BD_Boton cmdcancelar, True
            BD_Boton cmdeditar, False
            BD_Boton cmdborrar, False
            BD_Boton cmdbuscar, False
            BD_Boton borrar, False
            BD_Boton cmdagregar, False
            BD_Botones_desplazamiento cmdprimero, cmdanterior, cmdsig,
            cmdultimo, False
            Cambiar_Fondo txtnombre, txtdireccion, txttelf, txtcel,
            txtemail, False
            VerRegistroActual
    End If

End Sub
```

Botón Borrar todos los registros

Al dar clic en este botón se abrirá un formulario llamado frmborrar en el cual se debe introducir una clave de acceso a manera de seguridad para poder borrar todos los registros, al dar clic en aceptar se muestra un mensaje preguntando si realmente se desean borrar todos los registros, si es si se emite un mensaje informando que la base de datos ha quedado vacía.

```
Private Sub borrar_Click()
    frmborrar.Show 1
End Sub
```

El diseño del formulario frmborrar se muestra a continuación:



Botón Buscar

```
Private Sub cmdbuscar_Click()
Dim Nombre_b, com, nombre As String
Dim long_Nombre_b As Integer
    Nombre_b = InputBox("Escriba el nombre del registro que desea buscar ",
      "Buscar Registro")
    If Nombre_b = "" Then Exit Sub

    rt.MoveFirst

    While Not rt.EOF

        If UCase(Nombre_b) = UCase(rt!nombre) Then
            txtnombre.Text = rt!nombre
            txtdireccion.Text = rt!direccion
            txttelf.Text = rt!telf
            txtcel.Text = rt!celular
            txtemail.Text = rt!email
            Exit Sub
        End If
      rt.MoveNext
    Wend

  rt.MoveFirst
  VerRegistroActual
  MsgBox "Registro no encontrado", vbCritical, "Mi directorio"
End Sub
```

Botones de desplazamiento

Primero:

```
Private Sub cmdprimero_Click()
    If rt.AbsolutePosition = 1 Then
        Exit Sub
    End If
    rt.MoveFirst
    VerRegistroActual
End Sub
```

Siguiente:

```
Private Sub cmdsig_Click()

    If rt.AbsolutePosition = rt.RecordCount Then
        Exit Sub
    End If
    rt.MoveNext
    VerRegistroActual

End Sub
```

Anterior:

```
Private Sub cmdanterior_Click()
     If rt.AbsolutePosition = 1 Then
        Exit Sub
    End If
    rt.MovePrevious
    VerRegistroActual
End Sub
```

Ultimo:

```
Private Sub cmdultimo_Click()
    rt.MoveLast
    VerRegistroActual
End Sub
```

Como pueden observar en este formulario hay una barra de menú con dos opciones:
Archivo y Ver. En la opción archivo hay un submenú: **Salir** y en la opción ver un submenú: **Todos los registros**

Submenú Salir:

```
Private Sub Salir_Click()
   Me.Hide
End Sub
```

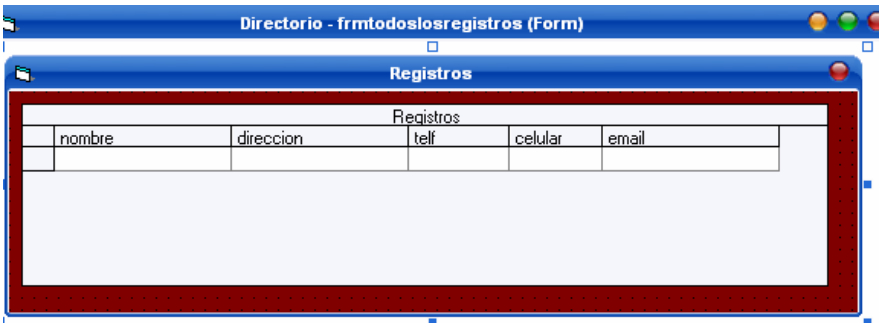
Al dar clic en este submenú se cierra el formulario

Submenu Todos los registros

```
Private Sub todos_Click()
    Me.Hide
    frmtodoslosregistros.Show
End Sub
```

Al dar clic en este submenú se debe abrir un formulario llamado **frmtodoslosregistros** en el cual se deben mostrar todos los contactos que hay en la agenda telefónica a través de un Datagrid.

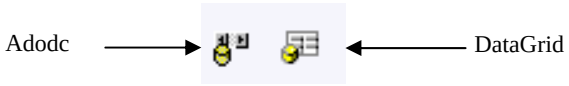
El diseño del formulario se muestra a continuación:



Resultado del diseño del formulario frmtodoslosregistros

Para agregar un DataGrid a un formulario es necesario ubicarse en la barra de menú y debe dar clic en Proyecto, luego escoger la opción Componentes y agregar de la pestaña Controles "**Microsoft DataGrid Control 6.0 (OLEDB)**", debe de estar seguro que está activo el control "**Microsoft ADO Data Control 6.0 (OLEDB)**", ya que éste control nos será útil.

Cuando los controles mencionados anteriormente están activos, estos se pueden encontrar en la barra de Herramientas ubicada en la parte izquierda del espacio de trabajo



Ahora simplemente hay que agregar ambos controles en el formulario, luego tiene que hacer la conexión del control Adodc con la base de datos y luego enlazarlo con el DataGrid, esto se hace de la siguiente forma:
En las propiedades del Adodc se encuentra la propiedad "**Connection String**" dar clic sobre ella y aparecerá un cuadro de diálogo llamado "Pagina de propiedades" allí escoger la opción "**usar cadena de conexión**", dar clic en el botón **generar.** Luego aparecerá otro cuadro de dialogo donde se debe seleccionar el controlador "**Microsoft OLE DB provider for ODBC Drivers"** y dar clic en siguiente, luego escoger la opción "**usar el nombre de origen de datos**" y seleccione el controlador ODBC que haya configurado (en nuestro caso **myodbc**) anteriormente, si desea probar la conexión de clic en el botón **probar conexión** y para finalizar de clic sobre el botón **Aceptar** para que los cambios se realicen.

A continuación de clic en la propiedad "**RecordSource**" del Adodc, aparecerá un cuadro de dialogo llamado Origen de registros, en la opción "**tipo de comando**", escoja **"2-adCmdTable"** y en la opción "**Tabla o nombre de procedimiento almacenado"** escoja la tabla con la que se quiere enlazar el DataGrid (en nuestro caso con la tabla **registros**).

Para enlazar el DataGrid con el control Adodc, de clic en la propiedad DataSource del DataGrid y seleccione el nombre del Adodc. El DataGrid tiene en su página de propiedades opciones que permiten realizar algunas funcionalidades sobre los registros., por ejemplo agregar, borrar y editar. En este caso solamente vamos a permitir editar por lo tanto de clic derecho sobre el DataGrid y seleccione la opción propiedades, aparecerá la pagina de propiedades en la cual debe activar solamente la opción AllowUpdate.

Una vez ejecutada la aplicación se puede editar cualquier registro, únicamente debe moverse de celda en el DataGrid para que los cambios sean guardados en la base de datos.

Validaciones de las cajas de texto

Caja de texto txtnombre

```
Private Sub txtnombre_KeyPress(KeyAscii As Integer)
    Dim a_min, z_min, A, Z, space, backspace, punto As Integer
    a_min = 97
    z_min = 122
    A = 65
    Z = 90
    space = 32
    backspace = 8
    punto = 46
    If (Not ((KeyAscii >= a_min And KeyAscii <= z_min) Or (KeyAscii >= A
And   KeyAscii <= Z) Or KeyAscii = space Or KeyAscii = backspace Or
KeyAscii    = punto)) Then   KeyAscii = 0
    End If
End Sub
```

Caja de texto txtdireccion

```
Private Sub txtdireccion_KeyPress(KeyAscii As Integer)
    Dim a_min, z_min, A, Z, space, backspace, punto, cero, nueve As Integer
    a_min = 97
    z_min = 122
    A = 65
    Z = 90
    space = 32
    backspace = 8
    punto = 46
    cero = 48
    nueve = 57
    If (Not ((KeyAscii >= a_min And KeyAscii <= z_min) Or (KeyAscii >= A
And   KeyAscii <= Z) Or KeyAscii = space Or KeyAscii = backspace Or
KeyAscii    = punto Or (KeyAscii >=  cero And KeyAscii <= nueve))) Then
                  KeyAscii = 0
    End If
End Sub
```

Caja de texto txttelf

```
Private Sub txttelf_KeyPress(KeyAscii As Integer)
    Dim cero, nueve, guion, backspace As Integer

    cero = 48
    nueve = 57
    backspace = 8
    guion = 45

    If (Not ((KeyAscii >= cero And KeyAscii <= nueve) Or KeyAscii =
      backspace Or KeyAscii = guion)) Then  KeyAscii = 0
    End If
End Sub
```

Caja de texto txtcel

```
Private Sub txtcel_KeyPress(KeyAscii As Integer)
    Dim cero, nueve, backspace As Integer

    cero = 48
    nueve = 57
    backspace = 8

    If (Not ((KeyAscii >= cero And KeyAscii <= nueve) Or KeyAscii =
backspace)) Then  KeyAscii = 0
    End If
End Sub
```

Modulo Funciones

```
Public con As ADODB.Connection
Public rt As ADODB.Recordset
Public editado As Boolean
Public idprod As Integer
```

```
Public Sub BD_Cajas(ByVal cnombre As TextBox, ByVal cdir As TextBox, ByVal
ctele As TextBox, ByVal ccel As TextBox, ByVal cemail As TextBox, ByVal aux
As Boolean)

    If aux = True Then
        cnombre.Enabled = False
        cdir.Enabled = False
        ctele.Enabled = False
        ccel.Enabled = False
        cemail.Enabled = False
      Else
        cnombre.Enabled = True
        cdir.Enabled = True
        ctele.Enabled = True
        ccel.Enabled = True
        cemail.Enabled = True
     End If

End Sub
```

```
Public Sub Limpiar_Cajas(ByVal nombre As TextBox, ByVal dire As TextBox,
ByVal tele As TextBox, ByVal cel As TextBox, ByVal email As TextBox)
    nombre.Text = ""
    dire.Text = ""
    tele.Text = ""
    cel.Text = ""
    email.Text = ""
End Sub
```

```
Public Sub BD_Boton(ByVal boton As CommandButton, ByVal aux As Boolean)
    If aux = True Then
        boton.Enabled = False
      Else
        boton.Enabled = True
    End If
End Sub
```

```
Public Sub BD_Botones_desplazamiento(ByVal b1 As CommandButton, ByVal b2 As
CommandButton, ByVal b3 As CommandButton, ByVal b4 As CommandButton, ByVal
aux As Boolean)
    If aux = True Then
          b1.Enabled = False
          b2.Enabled = False
          b3.Enabled = False
          b4.Enabled = False
        Else
          b1.Enabled = True
          b2.Enabled = True
          b3.Enabled = True
          b4.Enabled = True
    End If
End Sub
```

```
Public Function conectar()

   Set con = New ADODB.Connection
   Set rt = New ADODB.Recordset
   con.CommandTimeout = 40
   con.CursorLocation = adUseClient
```

```
   con.Open
"Server=localhost;Provider=MSDASQL;UserId=root;Password=mysql123;Data
Source=myodbcagenda;db=agenda"
End Function
```

```
Public Function VerificarEstadoRecordset()

     If (rt.State = adStateOpen) Then
          rt.Close
      End If
End Function
```

```
Public Sub Cambiar_Fondo(ByVal c1 As TextBox, ByVal c2 As TextBox, ByVal c3
As TextBox, ByVal c4 As TextBox, ByVal c5 As TextBox, ByVal aux As Boolean)
    If aux = True Then
         'fondo blanco para las cajas
        c1.BackColor = &HFFFFFF
        c2.BackColor = &HFFFFFF
        c3.BackColor = &HFFFFFF
        c4.BackColor = &HFFFFFF
        c5.BackColor = &HFFFFFF
    Else
        c1.BackColor = &H0
        c2.BackColor = &H0
        c3.BackColor = &H0
        c4.BackColor = &H0
        c5.BackColor = &H0
    End If

End Sub
```

PRACTICA 6.1

CREACIÓN DE UNA APLICACIÓN DESARROLLADA EN VISUAL BASIC CON ACCESO A UNA BASE DE DATOS MYSQL.

CONTINUACIÓN DE LA PRÁCTICA ANTERIOR

Ejercicio Propuesto

Una empresa distribuidora de productos desea tener un registro de todos sus productos, clientes y vendedores, así como la facturación de pedidos realizados a dicha empresa. Para ello hay que realizar una base de datos en la cual se guardara toda la información antes mencionada.

1. Crear una base de datos llamada ClienteVendedor con las siguientes tablas:

Tabla Cliente:
Atributos: NombreCliente varchar(30), Direccion varchar(50), Telefono varchar(8).

Tabla Producto:
Atributos: IdProd varchar(4) primary key, NombreProd varchar(20), PrecioCompra double, PrecioVenta double, Cantidad int, FechaCompra varchar(10).

Tabla Vendedor:
Atributos: NombreVendedor varchar(30), FechaNac varchar(10), Direccion varchar(50), Telefono varchar(8), FechaIngreso varchar(10), Antigüedad int.

Tabla Factura:
Atributos: NoFactura int primary key, Fecha varchar(10), Tipopago varchar(7), NombreVendedor varchar(30), NombreCliente varchar(30), MontoTotal double.

Tabla DetalleFactura:
Atributos: NoFactura int , IdProd varchar(4),NombreProd varchar(20), Cantidad int, PrecioVenta double, Monto double.

2. Todas las tablas a excepción de la tabla Producto deben estar vacías inicialmente, por tanto ingrese los siguientes registros en la tabla Producto:

IdProd	NombreProd	PrecioCompra	PrecioVenta	FechaCompra	Cantidad
0001	Calendarios de bolsillo	3,50	5,50	12/01/2007	200
0002	Parlantes para computador	110,00	117,00	12/01/2007	50
0003	Queso de crema	8,50	11,50	28/02/2007	150
0004	Leche descremada	5,00	7,00	30/02/2007	50
0005	Pan barra Bimbo	7,25	11,25	09/04/2007	10
0006	Margarina Unimar	10,00	15,00	15/05/2007	75
0007	Pan barra Aurora	7,00	10,00	15/05/2007	20
0008	Margarina Cremy	13,00	15,00	18/05/2007	20
0009	Cebollas Criollas	3,50	4,50	25/05/2007	100
0010	Tamales Pisques	2,50	3,50	01/07/2007	40
0011	Repollo Criollo	5,00	7,00	08/07/2007	15
0012	Mostaza Mcormik	11,14	14,14	10/07/2007	40
0013	Tarjetas BellSouth	100,00	112,00	14/08/2007	50
0014	CD-R 80min-700MB	12,00	15,00	25/08/2007	300

3. Crear los siguientes formularios

Formulario frmcliente

El objetivo de este formulario es registrar un nuevo cliente que realiza un pedido a la empresa.

- Utiliza como fuente de datos la tabla Cliente
- Tendrá 3 cajas de texto con sus respectivas etiquetas nombradas con el mismo nombre del atributo de la tabla y 10 botones de pulsación.

Botón Nuevo: Activa las cajas de texto para ingresar un nuevo registro.

Botón Guardar: registra en la base de datos al nuevo cliente especificado en las cajas de texto.

Botón Editar: Permitirá actualizar o modificar la información de un determinado cliente.

Botón Buscar: Abre una caja de dialogo en la que se debe especificar el nombre del cliente a buscar y visualiza toda la información de ese cliente en las respectivas cajas de texto.

Botón Borrar: Emite un mensaje indicando si se desea borrar el registro actual (el que se encuentra en ese momento en las cajas de texto), en caso afirmativo borra el registro de la base de datos y actualiza el objeto recordset.

Botón Cancelar*:* En caso de no realizar el registro del cliente, muestra en las cajas de texto el primer registro de la tabla.

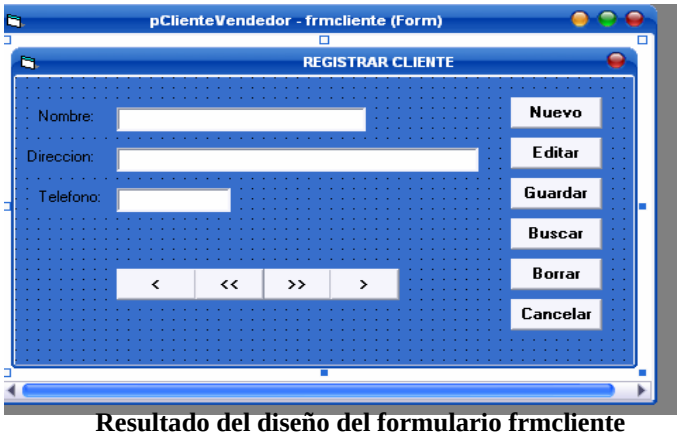
Botones de desplazamiento

Primero: Muestra el primer registro de la tabla.

Anterior: Verifica si el registro al que apunta el objeto recordset no es el primero, en caso afirmativo no se desplaza y sigue mostrando ese registro en caso contrario muestra el registro anterior.

Siguiente: Verifica si el registro al que apunta el objeto recordset no es el ultimo, en caso afirmativo no se desplaza y sigue mostrando ese registro en caso contrario muestra el siguiente registro.

Ultimo: Muestra el ultimo registro de la tabla.



Resultado del diseño del formulario frmcliente

Formulario frmvendedor

El objetivo de este formulario es registrar un nuevo vendedor que contrate la empresa.

- Utiliza como fuente de datos la tabla Vendedor
- Tendrá 6 cajas de texto con sus respectivas etiquetas nombradas con el mismo nombre del atributo de la tabla y 10 botones de pulsación.

Nota: La funcionalidad de los botones de pulsación es la misma que en el formulario frmcliente, por tanto solo modifique o agregue lo que haga falta para darle funcionalidad al formulario, tenga en cuenta validar las cajas de texto.

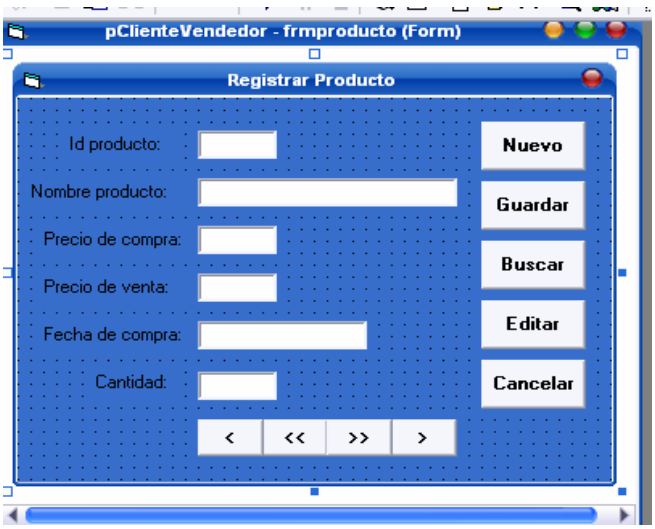
Resultado del diseño del formulario frmvendedor

Formulario frmproducto

El objetivo de este formulario es registrar, modificar y visualizar los productos de la empresa.

- Utiliza como fuente de datos la tabla Producto
- Tendrá 6 cajas de texto con sus respectivas etiquetas nombradas con el mismo nombre del atributo de la tabla y 9 botones de pulsación.

Nota: La única función que no tendrá este formulario es borrar un registro, por tanto será el único botón que a diferencia de los formularios anteriores no aparecerá. La funcionalidad de los demás es igual que en los otros formularios.



Resultado del diseño del formulario frmproducto

Formulario frmdetallefactura

El objetivo de este formulario es realizar el detalle de los productos solicitados por los clientes. .

- Utiliza como fuente de datos las tablas DetalleFactura y Factura.
- Tendrá 6 cajas de texto, 3 comboBox, 2 botones de radio y 6 botones de pulsación. Las etiquetas asociadas a las cajas de texto son: Fecha, NoFactura, Idproducto, Precio Unitario, Cantidad y Precio Total.

Al cargar el formulario debe aparecer automáticamente la fecha actual del sistema

Combobox:

Cboproducto: Al cargar el formulario deben aparecer todos los productos que hay en la empresa. Al seleccionar un producto del combo debe aparecer automáticamente el id del producto y el precio unitario del mismo en sus respectivas cajas de texto las cuales serán de solo lectura.

Cbonombrecliente: Al cargar el formulario deben aparecer todos los clientes que tiene hasta ese momento la empresa.

Cbonombrevendedor: Al cargar el formulario deben aparecer todos los vendedores que trabajan en la empresa.

Botones de Radio: permiten seleccionar el tipo de pago que hará el cliente, si es de contado o de crédito.

Botones de pulsación

Botón Nuevo: Activa las cajas de texto para ingresar los detalles de factura de un determinado cliente.

Botón Nuevo Detalle: Realiza el detalle de un nuevo producto solicitado en una misma orden de pedido, es decir, que debe ser registrado en una misma factura, por tanto los atributos (Fecha, NoFactura, NombreCliente, NombreVendedor y TipoPago) deben quedar inactivos en ese momento, de tal manera que solo se haga la selección del nuevo producto y la cantidad a detallar.

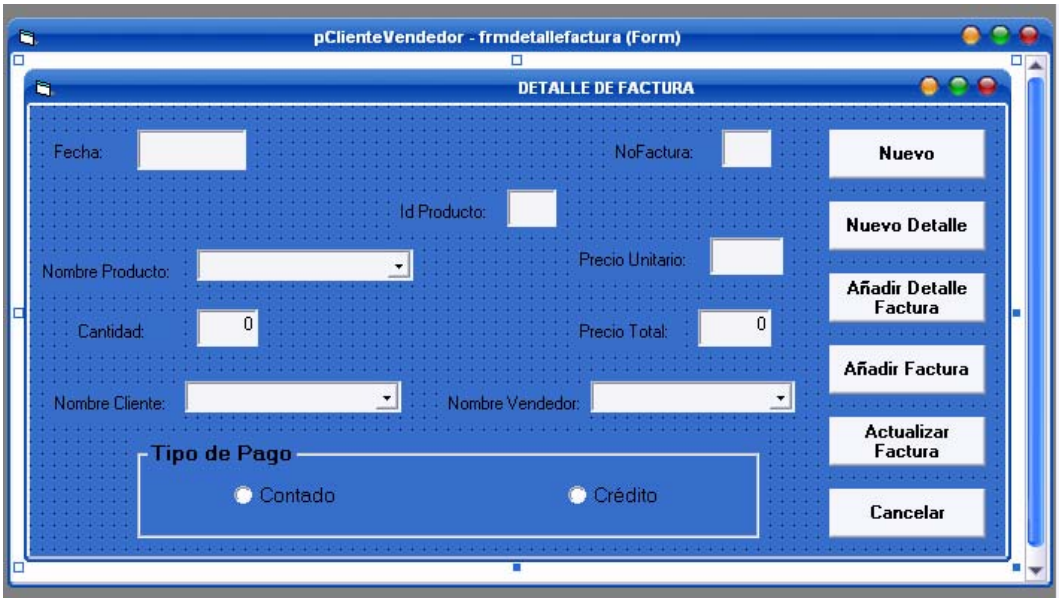
Botón Añadir Detalle Factura: Realiza el registro del detalle de un producto en la tabla DetalleFactura (solo los campos correspondientes a los atributos de la tabla).

Botón Añadir Factura: Realiza el registro de una nueva factura, es decir, cada vez que se haga clic en el botón nuevo se debe añadir el registro tanto a la tabla DetalleFactura como a la tabla Factura.

Botón Actualizar Factura: Realiza la actualización del monto de una determinada factura cada vez que se haga el detalle de un nuevo producto en ella misma.

Botón Cancelar: Realiza la cancelación de un nuevo detalle de factura dejando las cajas de texto vacías y solamente activo el botón **Nuevo**.

Nota: Activar y Desactivar botones de pulsación según la operación a realizar.



Resultado del diseño del formulario frmdetallefactura

Formulario frmfactura

El objetivo de este formulario es visualizar el total de facturas pendientes de pago y el monto total de dichas facturas de un cliente determinado.

- Utiliza como fuente de datos las tablas Cliente, DetalleFactura y Factura.
- Tendrá 3 cajas de texto, 1 comboBox. Las etiquetas asociadas a las cajas de texto son: Fecha, Facturas Pendientes de Pago y Monto Total.

ComboBox: Al cargar el formulario deben aparecer en el combo todos los clientes de la empresa y la fecha actual en su respectiva caja de texto. Al seleccionar uno de ellos debe aparecer automáticamente el total de facturas de texto.

Resultado del diseño del formulario frmfactura

Formulario frmpagodefactura

El objetivo de este formulario es registrar la cancelación de facturas pendientes de pago.

- Utiliza como fuente de datos la tabla Factura.
- Tendrá 1 caja de texto, 1 comboBox y 2 botones de pulsación (Aceptar y Salir). Las etiquetas asociadas al combo y a la caja de texto son: NoFactura y Total a pagar respectivamente.

ComboBox: Al cargar el formulario deben aparecer todos los números de facturas y al seleccionar uno de ellos debe aparecer automáticamente el monto correspondiente a esa factura en la caja de texto respectiva.

Botón Aceptar: Borrar de la tabla Factura y DetalleFactura los registros correspondientes al número de factura seleccionado del comboBox
Botón Salir: Cierra el formulario

Resultado del diseño del formulario frmpagodefactura

Formulario frmclientevendedor

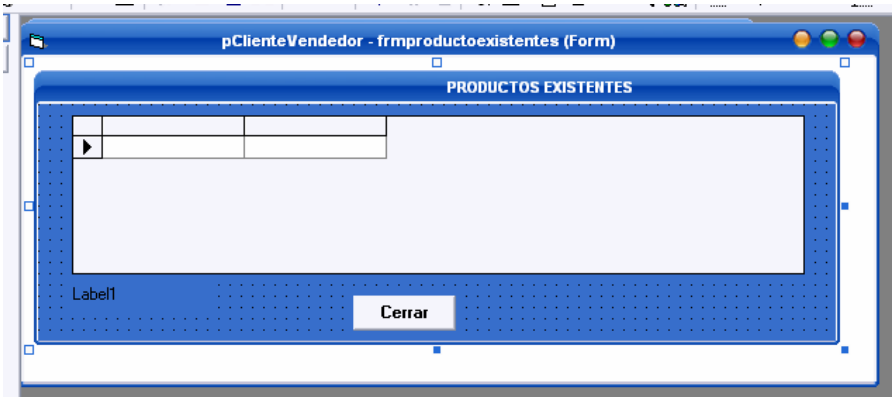
El objetivo de este formulario es mostrar cada uno de los formularios de la aplicación a través de botones de pulsación.

- Tendrá 6 botones de pulsación con los nombres de cada uno de los formularios creados anteriormente los cuales serán abiertos al dar clic en cada uno de ellos y una barra de menú con las siguientes opciones:

Menú: Archivo**, Submenú:** Salir
Menú: Ver, **Submenú:** Productos existentes, **Submenú:** Clientes existentes

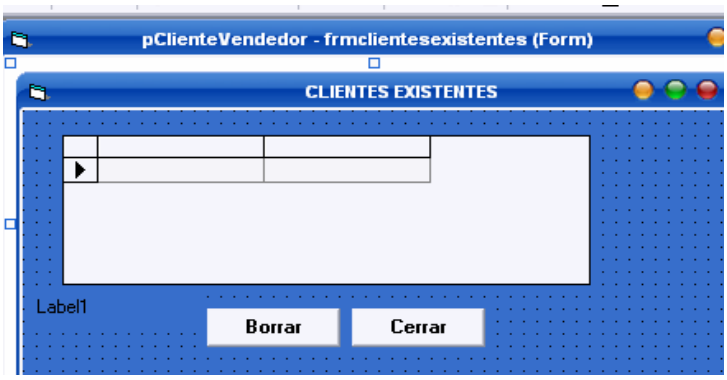
Submenú Salir: Cierra la aplicación

Submenú Productos existentes: Abre un formulario en el cual se debe mostrar a través de un DataGrid todos los productos que ofrece la empresa. El formulario tendrá un botón de pulsación que permitirá cerrar el formulario y una etiqueta en la que debe aparecer el total de productos que hay en la empresa.



Resultado del diseño del formulario frmproductosexistentes

Submenú Clientes existentes: Abre un formulario en el cual se debe mostrar a través de un DataGrid todos los clientes que tiene la empresa. El formulario tendrá dos botones de pulsación (Borrar y Cerrar) y una etiqueta en la que debe aparecer el total de clientes que tiene la empresa.



Resultado del diseño del formulario frmclientesexistentes

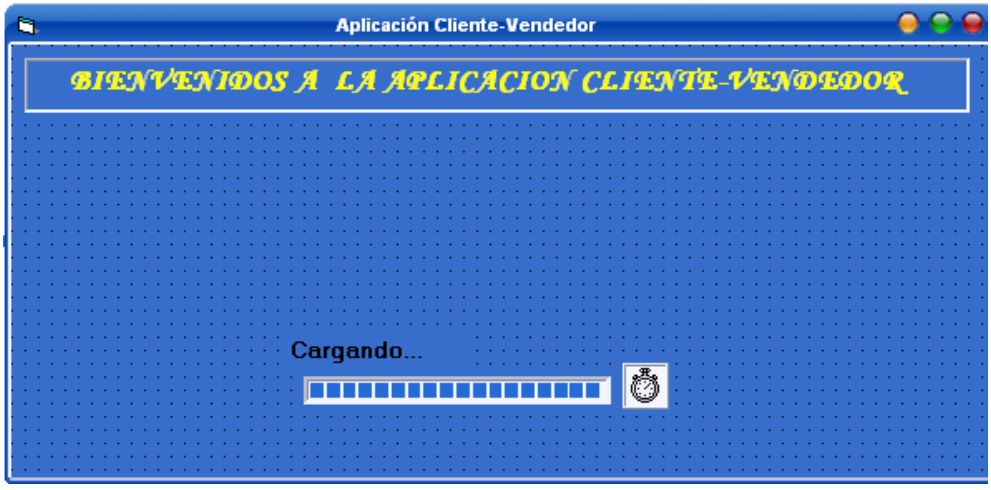
Botón Borrar: Al seleccionar un registro del DataGrid debe aparecer un mensaje preguntando si se desea borrar el registro seleccionado, en caso afirmativo lo borrara del DataGrid y de la base de datos y emitirá otro mensaje informando que el registro ha sido eliminado.

Nota: Ambos DataGrid deben permitir editar un registro y actualizar la base de datos con dicha modificación. Verifique sus propiedades para lograr esta funcionalidad.

Resultado del diseño del formulario frmclientevendedor

Formulario frmprincipal

El objetivo de este formulario es presentar una pantalla de bienvenida a la aplicación Cliente-Vendedor, se deja a creatividad del estudiante el diseño de esta interfaz. A modo de ejemplo presentamos nuestra pantalla:



Resultado del diseño del formulario frmprincipal

Practica 6.2: Como elaborar Informes con DataReport

Objetivo: El objetivo de esta práctica es que el estudiante conozca y aprenda a manejar DataReport para la generación de informes accediendo a una base de datos MySQL.

1. Introducción

Los informes son herramientas utilizadas por el programador de la Base de Datos para brindar al usuario final datos en formato impreso. Los informes muestran la información desde listas simples de registros hasta agrupaciones con cálculos complejos sobre cada grupo de datos.
DataReport permite crear informes basados en tablas o consultas, si la fuente de datos se encuentra almacenada en una tabla del sistema puede utilizar como fuente de datos la misma tabla; si los datos que desea mostrar se encuentran en distintas tablas cree una consulta con la información necesaria y dele formato al informe para mostrar la información dispuesta a su voluntad.

2. **Como crear un informe con DataReport**

Para crear un informe con esta herramienta es necesario enlazar nuestro Data Report con un Data Environment, el cual nos permitirá configurar nuestro controlador ODBC para poder acceder a la base de datos. Los pasos para realizar dicho enlace y configuración del controlador se muestran a continuación:

En la barra de menú de clic en proyecto y escoja la opción **Agregar Data Report**, luego la opción **Agregar Data Environment**. Una vez hecho esto aparecerá en el explorador de proyectos los iconos del DataReport y del Data Environment.

A continuación en el espacio de trabajo del Data Environment dé clic derecho sobre **Connection1** y seleccione la opción **Propiedades**, luego seleccione el proveedor **Microsoft OLE DB provider for ODBC Drivers** y clic en siguiente, luego en la opción **Usar el nombre de origen de datos** escoja el driver ODBC que haya configurado (en nuestro caso myodbc) y de clic en **Aceptar**.

En el explorador de proyectos de doble clic sobre el icono Data Environment, luego clic derecho en Connection1 y escoja Agregar Comando, aparecerá un icono llamado Command1, de clic derecho sobre el y aparecerá un cuadro de dialogo en el que puede dar nombre al comando (opcional) y debe seleccionar el origen de datos (puede ser un objeto de base de datos o Instrucción SQL), y por ultimo clic en Aceptar.

A continuación en la hoja de propiedades del Data Report, ubíquese en la propiedad DataSource y escoja Data Environment, luego en la propiedad Data Member escoja el nombre del comando creado en el Data Environment. Si no aparece alguno de estos componentes verifique si la conexión anterior se realizo correctamente.

Con esto ya tenemos configurado y enlazado nuestro Data Report con la base de datos de la cual queremos obtener información para el informe.

1.1 Familiarización con el entorno

Algunos ejemplos de informes a partir de los datos de la aplicación ClienteVendedor

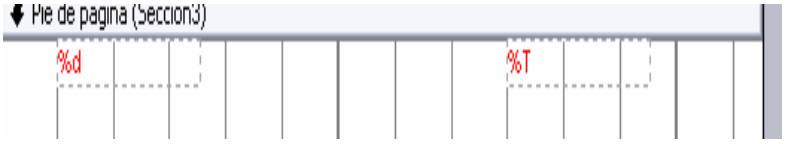
Listado de todos los Clientes

1. Crear un Data Report y enlazarlo con un Data Environment como se explico anteriormente. Ponga como **Objeto de base de datos**: tabla y en **Nombre del objeto** seleccione cliente.

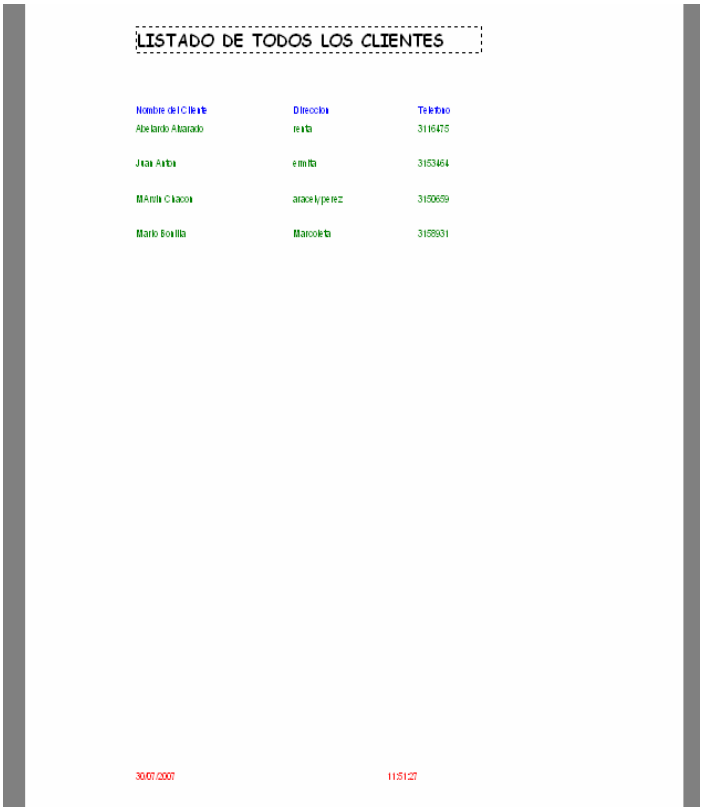
2. Una vez hecho el enlace, en la sección *encabezado de informe* del Data Report ponga el titulo del informe: **Listado de todos los Clientes**, en la sección *Encabezado de pagina* los encabezados: **Nombre del Cliente, Dirección y Teléfono** (Utilizando Etiquetas).

3. En la sección Detalle de clic derecho y escoja la opción **insertar control,** seleccione **cuadro de texto** y en sus propiedades ubíquese en la propiedad Data Member y escoja el comando creado en la configuración del Data Environment realizada en el paso 1, luego en Data Field aparecerán todos los campos de la tabla Clientes, escoja uno de ellos y repita lo mismo para los demás hasta obtener el siguiente diseño:

4. Para finalizar en la sección *pié de página* vamos a agregar la hora y fecha actual. Para realizar esto de doble clic sobre esta sección y escoja la opción insertar control, luego seleccione el formato en el que desea visualizar la fecha y hora actual (en este caso seleccione las opciones: (Fecha actual (formato largo) y Hora Actual (formato largo)). Al ejecutar la aplicación y generar dicho informe aparecerá la fecha y hora actual del sistema.



5. De formato al informe y ejecute la aplicación para observar los resultados. Deben obtener algo como la siguiente figura:



Consolidados de facturas por producto y monto total ordenados por Id de Producto.

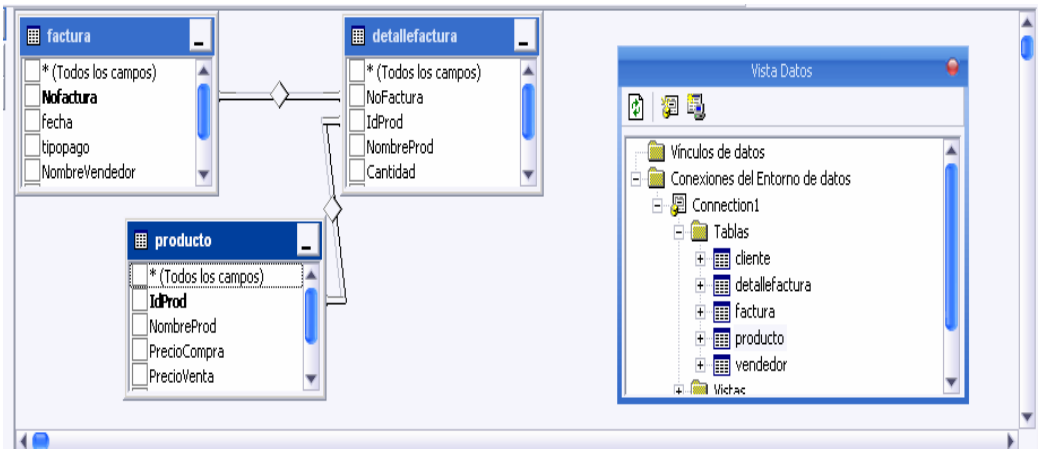
El informe para este tipo de consulta requiere el acceso a tres tablas (factura, detallefactura y producto), por tanto es necesario crear una consulta SQL para obtener la información requerida.

La consulta para este informe seria:

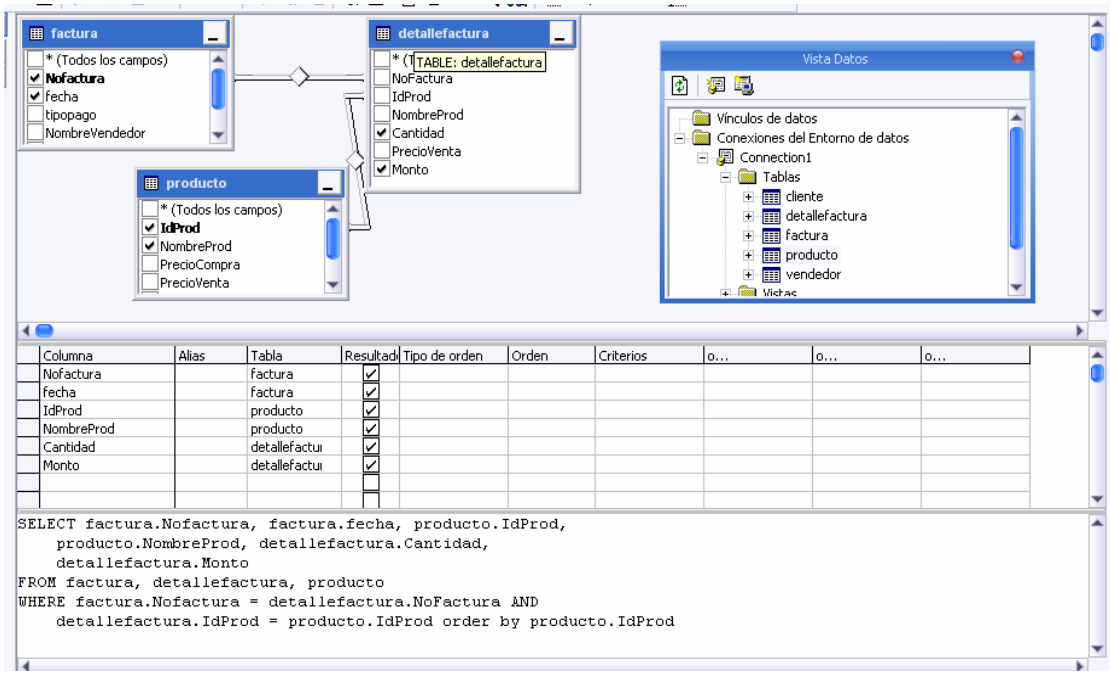
```
SELECT factura.Nofactura, factura.fecha, producto.IdProd, producto.NombreProd, detallefactura.Cantidad, detallefactura.Monto FROM factura, detallefactura, producto WHERE factura.Nofactura = detallefactura.NoFactura AND detallefactura.IdProd = producto.IdProd ORDER BY producto.IdProd
```

1 Crear un Data Environment, configurar el controlador ODBC y en el explorador de proyectos de doble clic sobre Data Environment, luego clic derecho en Connection1 y agregue un nuevo comando, luego dé nombre si lo desea al comando y en la opción **origen de datos** seleccione Instrucción SQL y de clic en el botón **Generador SQL**.

2 A continuación aparecerá una nueva pantalla y una nueva lista llamada Vista Datos en la cual aparecen nuestras conexiones y listados como si fuera un explorador. En el listado de conexiones aparece **Connection1** establecida cuando se creo el Data Environment, dé clic sobre el signo + y se desplegaran algunas carpetas, abra la carpeta Tablas, en esta deberían aparecer todas las tablas de la base de datos que se esta utilizando. Ahora solo hay que arrastrar a la pantalla aquellas tablas que forman parte de la consulta SQL (en este caso arrastre factura, detallefactura y producto). Como puede observar aparecen las tablas relacionadas por el atributo que tienen en común.



3 Ahora solo hay que marcar en cada una de las tablas aquellos campos que se desean que aparezcan en el informe, observe como abajo se va completando la consulta requerida, solo hay que agregar el orden por el cual queremos que parezcan los resultados, en este caso completamos la consulta poniendo ORDER BY producto.IdProd y damos clic en aceptar.



4 Nuestro comando ahora debería tener los 6 campos que hemos seleccionado en la consulta SQL, aun nos queda un paso más y es agrupar la consulta. Para ello abriremos las propiedades del comando y nos iremos a la pestaña **Agrupar**, luego activar la casilla agrupar comando y dejar el nombre por defecto del comando de agrupación y seleccionar el campo por el cual se hará la agrupación (**NoFactura).**

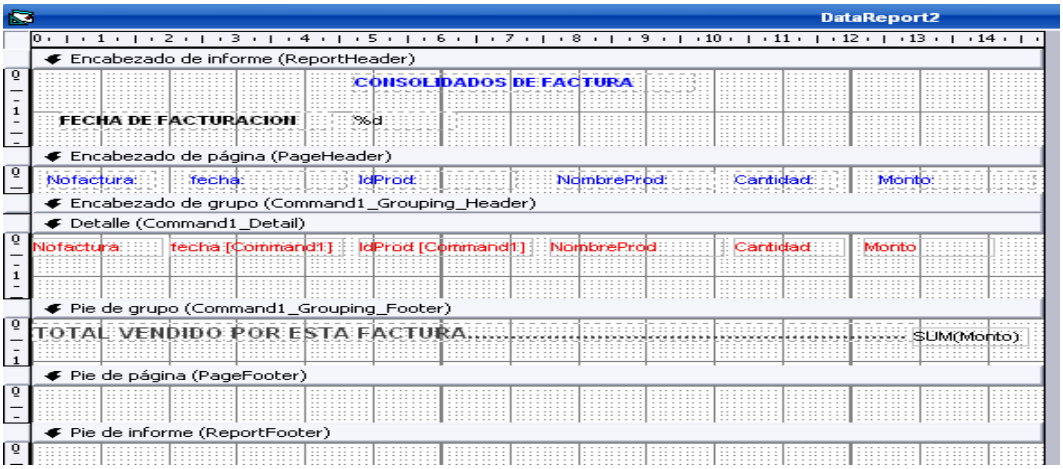
5 Una vez seleccionado el campo de agrupación, aceptamos y observamos como nuestra consulta ha cambiado, conteniendo ahora dos carpetas. La primera contiene el campo de agrupación y la segunda los campos que se mostrarán en la sección de detalle. Es hora ya de comenzar a diseñar nuestro informe, para ello tenemos que añadir a nuestro proyecto un nuevo DataReport, su propiedad *DataSource* la fijaremos al Data Environment creado y la propiedad *DataMember* tendremos que seleccionar el comando de agrupación creado anteriormente.

6 Pasaremos a colocar los campos en el informe, en la sección **Encabezado de informe** coloque el siguiente texto: CONSOLIDADOS DE FACTURAS, en la sección **Detalle** arrastre los campos que aparecen en una de las carpetas que se creo en el paso 5 al agrupar la consulta. Observe que se añade tanto el campo como su etiqueta, entonces arrastre las etiquetas de todos ellos a la sección **Encabezado de Pagina,** en la sección **Pie de Grupo** inserte una etiqueta con el texto TOTAL VENDIDO EN ESTA FACTURA……

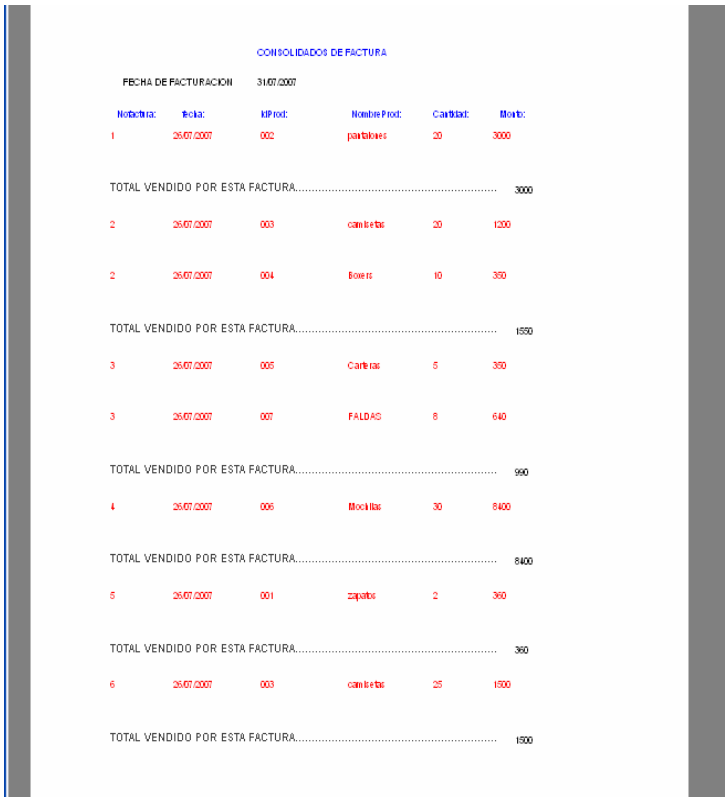
7 Luego para sumar todos los montos por números de factura dé clic derecho sobre la sección Pie de Grupo e inserte a la par de la etiqueta una función, (por defecto aparece la función suma), luego en las propiedades de esta función seleccione la propiedad DataMember en la cual aparecerá el comando creado, selecciónelo y luego en la propiedad DataField aparecerán todos los campos de la consulta, entonces escoja el campo Monto.

8 De formato al informe según su creatividad.

El diseño del informe debe quedar parecido o igual a la siguiente figura



9 Ejecute la aplicación y observe si los resultados son los esperados. Un ejemplo de una salida podría ser como se muestra en la siguiente figura:



CONSOLIDADOS DE FACTURA

FECHA DE FACTURACION 31/07/2007

NoFactura:	fecha:	IdProd:	NombreProd:	Cantidad:	Monto:
1	26/07/2007	002	pantalones	20	3000
TOTAL VENDIDO POR ESTA FACTURA.....					3000
2	26/07/2007	003	camisetas	20	1200
2	26/07/2007	004	Boxers	10	350
TOTAL VENDIDO POR ESTA FACTURA.....					1550
3	26/07/2007	005	Carteras	5	350
3	26/07/2007	007	FALDAS	8	640
TOTAL VENDIDO POR ESTA FACTURA.....					990
4	26/07/2007	006	Mochilas	30	8400
TOTAL VENDIDO POR ESTA FACTURA.....					8400
5	26/07/2007	001	zapatos	2	360
TOTAL VENDIDO POR ESTA FACTURA.....					360
6	26/07/2007	003	camisetas	25	1500
TOTAL VENDIDO POR ESTA FACTURA.....					1500

Ejercicios:

1. Genere un informe de las facturas que han sido facturadas como crédito por nombre de cliente y con las fechas y montos que debe cada cliente, también debe ir incorporada la dirección y el teléfono del cliente.

2. Genere un informe que refleje el monto total de ventas realizadas por cada vendedor hasta la fecha de ejecución del informe

3. Crear un informe con el nombre de los productos cuya existencia no sea cero y la ganancia de dicho producto supere los C$ 20

4. Genere un informe con la antigüedad de cada empleado (vendedor).

X. CONCLUSION

Al finalizar con este trabajo monográfico, consideramos que hemos logrado cumplir con los objetivos propuestos, al proporcionar al lector un documento que le brinde nociones básicas sobre cómo trabajar con las bases de datos bajo una plataforma diferente y que servirá como base para el desarrollo de futuros trabajos monográficos en esta área.

Al llevar a cabo la elaboración de cada una de las guías podemos concluir que:

- La tarea del docente es proporcionar al estudiante las herramientas básicas para el afianzamiento de los conocimientos adquiridos en las clases teóricas.

- La cantidad de horas asignadas a las clases prácticas de laboratorio no es suficiente.

- Existen herramientas muy buenas que nos ayudan a desarrollar aplicaciones con acceso a bases de datos que corren bajo un sistema operativo distinto al que se ha venido utilizando.

- Gracias a las herramientas utilizadas en el desarrollo de las guías se pudo ofrecer al estudiante ejercicios acorde al nivel de enseñanza impartido por el docente y concluir con este trabajo monográfico

XI. RECOMENDACIÓN

Debido a las características, enfoque y metodología empleada en el desarrollo de cada una de las guías de laboratorio, planteamos las siguientes recomendaciones:

- Adquirir y leer el libro y manual básico del entorno de desarrollo (Gambas), publicado en la Web donde se encuentran las practicas.

- Visitar las páginas Web citadas como referencia en este trabajo monográfico.

- Aprovechar otras herramientas del entorno que debido al nivel de aprendizaje en el que se encuentran los alumnos de tercer año no se pudieron aprovechar.

- Realizar las practicas de Windows en Visual Basic .Net

- Buscar información adicional a la proporcionada en este trabajo.

- Dedicar tiempo extra a las horas clase de laboratorio para la realización de las guías.

XII. BIBLIOGRAFIA

Silberschatz, Abraham “Fundamentos de Bases de Datos”. Tercera Edición.

Ceballos, Francisco Javier Enciclopedia de Visual Basic.

Referencias a Internet

Sitio oficial de Ubuntu con enlaces a otros sitios de soporte:
http://www.ubuntu.com/.

Pagina oficial de Gambas: http://www.gambassourceforge.net

Manual de referencia de MySQL.: http://dev.mysql.com/doc/.

www.mailxmail.com/curso/informatica/datareport.

Versión escaneada del libro de Gambas

http://gambas.gnulinex.org/libro/

Memorias de un aprendiz de Gambas

http://www.kdehispano.org/node/1109

XIII. ANEXOS

Acrónimos

SQL (Structured Query Language): Lenguaje de consulta estructurada.

SGBD: Sistema Gestor de Base de Datos.

LDD: Lenguaje de definición de datos.

LMD: Lenguaje de manipulación de datos.

GAMBAS (Gambas almost means Basic): Gambas casi significa Basic.

API: Interfaz para programar la aplicación.

ODBC: Controlador de acceso a bases de datos

IDE: Entorno de Desarrollo

DIA: Editor de diagramas

E-R: Entidad-Relacion

GPL: General Public Licence

KDE: Escritorio de desarrollo en Linux

TEMPORIZACIÓN

Cada semestre consta de 14 semanas lectivas. La asignatura tiene una frecuencia de 6 horas semanales (4 de teoría y 2 de prácticas de laboratorio), resultando en un total 56 horas teóricas y 28 horas prácticas.

Práctica #	No. de Horas
0	2
1	2
2	2
3 y 3.1	4
4 y 4.1	4
5 y 5.1	6
5.2	2
6 y 6.1	4
6.2	2
Total	**28**

PROPUESTA DE EVALUACIÓN DE LAS GUÍAS

Evaluación cualitativa:

- Entrega de algunas guías propuestas.
- Aplicación de pequeñas pruebas
- Defensa de algunas aplicaciones propuestas.

Evaluación Cuantitativa:

- La evaluación de la parte práctica representa el 40% de la nota final de cada parcial. Por tanto se propone que la aplicación de las evaluaciones previa al examen parcial práctico represente el 15% de modo que el alumno tenga un acumulado al presentar dicho examen.

VIVENCIAS ADQUIRIDAS

Experiencias:

- Experimentación del trabajo del docente
- Comparación de metodologías.

Logros:

- Consolidación de conocimientos adquiridos en la asignatura.
- Aprender a trabajar con MySQL como DBMS.
- Capacidad de desarrollar aplicaciones utilizando un software diferente.
- Trabajo en equipo

Sugerencias:

- Impartir algunos cursos o seminarios necesarios en el aprendizaje y asimilación de conocimientos de determinada asignatura.
- Homologar la metodología empleada en el diseño de las prácticas de laboratorio de cada una de las asignaturas.
- Separación de la evaluación de las clases teóricas y clases prácticas.