BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

SISTAR

Sistema de Administración de refacciones

TESIS

Que para obtener el título de:

INGENIERO EN CS. DE LA COMPUTACIÓN

Presenta:

MISAEL CONTRERAS VÁZQUEZ

Asesora:

M.C. CARMEN CERÓN GARNICA

Noviembre de 2008

“Cuando emprendas alguna obra
siempre habrá gente que intentará detenerte,
cuando vea que no puede detenerte
te dirá como debes hacer las cosas,
al final, cuando lo logres te dirán
que siempre creyeron en ti”

- John C. Maxwell

A mis Padres

... Porque desde el principio creyeron en mi

ÍNDICE

INTRODUCCIÓN

Se dice que “La información es el recurso más importante de una empresa” y nunca esa frase fue tan acertada como hoy. En el mercado actual la empresa mejor informada es la que brinda mejores servicios, mejor atención al cliente, maximiza ganancias y minimiza pérdidas. El objetivo de desarrollar un sistema computacional de administración para negocios es de brindar la información necesaria y en el tiempo requerido para lograr tal nivel de competencia de la empresa.

Una refaccionaria no es la excepción, la enorme cantidad de piezas distintas que es ofrecida al cliente hace difícil manejar correctamente los precios y el inventario. Si a esto anexamos que el manejo tanto de la información de las compras o de las ventas se realiza de manera manual, tenemos entonces un manejo algo impreciso de la información, ya que no siempre llevan de manera actualizada el control de los ingresos y egresos.

Uno de los problemas que se deriva de esto es que no existe algún tipo de control de almacenamiento. Es por ello, que en muchos casos la Refaccionaria queda sin abastecimiento de alguna pieza ó en otros casos tiene en gran cantidad otras tipos de piezas.

Con tantas piezas que se venden en la Refaccionaria se pierde el control manual para su abastecimiento correcto, y por ende no se toma una decisión correcta del periodo de compras. Esto también ocasiona que los clientes de la Refaccionaria en varias ocasiones no encuentran lo que se necesita y trayendo consigo pérdidas económicas sustanciales.

Es por ello que el objetivo general es elaborar un Sistema capaz de automatizar el proceso de compra y venta en una Refaccionaria, teniendo en cuenta Clientes Potenciales, Clientes Ocasionales y Distribuidores. Además, específicamente haremos lo siguiente:

- Analizar la problemática de la administración de la refaccionaria para proponer una estrategia de solución.
- Aplicar los conocimientos obtenidos en la implementación y uso de las Bases de Datos, Programación e Ingeniería de software para el desarrollo del sistema.
- Diseñar las formas de pantalla de manera que obtengamos una presentación organizada a cada uno de los procesos.

Al tener una mejor administración del inventario y ventas, la Empresa *Bicicletas y Refacciones de Tecamachalco* pretende mejorar la calidad de atención al cliente, estando siempre abastecida de todos los productos, consultar rápidamente el precio de cada artículo y estimar ganancias netas para próximas inversiones. Todo esto conllevará a un aumento en las ganancias de la empresa.

El contenido de esta tesis se presenta de la siguiente manera:

En el *capítulo 1* se describen los conceptos básicos para generar un Sistema de Bases de Datos, así mismo se proporciona un panorama breve sobre las herramientas de desarrollo de software de las que se dispone para llevar a cabo el análisis, desarrollo e implementación de una solución a la problemática tratada.

El *Capítulo2* plantea el análisis del sistema, la estrategia de solución y se especifican los requerimientos que debe satisfacer la misma.

En el *Capítulo 3* se presenta el diseño del sistema donde se especifica el diseño conceptual, lógico y físico.

En el *Capítulo 4,* se muestra la implantación de sistema y las pruebas realizadas.

Finalmente se presentan los siguientes puntos:
Conclusiones
Referencias
Anexos:
Anexo A. Código del sistema

Capitulo 1

1 Bases de Datos y Herramientas de desarrollo de software

CAPITULO 1.
Bases de Datos y herramientas de Desarrollo de Software

El objetivo de este capítulo es cubrir los conceptos básicos de la disciplina del desarrollo de software, de la metodología UML, de los sistemas de bases de datos y la programación en el Entorno de Desarrollo Delphi® de Borland con el fin de familiarizar al lector con los diversos términos técnicos de los que se hacen uso en los próximos capítulos.

1.1 Sistemas de Información

Un sistema de información es un sistema, automatizado o manual, que abarca personas, máquinas, y/o métodos organizados de recolección de datos, procesamiento, transmisión y diseminación de datos que representa información para el usuario. En otras palabras, es un conjunto de elementos que interactúan entre sí con el fin de apoyar las actividades de una empresa o negocio. Desde el punto de vista Informático, un sistema de información es una aplicación orientada a los negocios, la cual se conforma de una (o varias) base(s) de datos, software de aplicación y un manual. También abarca los procedimientos de máquina y el sistema operativo que hace el procesamiento de los datos.[5]

1.1.1 Historia

El estudio de los sistemas de información se originó como una sub-disciplina de las ciencias de la computación en un intento por entender y racionalizar la administración de la tecnología dentro de las organizaciones. Los sistemas de información han madurado hasta convertirse en un campo de estudios superiores dentro de la administración. Adicionalmente, cada día se enfatiza más como un área importante dentro de la investigación en los estudios de administración, y es enseñado en las universidades y escuelas de negocios más grandes en todo el mundo.

En la actualidad, la Información y la tecnología de la Información forman parte de los cinco recursos con los que los ejecutivos crean y/o modelan una organización, junto con el personal, dinero, material y maquinaria.[12] Muchas compañías han creado la posición de Director de Información (CIO, por sus siglas en inglés Chief Information Officer) quien asiste al comité ejecutivo de la compañía, junto con el Director Ejecutivo, el Director Financiero, el Director de Operaciones y el Director de Tecnología (es común que el Director de Información actúe como Director de Tecnología y viceversa).

1.1.2 Clasificación

Curiosamente, la manera en que se clasifican los sistemas de información obedece al tiempo en el que estos fueron apareciendo, y no tanto por la función a la que vayan destinados o el tipo de usuario final del mismo. Los SI pueden clasificarse en:

Sistema de procesamiento de transacciones (TPS).- Gestiona la información referente a las transacciones producidas en una empresa u organización.

Sistemas de información gerencial (MIS).- Orientados a solucionar problemas empresariales en general.

Sistemas de soporte a decisiones (DSS).- Herramienta para realizar el análisis de las diferentes variables de negocio con la finalidad de apoyar el proceso de toma de decisiones.

Sistemas de información ejecutiva (EIS).- Herramienta orientada a usuarios de nivel gerencial, que permite monitorizar el estado de las variables de un área o unidad de la empresa a partir de información interna y externa a la misma.

Sistemas de automatización de oficinas (OAS).- Aplicaciones destinadas a ayudar al trabajo diario del administrativo de una empresa u organización.

Sistema experto (SE).- Emulan el comportamiento de un experto en un dominio concreto.

Estos sistemas de información no surgieron simultáneamente en el mercado; los primeros en aparecer fueron los TPS, en la década de los 60, y los últimos fueron los SE, que alcanzaron su auge en los 90 (aunque estos últimos tuvieron una tímida aparición en los 70 que no cuajó, ya que la tecnología no estaba suficientemente desarrollada). [14]

En la siguiente gráfica podemos apreciar la evolución de los sistemas de información, comenzando desde su aparición en los 60's.

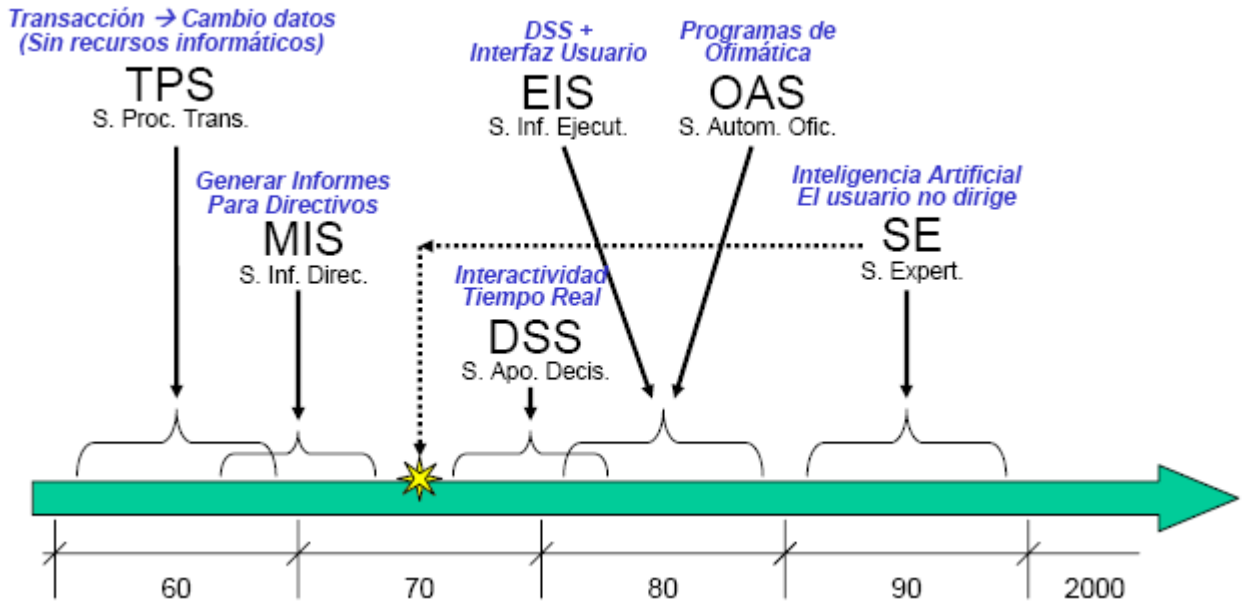


Fig. 1.1 – Evolución de los sistemas de información

También podemos clasificarlos acorde al entorno de aplicación. Esto es, por un entorno transaccional, que consiste en captar, manipular y almacenar los datos. Por tanto, lo importante es qué datos se modifican y cómo.

El segundo rubro de esta clasificación es el Entorno de Decisiones. En el cual, lo más importante es presentar información útil a la empresa en la toma de decisiones.

Los sistemas de información tratan el desarrollo, uso y administración de la infraestructura de la tecnología de la información en una organización.

En la era post-industrial, la era de la información, el enfoque de las compañías ha cambiado de la orientación hacia el producto a la orientación hacia el conocimiento, en este sentido el mercado compite hoy en día en términos del proceso y la innovación, en lugar del producto. El énfasis ha cambiado de la calidad y cantidad de producción hacia el proceso de producción en sí mismo, y los servicios que acompañan este proceso.

El mayor de los activos de una compañía hoy en día es su información, representada en su personal, experiencia, conocimiento, innovaciones (patentes, derechos de autor, secreto comercial). Para poder competir, las organizaciones deben poseer una fuerte infraestructura de información, en cuyo corazón se sitúa la infraestructura de la tecnología de información. De tal manera que el sistema de información se centre en estudiar las formas para mejorar el uso de la tecnología que soporta el flujo de información dentro de la organización.

1.2 Ingeniería de Software

> *Que el «software» es un elemento básico en nuestra sociedad actual como generador de servicios parece un hecho evidente. Que sus costes de desarrollo (y, en muchos casos, de adquisición) sean cada vez más altos respecto al hardware sobre el que se ejecuta también es evidente aunque nos resistamos a aceptarlo en nuestra práctica cotidiana. Que un sistema de software envejece sin necesidad de estropearse (haciéndose inútil en un contexto dado) es un hecho al que nos hemos acostumbrado a pesar de ser una gran paradoja en productos sin partes mecánicas y con un coste de replicación prácticamente nulo. Basta echar una ojeada a nuestro alrededor para ver cómo estos sistemas de software están teniendo una importancia creciente, responsabilizándose de los éxitos y fracasos de muchos sistemas basados en ellos y siendo también responsables de los éxitos y fracasos de las empresas que los construyen o utilizan. León (1996)* [7]

Según la definición del IEEE, citada por Lewis (1994), la ingeniería de software es la suma total de los programas de computadora, procedimientos, reglas, la documentación asociada y los

datos que pertenecen a un sistema de cómputo[8] y según el mismo autor un producto de software es un producto diseñado para un usuario específico. En este contexto, la Ingeniería de software es un enfoque sistemático del desarrollo, operación, mantenimiento y retiro del software, que en palabras más llanas, se considera que la Ingeniería de Software es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones costo-efectivas (eficaces en costo o económicas) a los problemas de desarrollo de software, es decir permite elaborar consistentemente productos correctos, utilizables y costo-efectivos. [3]

Jacobson (1998) define el proceso de ingeniería de software como "un conjunto de etapas parcialmente ordenadas con la intención de lograr un objetivo, en este caso, la obtención de un producto de software de calidad". El proceso de desarrollo de software "es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo". Concretamente define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo. [6]

El proceso de desarrollo de software requiere por un lado un conjunto de conceptos, una metodología y un lenguaje propio. A este proceso también se le llama el ciclo de vida del software que comprende cuatro grandes fases: concepción, elaboración, construcción y transición. La concepción define el alcance del proyecto y desarrolla un caso de negocio. La elaboración define un plan del proyecto, especifica las características y fundamenta la arquitectura. La construcción crea el producto y la transición transfiere el producto a los usuarios.

Actualmente se encuentra en una etapa de madurez el enfoque Orientado a Objetos (OO) como paradigma del desarrollo de sistemas de información. El Object Management Group (OMG) es un consorcio a nivel internacional que integra a los principales representantes de la industria de la tecnología de información OO. El OMG tiene como objetivo central la promoción, fortalecimiento e impulso de la industria OO. El OMG propone y adopta por consenso especificaciones entorno a la tecnología OO. Una de las especificaciones más importantes es la adopción en 1998 del Lenguaje de Modelado Unificado o UML (del inglés Unified Modeling Language) como un estándar, que junto con el Proceso Unificado (RUP) están consolidando la tecnología OO.

1.2.1 El Proceso Unificado de Desarrollo de Software.

El Proceso Unificado Racional (Rational Unified Process en inglés, habitualmente resumido como RUP) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

También se conoce por este nombre al software desarrollado por Rational, hoy propiedad de IBM, el cual incluye información entrelazada de diversos artefactos y descripciones de las diversas actividades. Está incluido en el Rational Method Composer (RMC), que permite la personalización de acuerdo a necesidades.

Originalmente se diseñó un proceso genérico y de dominio público, el Proceso Unificado, y una especificación más detallada, el Rational Unified Process, que se vendiera como producto independiente.

El origen del RUP proviene del modelo de espiral original de Barry Boehm. La Aproximación racional fue desarrollada por Rational Software entre 1980 y 1990. [6]

1.2.1.1 Diseño

Los creadores y desarrolladores del RUP se enfocaron en diagnosticar las características de diferentes proyectos de software que fracasaron. Haciendo esto, ellos trataron de reconocer la causa de estas fallas, y después buscaron en los procesos de ingeniería de software existentes una solución para los síntomas encontrados. Entre las causas de los fracasos (síntomas) de los proyectos analizados, las más comunes fueron:

- Comunicación imprecisa y ambigua.
- Arquitectura Débil. (Arquitectura que no trabaja apropiadamente bajo presión).
- Una complejidad abrumadora.
- Inconsistencias que no fueron detectadas en el análisis, diseño e implementación.
- Pruebas insuficientes.
- Una valoración subjetiva del estado del proyecto.
- No enfrentaron los riesgos de manera correcta.
- Una propagación sin control de cambios en el proyecto.
- No automatizaron procesos lo suficiente.

Una falla en un proyecto es causada por la combinación de síntomas severos, aunque cada proyecto falla en una forma única. El resultado de este estudio fue un sistema de “buenos hábitos” al desarrollar software al que llamaron el “proceso unificado de desarrollo Racional.”

Este proceso fue diseñado con las mimas técnicas que el equipo investigador utilizaba para diseñar software, por consiguiente, el proceso está basado en un modelo orientado a objetos utilizando el Lenguaje unificado de Modelado (UML).

RUP está centrado en la arquitectura. La arquitectura de un sistema de software se describe mediante diferentes vistas del sistema en construcción y es una visa del diseño completo con todas las características más importantes resaltadas, dejando los detalles de lado. [10]

1.2.1.2 Los seis principios clave

RUP está basado en un conjunto de seis principios clave para el desarrollo del manejo de negocios, a saber:

1. Adaptar el proceso.
2. Balancear prioridades.
3. Colaboración entre diversos equipos.
4. Demostrar el valor de manera iterativa.
5. Elevar el nivel de abstracción.
6. Fijar un enfoque permanente sobre la calidad.

A este conjunto de principios comúnmente se le nombra ABCDEF.

Adaptar el proceso

El proceso deberá adaptarse a las características propias del proyecto u organización. El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

Balancear prioridades

Los requerimientos de los diversos inversores pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos.

Colaboración entre equipos

La ingeniería de software es un esfuerzo de equipo. Con una amplia variedad de riesgos, todas las voces necesitan ser escuchadas. Con la demanda de desarrollo globalmente distribuido, la comunicación está disponible a través de herramientas modernas de comunicación. La colaboración no debe darse únicamente en los requerimientos, también debe estar en control de cambios, resultados de pruebas, administración de liberación del producto y planes del proyecto. Esto es especialmente importante para los proyectos basados en RUP, los cuales son ejecutados en una aproximación incremental iterativa.

Demostrar valor iterativamente

Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados

Elevar el nivel de abstracción

Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL o esquemas (frameworks) por nombrar algunos. Esto previene a los ingenieros de software ir directamente de los requisitos a la codificación de software a la medida del cliente. Un nivel alto de abstracción también permite discusiones sobre diversos niveles arquitectónicos. Éstos se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con UML.

Enfocarse en la calidad

El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción. El aseguramiento de la calidad forma parte del proceso de desarrollo y no de un grupo independiente.

1.2.1.3 Ciclo de vida del proyecto

El ciclo de vida RUP es una implementación del Desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones.

El RUP divide el proceso de desarrollo en ciclos, teniendo un producto final al culminar cada una de ellos, estos a la vez se dividen en fases que finalizan con un hito donde se debe tomar una decisión importante:

Concepción: se hace un plan de fases, se identifican los principales casos de uso y se identifican los riesgos.

Elaboración: se hace un plan de proyecto, se completan los casos de uso y se eliminan los riesgos. Esta fase debe satisfacer la Arquitectura del Ciclo de Vida Milestone en los siguientes criterios:

- Un modelo de casos de uso en el cual, los casos de uso y los actores han sido identificados y la mayoría de las descripciones de los casos de uso han sido desarrolladas. El modelo de casos de uso debe estar completo en al menos un 80%.
- Una descripción de la arquitectura del software en un proceso de desarrollo de sistemas de software.
- Una arquitectura que ejecute significativamente los casos de uso.
- Una revisión de la lista de riesgos y el modelo del negocio.
- Un plan de desarrollo para todo el proyecto.
- Prototipos que prueben mitigar técnicamente cada riesgo identificado.

Si el proyecto no aprueba el modelo Milestone, aún hay tiempo para que el proyecto sea cancelado o rediseñado. Después de pasar esta fase, el proyecto pasa a una fase de alto riesgo donde los cambios son más difíciles y perjudiciales si se llevan a cabo.

Construcción: se concentra en la elaboración de un producto totalmente operativo y eficiente y el manual de usuario

Transición: se Instala el producto en el cliente y se entrena a los usuarios. Como consecuencia de esto suelen surgir nuevos requisitos a ser analizados.

Mantenimiento: una vez instalado el producto, el usuario realiza requerimientos de ajuste, esto se hace de acuerdo a solicitudes generadas como consecuencia del interactuar con el producto.

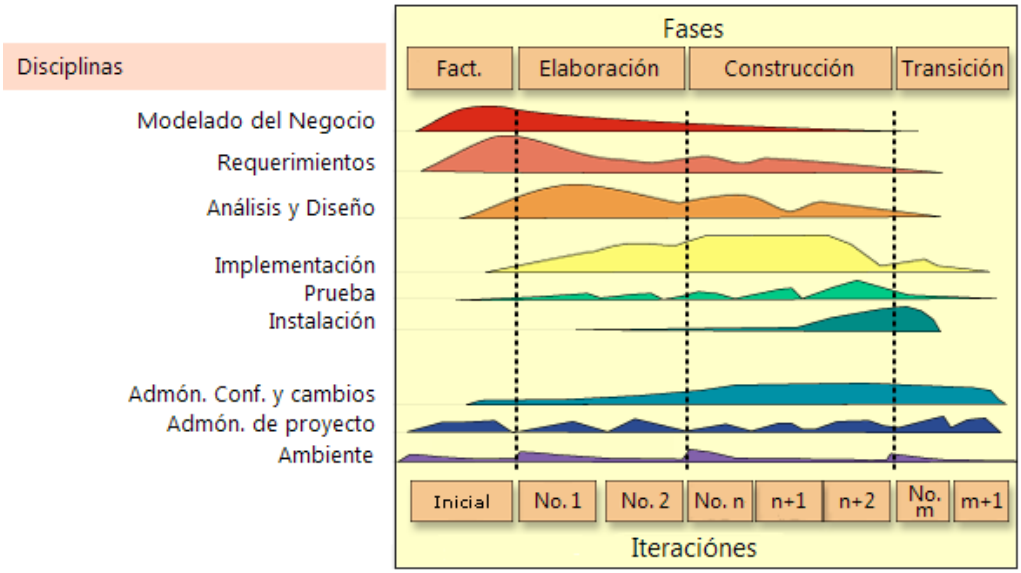
1.2.1.4 Disciplinas y Flujos de Trabajo.

RUP está basado en un conjunto de bloques de construcción que describen qué es lo que se debe producir, las habilidades necesarias y la explicación paso a paso describiendo como los objetivos específicos del desarrollo serán alcanzados. Los principales bloques de construcción (también llamados elementos de contenido) son los siguientes:

- Roles (Quien) – Un Rol define un conjunto de habilidades, competencias y responsabilidades.
- Productos de Trabajo (Qué) – Un producto de trabajo representa el resultado de una tarea incluyendo toda la documentación y modelos producidos en el proceso.
- Tareas (cómo) – Una tarea describe una unidad de trabajo asignada a un rola que provee un resultado significativo.

En cada iteración las tareas son categorizadas dentro de nueve disciplinas ubicadas en dos rubros:

- *Disciplinas de la ingeniería:* Modelado del negocio, requerimientos, análisis y diseño, implementación, pruebas e instalación.
- *Disciplinas de soporte*: Administración de configuración y cambios, Administración del proyecto y Ambiente.



Cada disciplina es un conjunto de actividades relacionadas (flujos de trabajo) vinculadas a un área específica dentro del proyecto total. Las importantes son: Requerimientos, Análisis, Diseño, codificación y prueba.

Cada disciplina está asociada con un conjunto de modelos que se desarrollan. Estos modelos están compuestos por artefactos. Los artefactos más importantes son los modelos que cada disciplina realiza: modelo de casos de uso, modelo de diseño, modelo de implementación y modelo de prueba. [10]

1.2.2 El leguaje unificado de modelado.

Todo gira en torno a una visión. Un sistema complejo toma forma cuando alguien tiene la visión de cómo la tecnología puede mejorar las cosas. Los desarrolladores tienen que entender completamente la idea y mantenerla en mente mientras crean el sistema que le dé forma.

El éxito de los proyectos de desarrollo de aplicaciones o sistemas se debe a que sirven como enlace entre quien tiene la idea y el desarrollador. El UML (Unifided Modeling Languaje, por sus siglas en inglés) es una herramienta que cumple con esta función, ya que le ayuda a capturar la idea de un sistema para comunicarla posteriormente a quien se está involucrando en su proceso de desarrollo; esto se lleva a cabo mediante un conjunto de símbolos y diagramas. Cada diagrama tiene fines distintos dentro del proceso de desarrollo. [15]

UML es, probablemente, una de las innovaciones conceptuales en el mundo tecnológico del desarrollo de software que más expectativas y entusiasmos haya generado en muchos años, comparable a la aparición e implantación de los lenguajes COBOL, BASIC, Pascal, C++ y más recientemente Java o XML. Además, todas las expectativas se han cumplido y han generado a su vez nuevas expectativas. UML es ya un estándar de la industria, pero no sólo de la industria del software sino, en general, de cualquier industria que requiera la construcción de modelos como condición previa para el diseño y posterior construcción de prototipos.

El lenguaje de modelado UML es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear configurar, mantener y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de la aplicación y medios. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado incorpora las mejores prácticas actuales en un acercamiento estándar. La especificación de UML no define un proceso estándar pero está pensado para ser útil en un proceso de desarrollo iterativo. [11]

UML no es un leguaje de programación. Las herramientas pueden ofrecer generadores de código de UML para una gran variedad de lenguajes de programación, así como construir modelos por ingeniería inversa a partir de programas existentes. UML tampoco es un lenguaje pensado para probar teoremas. UML es un lenguaje de modelado de propósito general.

UML no es un método de desarrollo. No dice cómo pasar del análisis al diseño y de este al código. No son una serie de pasos que nos lleven a producir código a partir de unas especificaciones.

UML al no ser un método de desarrollo es independiente del ciclo de desarrollo que se vaya a seguir, puede encajar en un tradicional ciclo en cascada, o en un evolutivo ciclo en espiral o incluso en los métodos ágiles de desarrollo.

1.2.2.1 Diagramas UML

El UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos, aunque no es parte de esta investigación presentar tales reglas.

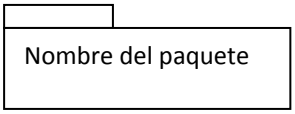
Los diagramas que son utilizados en UML son los diagramas de clases, objetos, casos de uso, estados, secuencias, actividades, colaboraciones, componentes y distribución. No es necesario utilizar en cada proyecto todos los diagramas.

Como se verá más adelante, los diagramas de UML le permiten examinar un sistema desde distintos puntos de vista. La mayoría de los modelos UML contienen un subconjunto de los diagramas indicados.

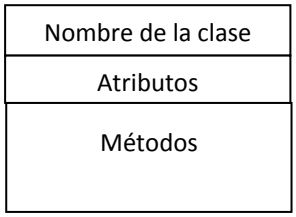
¿Por qué es necesario contar con diferentes perspectivas de un sistema? Por lo general, un sistema cuenta con diversas personas implicadas las cuales tienen enfoques particulares en diversos aspectos del sistema. El escrupuloso diseño de un sistema involucra todas las posibles perspectivas, y el diagrama UML le da una forma de incorporar una perspectiva en particular. El objetivo es satisfacer a cada persona implicada. [15]

1.2.2.2 Diagrama de clases

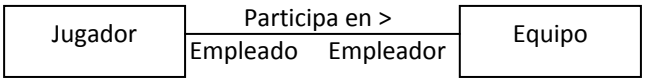
En UML, un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas informáticos, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.



Paquete: Forma en la que el UML organiza un diagrama de elementos



Clase. El nombre inicia con mayúscula, sin dejar espacios. Los atributos son propiedades o características de una clase y describe un rango de valores que la propiedad podrá contener. Los métodos indican la forma de acceder a los atributos de la clase.



Las asociaciones conectan dos clases y se especifica con una línea, además se indica la dirección con una flecha. Los papeles se representan bajo la línea de asociación.

1.2.2.3 Diagramas de casos de uso.

Una vista de casos de uso captura el comportamiento de un sistema, de un subsistema ó de una clase, tal como se muestra a un usuario exterior. Reparte la funcionalidad del sistema en transacciones significativas para los actores-usuarios ideales de un sistema. Las piezas de funcionalidad interactiva se llaman casos de uso. [11]

Imagínese al caso de uso como una colección de situaciones respecto al uso de un sistema. Cada escenario describe una secuencia de eventos. Cada secuencia se inicia por una persona, otro sistema, una parte del hardware o por el paso del tiempo. A las entidades que inician secuencias se les conoce como actores. EL resultado de la secuencia debe ser algo utilizable ya sea por el actor que la inició, o por otro actor. [15]

Así como el diagrama de clases es un buen medio para estimular a un cliente que hable respecto a un sistema desde su propio punto de vista, el caso de uso es una excelente herramienta para estimular a que los usuarios potenciales hablen, de un sistema, desde sus propios puntos de vista. No siempre es fácil para los usuarios explicar cómo pretenden utilizar un sistema. Puesto que el desarrollo tradicional de los sistemas era, con frecuencia, algo así como un a ciencia oculta, con muy poca información para los usuarios, a aquellos que osaban preguntar se les daba información muy poco explícita o ciertamente confusa respecto a lo que utilizarían.

La idea es involucrar a los usuarios en las etapas iníciales del análisis y diseño del sistema. Esto aumenta la probabilidad de que el sistema sea de mayor provecho para la gente a la que supuestamente ayudará, en lugar de ser un manojo de expresiones de computación incompresibles e inmanejables por los usuarios finales. En la figura 1.2 se muestra los elementos que conforman un diagrama de casos de uso.

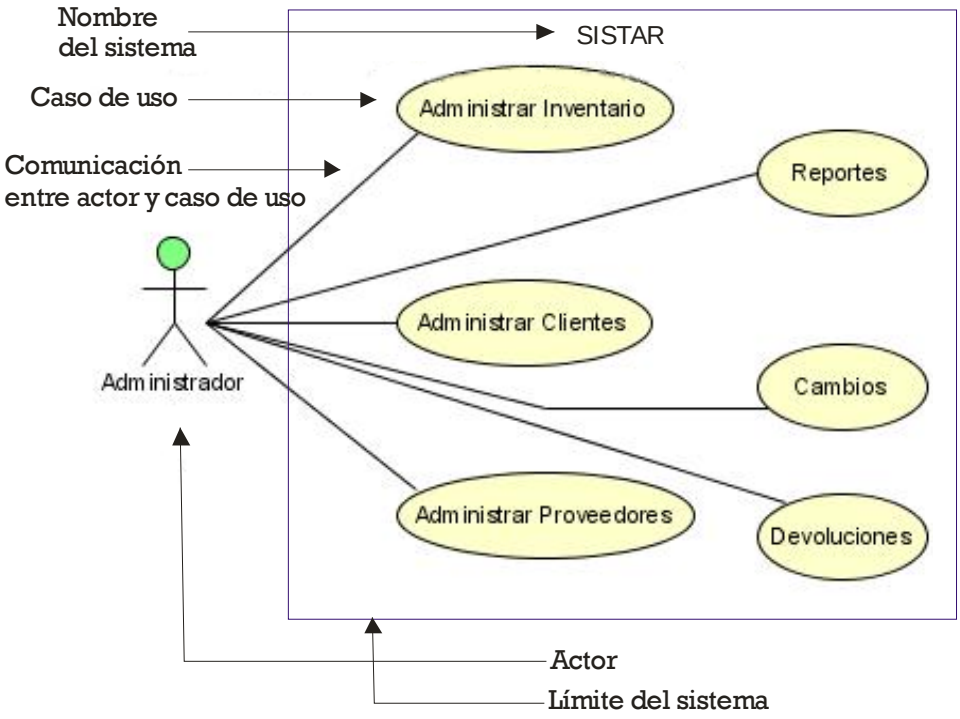


Fig. 1.2 – Diagrama de un caso de uso

1.2.2.4 Diagramas de secuencias.

El UML le permite expandir su campo de visión y le muestra la forma en que un objeto interacciona con otros. En este campo de visión expandido, incluirá una importante dimensión: el tiempo. La idea primordial es que las interacciones entre los objetos se realizan en una secuencia establecida y que la secuencia se toma su tiempo en ir del principio al fin. Al momento de crea un sistema tendrá que especificar la secuencia, y para ello, utilizará el diagrama correspondiente.

El diagrama de secuencias consta de objetos que se representan del modo usual: rectángulos con nombre (subrayado), mensajes representados por líneas continuas con una punta de flecha y el tiempo representado como una progresión vertical. [15]

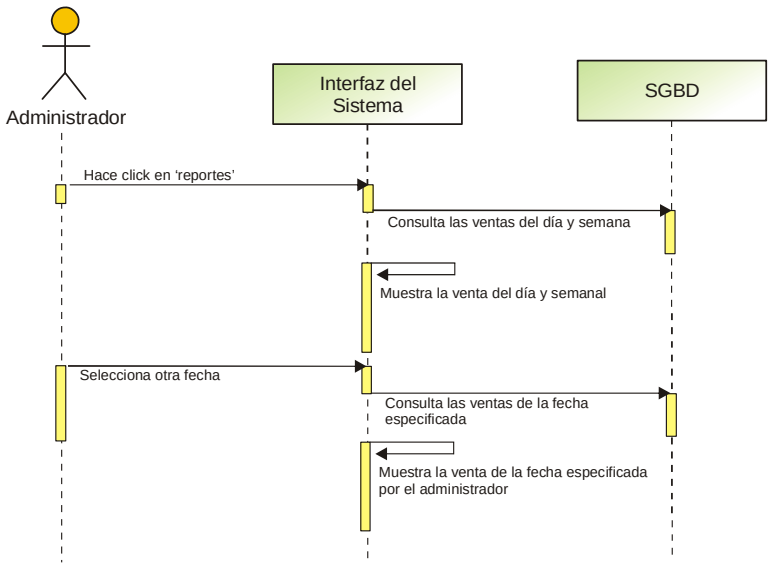


Fig. 1.3 – Diagrama de secuencia

1.2.2.5 Diagramas de actividades.

El diagrama de actividades del UML es muy parecido a los viejos diagramas de flujo. Le muestra los pasos (conocidos como actividades) así como puntos de decisión y bifurcaciones. Es útil para mostrar lo que ocurre en un proceso de negocios u operación y es parte integral del análisis de un sistema.

Para empezar un diagrama de actividades ha sido diseña para mostrar una visón simplificada de lo que ocurre durante una operación o proceso. Es una extensión de un diagrama de estados. El diagrama de estados muestra los estados de un objeto y representa las actividades como flechas que conectan a los estados. El diagrama de actividades resalta, precisamente, a las actividades. [9]

A cada actividad se le representa por un rectángulo con las esquinas redondeadas. Una flecha representa la transacción de una a otra actividad. Un circulo relleno representa el punto inicial del diagrama y una diana representa el final del mismo.

Las decisiones se representan con un rombo (fig. 1.4a) y los la separación o unión de rutas (hilos) concurrentes con una barra (fig. 1.4b).

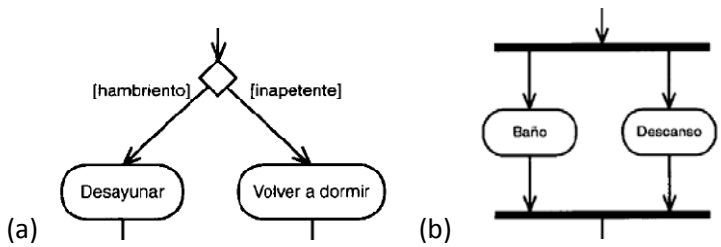


Figura 1.4 - Decisiones y bifurcaciones

Las indicaciones son envío de mensajes que se representan por medio de pentágonos, cóncavo para la recepción de un evento y convexo para el envío (Fig. 1.5).

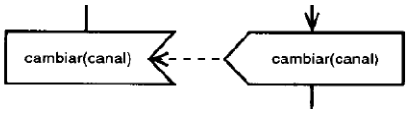


Figura 1.5 – Envío y recepción de mensajes

1.2.2.6 Diagramas de componentes.

Un componente de software es una parte física de un sistema y se encuentra en la computadora, no en la mente del analista. ¿Qué puede tomarse como componente? Una tabla, un archivo de datos, un archivo ejecutable, una DLL, documentos y cosas por el estilo.

¿Cuál es la relación entre un componente y una clase? Imagine a un componente como la personificación en software de una clase. La clase representa una abstracción de un conjunto de atributos y operaciones. Un importante punto por recordar de los componentes y clases es que un componente puede ser la implementación de más de una clase. [15]

Representar los componentes (utilizando un diagrama u otra técnica) se debe a diversas razones, a saber:

- Los clientes pueden ver la estructura del sistema finalizado
- Los desarrolladores cuentan con una estructura con la cual trabajar en adelante.
- Quienes escriban las notas técnicas y la documentación podrán entender de qué escribirán.
- Se distinguen más fácilmente y pueden ser reutilizados.

Con las necesidades actuales de los negocios de soluciones rápidas, entre más rápido presente un sistema para producción mayor será su competitividad. Si puede crear un componente para un sistema y puede volver a utilizarlo en otro, habrá contribuido a esa competitividad. Es recomendable tomarse el tiempo y esfuerzo para modelar un componente que ayude a que esta reutilización pueda llevarse a cabo.

Podemos diferenciar tres tipos de componentes:

1. *Componentes de distribución,* que conforman el fundamento de los sistemas ejecutables (por ejemplo, DLL, .exe, controles ActiveX, etc.).
2. *Componentes para trabajar en el producto,* a partir de los cuales se han creado los componentes de distribución (como archivos de base de datos y de código).
3. *Componentes de ejecución,* creados como resultado de un sistema en ejecución.

Una vez que se conocen los componentes de un sistema, podemos representarlos en un diagrama de componentes.

El símbolo principal de un diagrama de componentes es un rectángulo que lleva dentro el nombre del componente, se puede indicar el tipo del mismo arriba del nombre (Fig. 1.6).

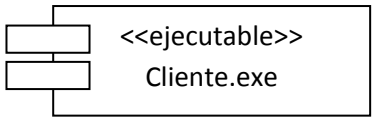


Fig. 1.6 – Representación de un componente

Después del nombre podemos agregar ' : ' y seguidamente podemos enumerar las clases que implementa el componente. Los archivos de entrada pueden ser representados por un recuadro con una esquina truncada.

1.3 Bases de Datos

Uno de los usos más frecuentes de los ordenadores es el de almacenamiento masivo de datos. Aquí es importante hace distinción entre datos e información. Cuando nos referimos a datos, estamos hablando de información que puede ser relacionada entre sí, información que la computadora puede almacenar organizar y recuperar.

Sabiendo que es un dato, ahora podemos definir lo que es una base de datos. Aunque no podemos dar una definición completa y única, ya que existen varios puntos de vista de los expertos y por ende varias definiciones.

En esencia, una base de datos no es más que una colección de información que existe a lo largo de un periodo de tiempo, a menudo de varios años. Más claramente el término base de datos se refiere a una colección de datos gestionada por un sistema gestor de bases de datos (en lo posterior nos referiremos a este como S.G.B.D.). [16]

Otra definición es que las Bases de Datos son un conjunto exhaustivo no redundante de datos estructurados, organizados independientemente de su utilización y su implementación, en máquinas accesibles en tiempo real, y compatibles con usuarios concurrentes con necesidad de información diferente y no predicable en tiempo. [13]

1.3.1. Historia de los Sistemas de Bases de Datos

El procesamiento de datos impulsa el crecimiento de los computadores, como ocurriera en los primeros días de los computadores comerciales. Desde las tarjetas perforadas, inventadas por Hollerith, que se usaron en los principios del siglo XX para registrar los datos del censo de los EE.UU., y se usaron sistemas mecánicos para procesar las tarjetas y para tabular los resultados. Las técnicas del almacenamiento de datos han evolucionado a lo largo de los años.

- **Década de 1950 principios de 1960.** Se desarrollaron las cintas magnéticas para el almacenamiento de datos las tareas de procesamiento de datos tales-como las nóminas fueron automatizadas, con los datos almacenados en cintas. Solo se podían leer secuencialmente, y los tamaños de datos eran mucho mayores que la memoria principal.

• **Finales de 1960 y 1970.** El amplio uso de los discos fijos a finales de la década de 1960 cambió en gran medida el escenario del procesamiento de datos, ya que los discos fijos permitieron el acceso directo a los datos.

• **Década de 1980.** Aunque académicamente interesante, el modelo relacional no se usó inicialmente en la práctica debido a sus inconvenientes por el rendimiento (las bases de datos relaciónales no pudieron competir con el rendimiento de las bases de datos de red y jerárquicas existentes).

• **Principios de 1990.** El lenguaje SQL se diseñó fundamentalmente para las aplicaciones de ayuda a la toma de decisiones, que son intensivas en consultas.

• **Finales de 1990.** El principal acontecimiento fue el crecimiento explosivo de World Wide Web. Las bases de datos se implantaron mucho más extensivamente que nunca antes. Esta corriente continúa en el 2000.[13]

1.3.2. Modelo de los Datos

Bajo la estructura de la base de datos se encuentra el modelo de datos: una colección de herramientas conceptuales para describir los datos, las relaciones, la semántica y las restricciones de integridad. [2]

1.3.2.1 Modelo Entidad - Relación.

El modelo de datos Entidad - Relación (E -R) está basado en una percepción del mundo real que consta de una colección de objetos básicos, llamados entidades, y de relaciones entre estos objetos. Las entidades se describen en una base de datos mediante un conjunto de atributos . [2]

Una relación es una asociación entre varias entidades. El conjunto de todas las entidades del mismo tipo, y el conjunto de todas las relaciones del mismo tipo, se denominan respectivamente conjunto de entidades y conjunto de relaciones.

La estructura lógica general de una base de datos se puede expresar gráficamente mediante un diagrama Entidad - Relación (comúnmente llamado Diagrama ER), que consta de los siguientes componentes:

- **Rectángulos,** que representan conjuntos de entidades.
- **Elipses,** que representan atributos.
- **Rombos,** que representan relaciones entre conjuntos de entidades.
- **Líneas,** que unen los atributos con los conjuntos de entidades y los conjuntos de entidades con las relaciones.

Cada componente se etiqueta con la entidad o relación que representa. [13]

1.3.2.2 Relaciones.

Las relaciones representan las reglas y la información que el negocio necesita. Una relación es una asociación importante entre dos entidades. [1]

Una relación en un diagrama ER se muestra con una línea que une a dos entidades, si la línea es continua indica una relación obligatoria, si la línea es punteada la relación es opcional.

El grado de una relación se representa en los extremos de las relaciones. Si la línea es única, se trata de un rango de "uno y solo uno", si la línea se abre en dos vértices la relación es de grado "uno o más". La figura 1.7 muestra el uso esta simbología para ilustrar diagramas ER.

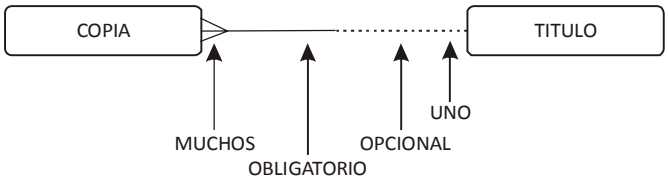


Fig. 1.7 - Ejemplo de un diagrama ER

Los tipos de relaciones pueden ser:

- *Relaciones Muchos a Uno*: Estas son las más comunes y muestran que una relación tiene grado de uno o más en una dirección y solamente uno en la otra dirección.
- *Relaciones Muchos a Muchos*: Hay un grado de uno o más en ambas direcciones y también son muy comunes. Usualmente son opcionales en ambas direcciones.
- *Relaciones Uno a Uno*: Hay un grado de uno y sólo uno en ambas direcciones. Este tipo de relaciones es bastante raro, comúnmente indican que las dos entidades son realmente la misma entidad en términos de negocio.

1.3.2.3 Atributos.

Los atributos son información acerca de una entidad que necesita ser conocida o descrita. Los atributos describen una entidad por que la habilitan, identifican, clasifican, enumeran ó expresan su estado. [1]

Los atributos solo pueden tener un solo tipo de dato y almacenar un único valor. Cuando un atributo es obligatorio (marcado generalmente con un asterisco o si el nombre de este está subrayado) debe tener almacenado un valor. Si el atributo es opcional (marcado con una °) entonces el atributo puede no tener valor.

1.3.3 Normalización

Para el experto diseñador de bases de datos, derivar entidades o registro de tipo conceptual de un grupo de datos se puede hacer intuitivamente. Sin embargo, tal intuición no

siempre surge espontáneamente en los principiantes, especialmente cuando el diseño es muy complejo.

La teoría de la normalización es una ayuda que proporciona un procedimiento riguroso para el diseño de bases de datos. Una base de datos mal diseñada puede funcionar inicialmente pero puede mostrar anomalías en el almacenamiento debidas al agrupamiento indiscriminado de los campos cuando se efectúan en los archivos las operaciones de inserción, actualización y eliminación. La teoría de normalización ayuda a reconocer las cualidades no deseadas en un archivo y la forma de corregirlas. [1]

Una relación no-normalizada es una relación que contiene varias ocurrencias de algunos valores en cualquiera de sus campos. Por otro lado, una relación normalizada sólo permite una ocurrencia de un valor en cada campo. Las relaciones se agrupan en cuatro categorías llamadas formas normales (FN) siendo cada nivel una descomposición más completa de una relación que la del nivel anterior.

1.3.3.1 Formas Normales

Son las técnicas para prevenir las anomalías en las tablas. Decimos que una tabla está en una forma normal en particular si satisface cierto conjunto de condiciones preestablecidas. Se han definido muchas formas normales. El diseño de una base de datos debe cumplir con al menos las tres primeras.

Primera forma normal

Definición formal: Una relación R se encuentra en 1FN si y sólo si por cada renglón columna contiene valores atómicos. Abreviada como 1FN, se considera que una relación se encuentra en la primera forma normal cuando cumple lo siguiente:

1. Las celdas de las tablas poseen valores simples y no se permiten grupos ni arreglos repetidos como valores, es decir, contienen un solo valor por cada celda.
2. Todos los ingresos en cualquier columna (atributo) deben ser del mismo tipo.
3. Cada columna debe tener un nombre único, el orden de las columnas en la tabla no es importante.
4. Dos filas o renglones de una misma tabla no deben ser idénticas, aunque el orden de las filas no es importante.

Segunda forma normal.

Para definir formalmente la segunda forma normal requerimos saber que es una dependencia funcional: Consiste en edificar que atributos dependen de otro(s) atributo(s).

Definición formal: Una relación R está en 2FN si y sólo si está en 1FN y los atributos no primos dependen funcionalmente de la llave primaria.

Una relación se encuentra en segunda forma normal, cuando cumple con las reglas de la primera forma normal y todos sus atributos que no son claves (llaves) dependen por completo de

la clave. De acuerdo con esta definición, cada tabla que tiene un atributo único como clave, está en segunda forma normal.

Tercera forma normal.

Para definir formalmente la 3FN necesitamos definir dependencia transitiva: En una afinidad (tabla bidimensional) que tiene por lo menos 3 atributos (A,B, C) en donde A determina a B, B determina a C pero no determina a A.

Definición formal: Una relación R está en 3FN si y sólo si esta en 2FN y todos sus atributos no primos dependen no transitivamente de la llave primaria.

Consiste en eliminar la dependencia transitiva que queda en una segunda forma normal, en pocas palabras una relación esta en tercera forma normal si está en segunda forma normal y no existen dependencias transitivas entre los atributos, nos referimos a dependencias transitivas cuando existe más de una forma de llegar a referencias a un atributo de una relación.

Forma normal de Boyce Codd.

Determinante: Uno o más atributos que, de manera funcional, determinan otro atributo o atributos. En la dependencia funcional (A,B)-->C, (A,B) son los determinantes.

Definición formal: Una relación R esta en FNBC si y sólo si cada determinante es una llave candidato. Denominada por sus siglas en inglés como BCNF; Una tabla se considera en esta forma si y sólo sí cada determinante o atributo es una llave candidato.

1.3.4 Lenguajes de Bases de datos.

Un sistema de bases de datos proporciona un **lenguaje de definición de datos** para especificar el esquema de la base de datos y un **lenguaje de manipulación de datos** para expresar las consultas a la base de datos y las modificaciones. [12]

1.3.4.1 Lenguaje de Definición de Datos

Un esquema de base de datos se especifica mediante un conjunto de definiciones expresadas mediante un lenguaje especial llamado lenguaje de definición de datos (LDD).

Especificamos el almacenamiento y los métodos de acceso usados por el sistema de bases de datos por un conjunto de instrucciones en un tipo especial de LDD. Estas instrucciones definen los detalles de implementación de los esquemas de base de datos, que se ocultan usualmente a los usuarios. [2]

1.343.1 Lenguaje de Manipulación de Datos.

La manipulación de datos es:

- La recuperación de 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.

Un **lenguaje de manipulación de datos (LMD)** es un lenguaje que permite a los usuarios acceder o manipular los datos organizados mediante el modelo de datos apropiado. [2]

1.3.5 El sistema gestor de bases de datos.

Según Silberchatz, podemos considerar un SGBD como «un programa que proporciona la interfaz entre los datos de bajo nivel almacenados en la base de datos, y los programas de aplicación y consultas hechos al sistema». [2]

Ya hemos indicado las características propias de una base de datos, la mayoría de las cuales recae precisamente sobre el SGDB para dejarlas más patentes, las reunimos y ampliamos aquí.

- Interactuar con el sistema operativo.
- Mantener la integridad.
- Mantener la seguridad, permitir las copias de seguridad.
- Controlar la concurrencia.
- Suministrar mecanismos que faciliten la interacción con la base de datos.

El objetivo de un SGDB es minimizar hasta donde sea posible el acceso a disco.

1.4 Panorámica del SGBD MySQL[9]

MySQL, el sistema de gestión de bases de datos (SGBD) SQL Open Source más popular, lo desarrolla, distribuye y soporta MySQL AB. MySQL AB es una compañía comercial, fundada por los desarrolladores de MySQL. Es una compañía Open Source de segunda generación que une los valores y metodología Open Source con un exitoso modelo de negocio.

• MySQL es un sistema de gestión de bases de datos Una base de datos es una colección estructurada de datos. Puede ser cualquier cosa, desde una simple lista de compra a una galería de pintura o las más vastas cantidades de información en una red corporativa. Para añadir, acceder, y procesar los datos almacenados en una base de datos, necesita un sistema de gestión de base de datos como MySQL Server. Al ser los computadores muy buenos en tratar grandes cantidades de datos, los sistemas de gestión de bases de datos juegan un papel central en computación, como aplicaciones autónomas o como parte de otras aplicaciones. Información general

• MySQL es un sistema de gestión de bases de datos relacionales. Una base de datos relacional almacena datos en tablas separadas en lugar de poner todos los datos en un gran almacén.

Esto añade velocidad y flexibilidad. La parte SQL de "MySQL"se refiere a "Structured Query Language". SQL es el lenguaje estandarizado más común para acceder a bases de datos y está definido por el estándard ANSI/ISO SQL.

- MySQL software es Open Source. Open Source significa que es posible para cualquiera usar y modificar el software. Cualquiera puede bajar el software MySQL desde internet y usarlo sin pagar nada. Si lo desea, puede estudiar el código fuente y cambiarlo para adaptarlo a sus necesidades

- MySQL Server se desarrolló originalmente para tratar grandes bases de datos mucho más rápido que soluciones existentes y ha sido usado con éxito en entornos de producción de alto rendimiento durante varios años. MySQL Server ofrece hoy en día una gran cantidad de funciones Su conectividad, velocidad, y seguridad hacen de MySQL Server altamente apropiado para acceder bases de datos en Internet

- MySQL Server trabaja en entornos cliente/servidor o incrustados. El software de bases de datos MySQL es un sistema cliente/servidor que consiste en un servidor SQL multi-threaded que trabaja con diferentes backends, programas y bibliotecas cliente, herramientas administrativas y un amplio abanico de interfaces de programación para aplicaciones (APIs).

- También se proporciona el MySQL Server como biblioteca incrustada multi-threaded que puede ligar en su aplicación para obtener un producto más pequeño, rápido y fácil de administrar.

1.4.1 Principales características de MySQL [9]

La siguiente lista describe algunas de las características más importantes del software de base de datos MySQL.

- Interioridades y portabilidad
- Escrito en C y en C++
- Probado con un amplio rango de compiladores diferentes
- Funciona en diferentes plataformas.
- APIs disponibles para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl.
- Uso completo de multi-threaded mediante threads del kernel. Pueden usarse fácilmente multiple CPUs si están disponibles.
- Proporciona sistemas de almacenamiento transaccionales y no transaccionales.
- Usa tablas en disco B-tree (MyISAM) muy rápidas con compresión de índice.
- Un sistema de reserva de memoria muy rápido basado en threads.
- Joins muy rápidos usando un multi-join de un paso optimizado.
- Las funciones SQL están implementadas usando una librería altamente optimizada y deben ser tan rápidas como sea posible. Normalmente no hay reserva de memoria tras toda la inicialización para consultas.

- El servidor está disponible como un programa separado para usar en un entorno de red cliente/ servidor. También está disponible como biblioteca y puede ser incrustado (ligado) en aplicaciones autónomas. Dichas aplicaciones pueden usarse por sí mismas o en entornos donde no hay red disponible.
- Diversos tipos de columnas.
- Registros de longitud fija y longitud variable.
- Sentencias y funciones
- Soporte para alias en tablas y columnas como lo requiere el estándar SQL.
- Puede mezclar tablas de distintas bases de datos en la misma consulta.
- Un sistema de privilegios y contraseñas que es muy flexible y seguro, y que permite verificación basada en el host. Las contraseñas son seguras porque todo el tráfico de contraseñas está encriptado cuando se conecta con un servidor.
- Soporte a grandes bases de datos. Los creadores de MySQL Server lo usan con bases de datos que contienen 50 millones de registros. También se sabe usuarios que usan MySQL Server con 60.000 tablas y acerca de 5.000.000 de registros.
- Se permiten hasta 64 índices por tabla
- Los clientes pueden conectar con el servidor MySQL usando sockets TCP/IP en cualquier plataforma. En sistemas Windows de la familia NT (NT,2000,XP, o 2003), los clientes pueden usar 'named pipes' para la conexión. En sistemas Unix, los clientes pueden conectar usan- do ficheros socket Unix.
- La interfaz para el conector ODBC (MyODBC) proporciona a MySQL soporte para programas clientes que usen conexiones ODBC (Open Database Connectivity).

1.4.2 Sintaxis básica del SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

1.4.2.1 Comandos

Existen dos tipos de comandos SQL:

1. Los comandos de DDL que permiten crear y definir nuevas bases de datos, campos e índices.
2. Los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comandos DDL

(*Data Definition Languaje*, Lenguaje de definición de datos, por sus siglas en inglés)

- **CREATE** Utilizado para crear nuevas tablas, campos e índices.
- **DROP** Empleado para eliminar tablas e índices.

- **ALTER** Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos. [9]

Comandos DML

(*Data manipulation languaje*, Lenguaje de manipulación de datos),

- **SELECT** Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
- **INSERT** Utilizado para cargar lotes de datos en la base de datos en una única operación.
- **UPDATE** Utilizado para modificar los valores de los campos y registros especificados
- **DELETE** Utilizado para eliminar registros de una tabla de una base de datos.[9]

Cláusulas.

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

- **FROM** Utilizada para especificar la tabla de la cual se van a seleccionar los registros.
- **WHERE** Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.
- **GROUP BY** Utilizada para separar los registros seleccionados en grupos específicos.
- **HAVING** Utilizada para expresar la condición que debe satisfacer cada grupo.
- **ORDER BY** Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico. [9]

Operadores Lógicos.

- **AND** Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
- **OR** Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
- **NOT** Negación lógica. Devuelve el valor contrario de la expresión. [9]

Operadores de Comparación.

- < Menor que.
- > Mayor que.
- <> Distinto de.
- <= Menor ó Igual que.
- >= Mayor ó Igual que.
- **BETWEEN** Utilizado para especificar un intervalo de valores.
- **LIKE** Utilizado en la comparación de un modelo.
- **In** Utilizado para especificar registros de una base de datos. [9]

Funciones de Agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

- **AVG** Utilizada para calcular el promedio de los valores de un campo determinado.
- **COUNT** Utilizada para devolver el número de registros de la selección.
- **SUM** Utilizada para devolver la suma de todos los valores de un campo determinado.
- **MAX** Utilizada para devolver el valor más alto de un campo especificado.
- **MIN** Utilizada para devolver el valor más bajo de un campo especificado.
-

1.5 Delphi

Delphi es un entorno de desarrollo de software (IDE) diseñado para la programación de propósito general con énfasis en la programación visual. En Delphi utiliza como lenguaje de programación una versión moderna de Pascal llamada Object Pascal. Es producido comercialmente por la empresa estadounidense CodeGear. En sus diferentes variantes, permite producir archivos ejecutables para Windows, Linux y la plataforma .NET. [4]

CodeGear ha sido subsidiaria de la empresa Borland, donde Delphi se creó originalmente, tras un proceso que pretendía en principio la venta del departamento de herramientas para desarrollo.

El nombre Delphi hace referencia al oráculo de Delfos. Borland eligió ese nombre para resaltar su principal mejora con respecto a su antecesor (Turbo Pascal), que sería su conectividad con bases de datos Oracle (oráculo, en inglés). El nombre se suele pronunciar delfi en Europa, incluida Gran Bretaña. Se usa delfái en inglés de Estados Unidos (existe una población Delphi con esa pronunciación), por lo que es la preferida por Borland.

1.5.1 Uso y variantes.

Un uso habitual de Delphi (aunque no el único) es el desarrollo de aplicaciones visuales y de bases de datos cliente-servidor y multicapas. Debido a que es una herramienta de propósito múltiple, se usa también para proyectos de casi cualquier tipo, incluyendo aplicaciones de consola, aplicaciones de web (por ejemplo servicios web, CGI, ISAPI, NSAPI, módulos para Apache), servicios COM y DCOM, y servicios del sistema operativo. Entre las aplicaciones más populares actualmente destaca Skype, un programa de telefonía por IP, Nero Burning Rom y varios más. [4]

Existe una versión de Delphi para sistemas Unix y Linux, denominada Kylix (de la cual existe un versión gratuita, aunque limitada). Sin embargo Kylix fue congelado por Borland en su versión 3.0.

1.5.2 El lenguaje de programación.

Delphi está basado en una versión moderna de Pascal, denominada Object Pascal. Borland en los últimos años defendía que el nombre correcto del lenguaje es también Delphi, posiblemente debido a pretensiones de marca, aunque en sus mismos manuales el nombre del lenguaje aparecía como Object Pascal, por lo que la comunidad de programadores no ha adoptado mayoritariamente este cambio (supuesta aclaración, según Borland). Object Pascal expande las funcionalidades del Pascal estándar:

Soporte para la programación orientada a objetos (habitualmente llamada POO) también existente desde Turbo Pascal 5.5, pero más evolucionada en cuanto a:

- Encapsulación: declarando partes privadas, protegidas, públicas y publicadas de las clases
- Propiedades: concepto nuevo que luego han adaptado muchos otros lenguajes. Las propiedades permiten usar la sintaxis de asignación para setters y getters.
- Simplificación de la sintaxis de referencias a clases y punteros.
- Soporte para manejo estructurado de excepciones, mejorando sensiblemente el control de errores de usuario y del sistema.
- Programación activada por eventos (event-driven), posible gracias a la técnica de delegación de eventos. Esta técnica permite asignar el método de un objeto para responder a un evento lanzado sobre otro objeto. Fue adoptada por Niklaus Wirth, autor del Pascal Original, e incorporada a otros de sus lenguajes como Component Pascal. [4]

1.5.3 Componentes.

Delphi dio una implementación muy buena a la idea del uso de componentes, que son piezas reutilizables de código (clases) que pueden interactuar con el IDE en tiempo de diseño y desempeñar una función específica en tiempo de ejecución. Desde un enfoque más específico de la herramienta, se catalogan como componentes todos aquellos objetos que heredan de la clase TComponent, donde se implementa la funcionalidad necesaria para interactuar con el entorno de desarrollo, la carga dinámica desde streams y la liberación de memoria mediante una jerarquía. Una gran parte de los componentes disponibles para Delphi son controles (derivados de TControl), que encapsulan los elementos de interacción con el usuario como botones, menus, barras de desplazamiento, etcétera.

Delphi incluye una biblioteca de clases bien diseñada denominada VCL (Visual Component Library, Biblioteca de Componentes Visuales) y, en sus versiones 6 y 7, una jerarquía multiplataforma paralela denominada CLX. Ésta también se incluye en Kylix. Estas jerarquías de

objetos incluyen componentes visuales y no visuales, tales como los pertenecientes a la categoría de acceso a datos, con los que puede establecerse conexiones de forma nativa o mediante capas intermedias (como ADO, BDE u ODBC) a la mayoría de las bases de datos relacionales existentes en el mercado. La VCL también está disponible para el desarrollo en.NET.

Además de poder utilizar en un programa estos componentes estándar (botones, grillas, conjuntos de datos, etc.), es posible crear nuevos componentes o mejorar los ya existentes, extendiendo la funcionalidad de la herramienta. En Internet existe un gran número de componentes, tanto gratuitos como comerciales, disponibles para los proyectos a los que no les basten los que vienen ya con la herramienta. [4]

1.5.4 Eventos.

Delphi permite de manera sencilla ejecutar trozos de código en respuesta a acciones o eventos (sucesos) que ocurren durante el tiempo que un programa se ejecuta. Por ejemplo, cuando se presiona un botón, la VCL captura la notificación estándar de windows, y detecta si hay algún método asociado al evento OnClick del botón. Si lo hay, manda ejecutar dicho método.

Los eventos pueden generarse debido a la recepción de señales desde elementos de hardware como el ratón o el teclado, o pueden producirse al realizar alguna operación sobre un elemento de la propia aplicación (como abrir un conjunto de datos, que genera los eventos BeforeOpen/AfterOpen). La VCL ha demostrado estar bien diseñada y el control que se tiene a través de los eventos de la misma es suficiente para la gran mayoría de las aplicaciones. [4]

1.5.5 Base de Datos.

Una de las principales características y ventajas de Delphi es su capacidad para desarrollar aplicaciones con conectividad a bases de datos de diferentes fabricantes. El programador de Delphi cuenta con una gran cantidad de componentes para realizar la conexión, manipulación, presentación y captura de los datos, algunos de ellos liberados bajo licencias de código abierto o gratuitos. Estos componentes de acceso a datos pueden enlazarse a una gran variedad de controles visuales, aprovechando las características del lenguaje orientado a objetos, gracias al polimorfismo.

En la paleta de componentes pueden encontrarse varias pestañas para realizar una conexión a bases de datos usando diferentes capas o motores de conexión.

Hay motores que permiten conectarse a bases de datos de diferentes fabricantes tales como BDE, DBExpress o ADO, que cuentan con manejadores para los formatos más extendidos.

También hay componentes de conexión directa para un buen número de bases de datos específicas: Firebird, Interbase, Oracle, etcétera.

A continuación un breve resumen (aún recopilándose) de las capas de conexión disponibles para las bases de datos más populares:

- Interbase/Firebird: IBX (InterBase eXpress), IBO (IB Objects), MDO (Mercury Data Objects), *DBExpress, BDE, FibPlus, Zeos.
- Oracle: DOA (Direct Oracle Access), NCOci8.
- dBase: BDE.
- FoxPro: BDE.
- Paradox: BDE.
- MS-SQL: BDE, ADO, *DBExpress.
- Postgres: BDE, ADO.
- MySQL: Zeos (nativo), *DBExpress, BDE y ADO (usando ODBC).

Este último, Zeos, es un componente que utilizaremos para acceder a nuestra base de datos en MySQL desde Delphi.

1.5.6 Borland Database Engine (BDE).

Es un motor de conexión a bases de datos de uso bastante amplio y que permite manejar bases de datos de escritorio como dBase, Foxpro y Paradox, además de ofrecer la capacidad para conectarse a servidores SQL locales y remotos. Su uso, va siendo cada vez menor, debido a la pobre gestión de memoria que realiza, sustituyéndolo por componentes más actualizados y especializados como DOAC (Direct Oracle Access Components) o DBExpress, esto sumado a la fiabilidad que estan presentando los nuevos gestores de Datos en especial tecnologías como RDO y ADO; los cuales son mantenidos por sus fabricantes, forzando la compatibilidad con las versiones preliminares; liberando al programador de actualizaciones en cuanto a gestión de datos. Actualmente ya no es desarrollado por Codegear. [4]

1.5.7 Desarrollo visual.

Como entorno visual, la programación en Delphi consiste en diseñar los formularios que componen al programa colocando todos sus controles (botones, etiquetas, campos de texto, etc.) en las posiciones deseadas, normalmente usando un ratón. Luego se asocia código a los eventos de dichos controles y también se pueden crear módulos de datos, que regularmente contienen los componentes de acceso a datos y las reglas de negocio de una aplicación. [4]

1.5.8 Depurador integrado.

Es una potente característica que nos permite establecer puntos de ruptura (breakpoints), la ejecución paso a paso de un programa, el seguimiento de los valores de las variables y de la pila

de ejecución, así como la evaluación de expresiones con datos de la ejecución del programa. Con su uso, un programador experimentado puede detectar y resolver errores lógicos en el funcionamiento de un aplicativo desarrollado con Delphi. En las ediciones Client/Server y Enterprise se añade la opción de depuración a programas corriendo en equipos remotos (remote debugging), lo que posibilita el uso de todas las características del depurador con un programa ejecutándose en su entorno normal de trabajo y no en el ordenador del programador (en donde muchas veces no ocurren los errores). [4]

Capitulo 2

2 Análisis del sistema

CAPITULO 2.
Análisis del Sistema

2.1. Planteamiento Del Problema

Actualmente, en una tienda de refacciones para bicicletas se maneja demasiada información: un enorme catálogo de productos y una gran existencia de estos mismos en almacén y en exhibición; control de ventas, registro de clientes, etc.

Toda esta información se maneja de manera manual y es guardada en papel, la consulta de precios de los artículos se hace a través de diversos catálogos que los vendedores han provisto. Esto nos lleva a que no se tengan actualizados los precios de los productos.

Otro problema surge al realizar ventas, no se sabe con exactitud de que piezas se dispone, ni de la cantidad de la misma, por ello, el encargado debe hacer una búsqueda en el almacén para constatar la existencia del producto. De igual manera no se puede saber que piezas están agotadas y el dueño, para realizar una lista de pedidos a sus proveedores, debe hacer un recorrido por el almacén y verificar que productos hacen falta o están agotados.

Considerando todos estos problemas, y la dificultad de manipular de manera efectiva una información tan vasta y compleja, surge la necesidad de contar con una herramienta computacional para llevar el control de toda esta información.

Desarrollaremos e implantaremos un sistema que lleve el control de la existencia de los productos en el inventario, la relación de empresas con las piezas que proveen y registre las ventas logradas. Además automatizará la generación de pedidos, la impresión de comprobantes de venta y se advertirá al usuario acerca de piezas por agotarse.

2.2 Referencias Del Sistema

El usuario final y quien ha brindado toda la información necesaria para la elaboración del sistema es:

NOMBRE: JESUS BAUTISTA TELLEZ
PUESTO: DUEÑO
TEL: 249 113 3763

2.3 Restricciones.

a) En infraestructura

No Aplica

b) Funcionalidad

Los Encargados de mostrador no podrán ver la siguiente información cuando efectúan una venta o consultan el precio de un artículo: Que proveedor surte la pieza y el precio a que se compra.

c) Restricciones de entorno de Operación.

No aplica

2.4 Descripción De La Información.

Este proyecto consiste en el desarrollo de un sistema que permita a la empresa 'Bicicletas y Refacciones de Tecamachalco' llevar el control y la administración de la información que se tiene dentro de la refaccionaria, tal como:

- Llevar control sobre la existencia de productos en almacén y en exhibición (inventario).
- Advertir acerca de los productos que estén por agotarse y generar pedidos según el proveedor.
- Información de Clientes.
- Información de proveedores.
- Venta de productos y servicios.
- Emisión de notas de compra (tickets).
- Reportes de venta: diario y semanal.
- Manejar devoluciones y cambio de artículos ya vendidos.

El Sistema "SISTAR" (Sistema administrador de refaccionarias) será usado por varios trabajadores de la refaccionaria, entre ellos se encuentra un **administrador** que podrá realizar cualquier tipo de operación que hacen los empleados de mostrador y además podrá hacer altas y bajas en el inventario, hacer consultas sobre las ventas, administrar la información de clientes, proveedores y productos. Es el único que puede recibir devoluciones de artículos ó hacer cambio de los mismos. Los demás trabajadores, que llamaremos **encargados de mostrador ó cajeros,** solo podrán hacer consultas y registrar ventas e introducir la llegada de mercancía.

El **encargado** es una persona que estará al frente del mostrador para atender al cliente, vendiéndole los productos que desee o informando del costo de los artículos y servicios. Por lo

que necesita una manera de encontrar rápidamente el producto en el inventario, ya sea para agregarlo a la venta o solo para cotizarlo.

Los **productos** que se venden son bicicletas, accesorios para ciclistas, refacciones y accesorios para bicicletas, los cuales son identificados por el tipo de refacción o producto y marca, se anexa una descripción acerca de los mismos y la ubicación de estos en el establecimiento, es decir, si están en exhibición o en almacén. Los servicios que se ofrecen son reparación, pintura y ensamblaje de bicicletas.

Los **clientes** son personas que solicitan al encargado de mostrador cualquier tipo de producto o servicio. En la refaccionaria se manejan dos tipos de clientes, el 'publico' denominado así a los clientes ocasionales, 'taller' a los clientes frecuentes los cuales estarán registrados.

Al no. De productos que un cliente pide, con la descripción del producto y el costo le llamaremos 'pedido', es decir, cada producto que el cliente compre, la cantidad de dicho producto, el costo unitario y total le llamaremos pedido. No se puede hacer un pedido si no hay existencias de ese producto. Para cada pedido se especificará el precio del artículo, se manejan tres tipos de precios: Menudeo, Mayoreo y Preferencial al que denominaremos "Taller".

El total de pedidos efectuados en una misma compra se llamará 'venta'. Cada venta la identificaremos por el cliente que la compró, la fecha, todos los pedidos que la componen, sub-total, IVA y costo total. Al finalizar cada venta, el sistema debe imprimir un comprobante o nota de venta al cual llamaremos "ticket".

Un **Proveedor** es una persona ó empresa que se encarga de surtir a la refaccionara de diferentes artículos. Cada pieza es almacenada en el inventario y se especifica qué proveedor surte dicha pieza y el precio al que se obtiene. Se almacenará información indispensable acerca del proveedor, como es su nombre, dirección, teléfono, fax o correo electrónico en el que se le pueden hacer llegar los pedidos.

El sistema deberá ser capaz de generar la lista de pedidos, consultando cuales son los productos por agotarse y, especificando el proveedor, generar una lista de artículos para ser solicitados.

Requerimientos:

El usuario cuenta ya con un equipo de cómputo con las siguientes características:

- Procesador Pentiun IV® a 2.00 Ghz.
- Memoria Ram de 256 mb
- Disco duro de 80 GB
- Sistema Operativo Windows XP®

2.5 Estrategia de solución

Considerando los puntos anteriores podemos ya elaborar una estrategia para resolver la problemática de la empresa:

1. Se utilizará la metodología RUP para obtener los requerimientos del sistema, las diferentes vistas producidas por las disciplinas del RUP nos permitirán elaborar un sistema que se acerque cada vez más a la solución definitiva en cada iteración.
2. Se implementará una Base de Datos en MySQL. Se aprovechará la tecnología Cliente-Servidor de MySQL para implementar el sistema en varias maquinas y se puedan realizar ventas de manera concurrente.
3. Se elaborarán dos aplicaciones usando Delphi. Una para administrar todo el sistema y otra funcionará como punto de venta.

2.6 Especificación De Requerimientos.

El análisis y la especificación de requerimientos se mostrarán a través del modelo de casos de uso y del modelo diseño. Ambos Modelos en su última iteración producen los diagramas de casos de uso y de secuencia respectivamente.

2.6.1 Casos de Uso.

Nombre	Login (iniciar sesión caja)
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador Botón de iniciar	
Descripción El usuario se identifica como un cajero registrado en el sistema	
Actores Usuario Cajero	
Precondiciones Ninguna	
Flujo Normal 1. El usuario hace click en el botón de iniciar sesión 2. El sistema inhabilita las funciones de cajero	
Poscondiciones	
1. El sistema muestra la ventana principal de Caja	

Nombre	Login (iniciar sesión)
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador Botón de iniciar sesión como administrador	
Descripción El usuario se identifica como administrador para efectuar operaciones que requieren privilegios de administrador	
Actores Usuario Administrador	
Precondiciones El programa no debe estar ejecutando alguna otra operación	
Flujo Normal 1. El usuario hace click en el botón de iniciar sesión 2. El sistema muestra una ventana indicando al usuario que escriba su nombre de usuario y contraseña 3. El sistema comprueba la validez de los datos y habilita las funciones de administrador	
Flujo Alternativo 4. El sistema muestra un mensaje indicando que el nombre de usuario y contraseña es incorrecto 5. El usuario cancela la operación y el sistema cierra la ventana de login	
Poscondiciones 1. El sistema muestra el botón para cerrar la sesión	

Nombre	Administrar inventario
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador Botón de inventario	
Descripción Muestra una ventana con la lista de todos los productos con su respectiva información (costo, proveedor, existencia, etc).	
Actores Usuario Administrador	
Precondiciones El administrador debió iniciar sesión previamente	
Flujo Normal 1. El usuario hace click en el botón de inventario 2. El sistema muestra una ventana listando todos los artículos que se ofertan	

Nombre	Configurar
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador Botón de configuración	
Descripción El sistema muestra una ventana con diferentes opciones del sistema	
Actores Usuario Administrador	
Precondiciones El usuario se ha identificado como administrador	
Flujo Normal 1. El usuario hace click en el botón de configurar 2. El sistema muestra una ventana muestra una ventana con las opciones y configuración aplicables al sistema 3. El usuario hace los cambios convenientes y hace click en el botón de cerrar ventana 4. El sistema muestra una ventana pidiendo confirmar que la información proporcionada es correcta 5. El usuario confirma la información proporcionada y se cierra la ventana de opciones	
Flujo Alternativo 1. Es la primera vez que se utiliza el sistema y pide al usuario configurar todas las opciones del programa mostrando la ventana de configuración 5. El usuario cancela la confirmación de los datos y el sistema muestra nuevamente la ventana de opciones	
Poscondiciones 1. El sistema muestra la ventana principal con las operaciones de administrador habilitadas y el botón de cerrar sesión	

Nombre	Eliminar Productos
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador Botón de eliminar productos	
Descripción El administrador puede quitar productos del inventario	
Actores Usuario Administrador	
Precondiciones El administrador debe haber iniciado sesión y activado el botón de inventario	
Flujo Normal 1. El usuario hace click en el botón de eliminar producto 2. El sistema pide confirmar la acción al usuario 3. El usuario confirma la acción y el sistema elimina el producto	
Flujo Alternativo 2. El usuario cancela la operación	
Poscondiciones 1. El sistema actualiza y muestra el inventario	

Nombre	Agregar Productos
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador	
Botón de agregar productos	
Descripción	
El administrador puede agregar nuevos productos al inventario	
Actores	
Usuario Administrador	
Precondiciones	
El administrador debe haber iniciado sesión y activado el botón de inventario	
Flujo Normal	
1. El usuario hace click en el botón de agregar producto 2. El sistema muestra una ventana indicando al usuario que escriba el nombre, descripción, marca, existencias, precios, etc. Del producto. 4. El sistema genera la clave del producto y muestra una ventana para confirmar que los datos proporcionados por el usuario son correctos 5. El usuario confirma los datos y estos son agregados al inventario	
Flujo Alternativo	
1. El usuario cancela la operación 5. El usuario indica que los datos son erróneos y se pide reescribir los datos	
Poscondiciones	
1. El sistema actualiza y muestra el inventario	
Excepciones	
2.0.E.1 El usuario no escribe números válidos en los campos de existencias y precios. El sistema manejará la excepción descartando la información dada y pidiendo al usuario introducir números válidos	

Nombre	Actualizar información de productos
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador	
Botón de modificar o doble click sobre un elemento del inventario	
Descripción	
El usuario puede cambiar la información de cualquier producto en el inventario	
Actores	
Usuario Administrador	
Precondiciones	
El administrador tiene abierta la ventana del inventario	
Flujo Normal	
1. El usuario hace click en el botón de modificar 2. El sistema muestra una ventana con la información del artículo en campos editables 3. El usuario actualiza la información y finaliza con el botón de aceptar	
Flujo Alternativo	
1. El usuario hace doble click sobre un elemento de la lista de artículos del inventario 3. El usuario hace click en el botón cancelar y el sistema cierra la ventana y cancela la operación	
Poscondiciones	
1. La información del artículo ha sido actualizada	
Excepciones	

3.0.E.1 El usuario no escribe números válidos en los campos de existencias y precios. El sistema manejará la excepción descartando la información dada y pidiendo al usuario introducir números válidos

Nombre	Reportes
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador	
Botón de Reportes	
Descripción	
El sistema muestra un resumen de las ventas acumuladas en el día y en la semana	
Actores	
Usuario Administrador	
Precondiciones	
El administrador ha iniciado sesión	
Flujo Normal	
1. El usuario hace click en el botón de reportes 2. El sistema muestra el resumen de ventas del día y de la semana	
Flujo Alternativo	
3. El usuario especifica otra fecha u otra semana para consultar las ventas	
Poscondiciones	

Nombre	Administrar Clientes
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador	
Botón de Clientes	
Descripción	
El sistema lista todos los clientes y activa las opciones de modificar, agregar o eliminar	
Actores	
Usuario Administrador	
Precondiciones	
El administrador ha iniciado sesión	
Flujo Normal	
1. El usuario hace click en el botón de Clientes 2. El sistema muestra una ventana listando la información de todos los clientes	
Flujo Alternativo	
Poscondiciones	
1. El sistema muestra la lista de los clientes en una ventana	
Excepciones	

Nombre	Agregar Clientes
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador	
Botón de agregar clientes	
Descripción	
El administrador puede agregar nuevos clientes	
Actores	
Usuario Administrador	
Precondiciones	
El administrador debe haber iniciado sesión y activado el botón de clientes	
Flujo Normal	
1. El usuario hace click en el botón de agregar cliente 2. El sistema muestra una ventana indicando al usuario que escriba el nombre, rfc, etc. Del nuevo cliente. 3. El sistema genera el numero de cliente y muestra una ventana para confirmar que los datos proporcionados por el usuario son correctos 4. El usuario confirma los datos y estos son agregados al sistema	
Flujo Alternativo	
1. El usuario cancela la operación 4. El usuario indica que los datos son erróneos y se pide reescribir los datos	
Poscondiciones	
1. El sistema actualiza y muestra la lista de clientes	
Excepciones	

Nombre	Eliminar Cliente
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador	
Botón de eliminar clientes	
Descripción	
El administrador puede eliminar clientes	
Actores	
Usuario Administrador	
Precondiciones	
El administrador debe haber iniciado sesión y activado el botón de clientes	
Flujo Normal	
1. El usuario hace click en el botón de eliminar cliente 2. El sistema muestra una ventana para confirmar la acción 3. El usuario confirma la acción y la información del cliente es eliminada del sistema	
Flujo Alternativo	
3. El usuario cancela la acción	
Poscondiciones	
1. El sistema actualiza y muestra el listado de clientes	

Nombre	Actualizar información de clientes
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador	
Botón de modificar o doble click sobre el nombre de un cliente	
Descripción	
El usuario puede cambiar la información de los clientes almacenados en el sistema	
Actores	
Usuario Administrador	
Precondiciones	
El administrador tiene abierta la ventana de clientes	
Flujo Normal	
1. El usuario hace click en el botón de modificar 2. El sistema muestra una ventana con la información actual del cliente en campos editables 3. El usuario actualiza la información y finaliza con el botón de aceptar	
Flujo Alternativo	
1. El usuario hace doble click sobre un elemento de la lista de clientes 3. El usuario hace click en el botón cancelar y el sistema cierra la ventana y cancela la operación	
Poscondiciones	
1. La información del cliente ha sido actualizada	
Excepciones	

Nombre	Administrar Proveedores
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador	
Botón de Proveedores	
Descripción	
El sistema lista todos los proveedores y activa las opciones de modificar, agregar o eliminar	
Actores	
Usuario Administrador	
Precondiciones	
El administrador ha iniciado sesión	
Flujo Normal	
1. El usuario hace click en el botón de Proveedores 2. El sistema muestra una ventana listando la información de todos los proveedores	
Flujo Alternativo	
Poscondiciones	
1. El sistema muestra la lista de los proveedores en una ventana	
Excepciones	

Nombre	Agregar Proveedores
Autor	Misael Contreras
Fecha	4/Nov/2007

Disparador
Botón de agregar proveedores
Descripción
El administrador puede agregar nuevos proveedores
Actores
Usuario Administrador
Precondiciones
El administrador debe haber iniciado sesión y activado el botón de proveedores
Flujo Normal
1. El usuario hace click en el botón de nuevo proveedor 2. El sistema muestra una ventana indicando al usuario que escriba el nombre, teléfono, etc. Del nuevo cliente. 3. El sistema genera el numero de proveedor y muestra una ventana para confirmar que los datos proporcionados por el usuario son correctos 4. El usuario confirma los datos y estos son agregados al sistema
Flujo Alternativo
1. El usuario cancela la operación 4. El usuario indica que los datos son erróneos y se pide reescribir los datos
Poscondiciones
1. El sistema actualiza y muestra la lista de proveedores
Excepciones

Nombre	Eliminar proveedor
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador	
Botón de eliminar proveedores	
Descripción	
El administrador puede eliminar información de proveedores del sistema	
Actores	
Usuario Administrador	
Precondiciones	
El administrador debe haber iniciado sesión y activado el botón de proveedores	
Flujo Normal	
1. El usuario hace click en el botón de eliminar proveedor 2. El sistema muestra una ventana para confirmar la acción 3. El usuario confirma la acción y la información del proveedor es eliminada del sistema	
Flujo Alternativo	
3. El usuario cancela la acción	
Poscondiciones	
1. El sistema actualiza y muestra el listado de proveedores	

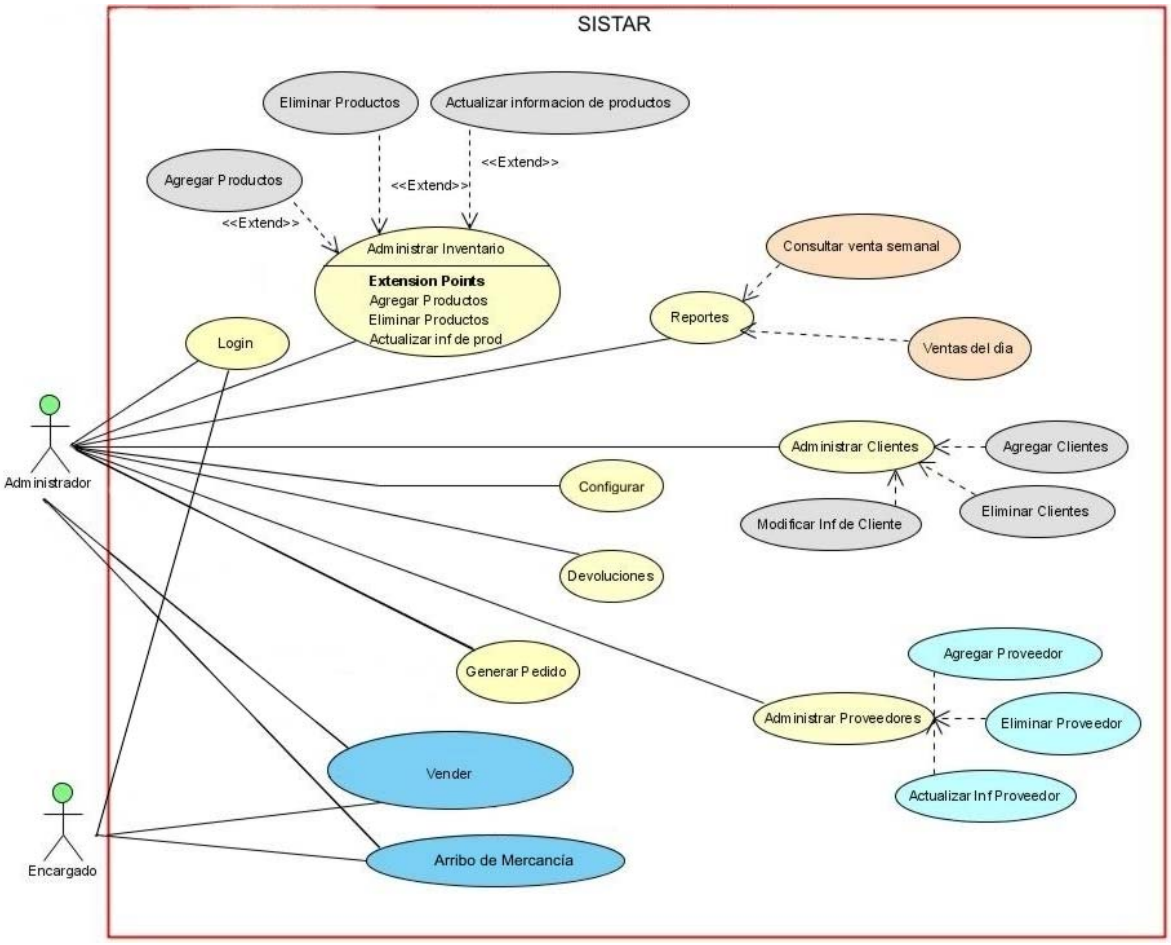
Nombre	Actualizar información de proveedores
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador	
Botón de modificar o doble click sobre el nombre de un proveedor	
Descripción	
El usuario puede cambiar la información de los proveedores almacenados en el sistema	
Actores	
Usuario Administrador	
Precondiciones	
El administrador tiene abierta la ventana de clientes	
Flujo Normal	
1. El usuario hace click en el botón de modificar 2. El sistema muestra una ventana con la información actual del proveedor en campos editables 3. El usuario actualiza la información y finaliza con el botón de aceptar	
Flujo Alternativo	
1. El usuario hace doble click sobre un elemento de la lista de proveedores 3. El usuario hace click en el botón cancelar y el sistema cierra la ventana y cancela la operación	
Poscondiciones	
1. La información del cliente ha sido actualizada	
Excepciones	

Nombre	Devoluciones
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador	
Botón de Cambios	
Descripción	
El administrador puede efectuar cambios en ventas realizadas	
Actores	
Usuario Administrador	
Precondiciones	
El usuario debe haber iniciado sesión como administrador	
Flujo Normal	
1. El usuario hace click en el botón devoluciones 2. El sistema muestra una ventana indicando al usuario que escriba el numero de venta 3. El sistema muestra la venta y el usuario puede eliminar artículos de la venta o modificar la cantidad 4. El usuario cierra la ventana, confirma los cambios y el sistema guarda los cambios	
Flujo Alternativo	
2. El usuario cancela la acción 3. El usuario cancela la operación y el sistema cierra la ventana	
Poscondiciones	
1. El sistema actualiza el inventario y las ventas según la fecha de la nota	
Excepciones	

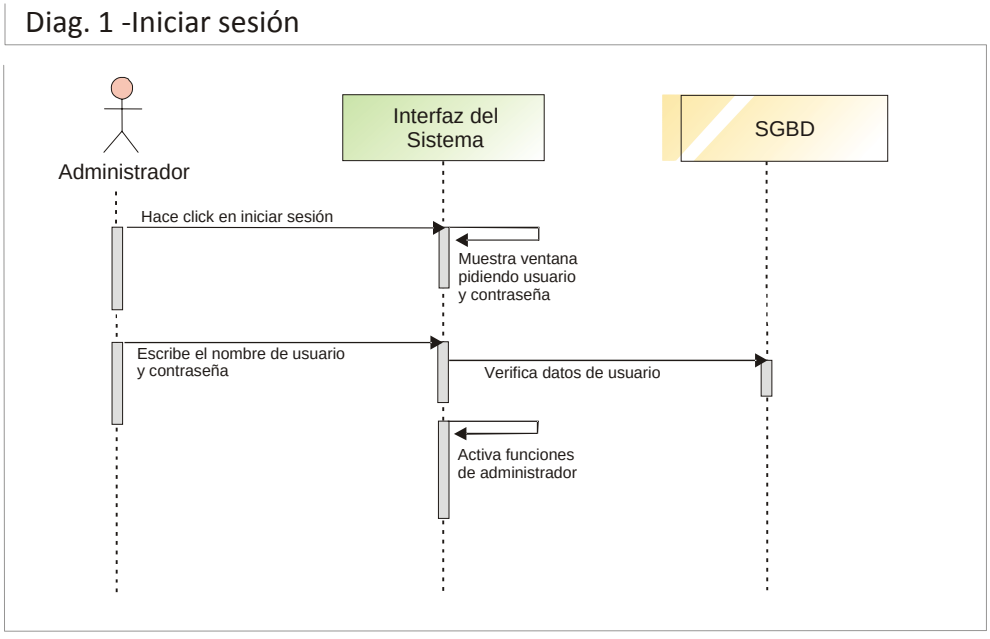
Nombre	Generar Pedido
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador Botón de Pedidos	
Descripción El administrador puede generar pedidos según el proveedor	
Actores Usuario Administrador	
Precondiciones El usuario debe haber iniciado sesión como administrador	
Flujo Normal 1. El usuario hace click en el botón Pedidos 2. El sistema muestra una ventana indicando al usuario que defina el mínimo de piezas en existencia por producto para ser consideradas en el pedido y el proveedor. 3. El usuario proporciona los datos y hace click en generar 4. El sistema genera el pedido según los criterios dados por el usuario 5. El usuario decide si debe guardar o imprimir la lista de pedido	
Flujo Alternativo 2. El usuario cancela la acción 5. El usuario cierra la ventana del pedido	

Nombre	Vender
Autor	Misael Contreras
Fecha	4/Nov/2007
Disparador Botón Vender	
Descripción El usuario puede realizar una venta o cotización e imprimir nota de venta	
Actores Administrador ó Encargado de Mostrador	
Flujo Normal 1. El usuario hace click en el botón Nueva Venta 2. El sistema muestra una ventana donde el usuario podrá ir seleccionando el artículo y definiendo la cantidad de piezas, el tipo de precio, el cliente, e indicará si es una venta o cotización 3. Cuando el usuario termina de agregar productos, hace click en 'terminar venta' donde el sistema mostrará una ventana con el total y un campo donde el usuario de manera opcional puede escribir el efectivo de manera que el programa calcule el cambio automáticamente 4. El sistema imprime el comprobante de venta	
Flujo Alternativo 2. El usuario hace click en 'limpiar' y se eliminan los datos de la venta actual	

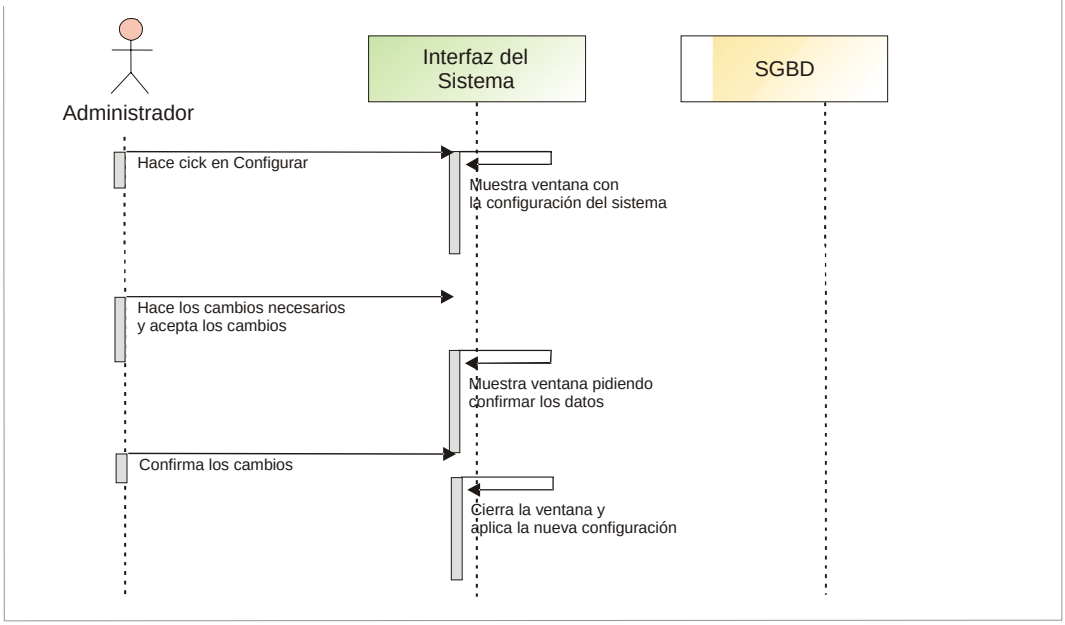
2.6.2 Diagrama de Casos de Uso.



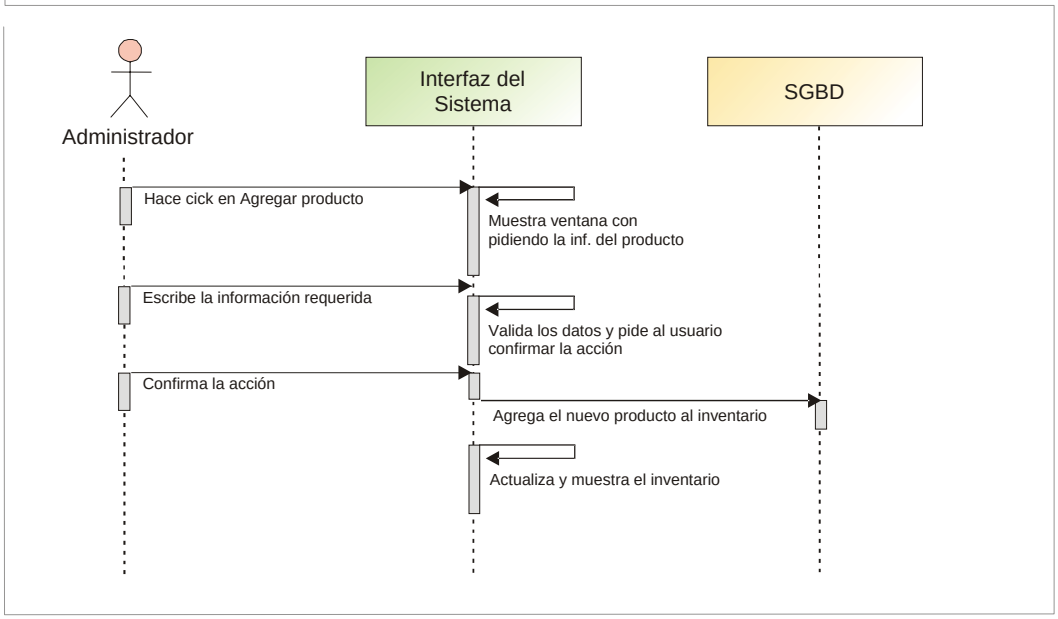
2.6.3 Modelo de Diseño: Diagramas de secuencia.



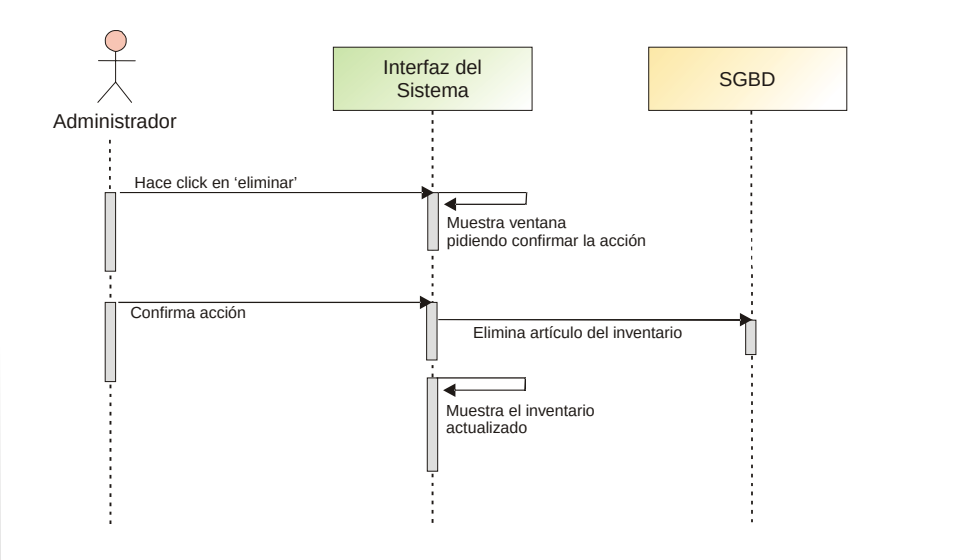
Diag. 2 - Configurar



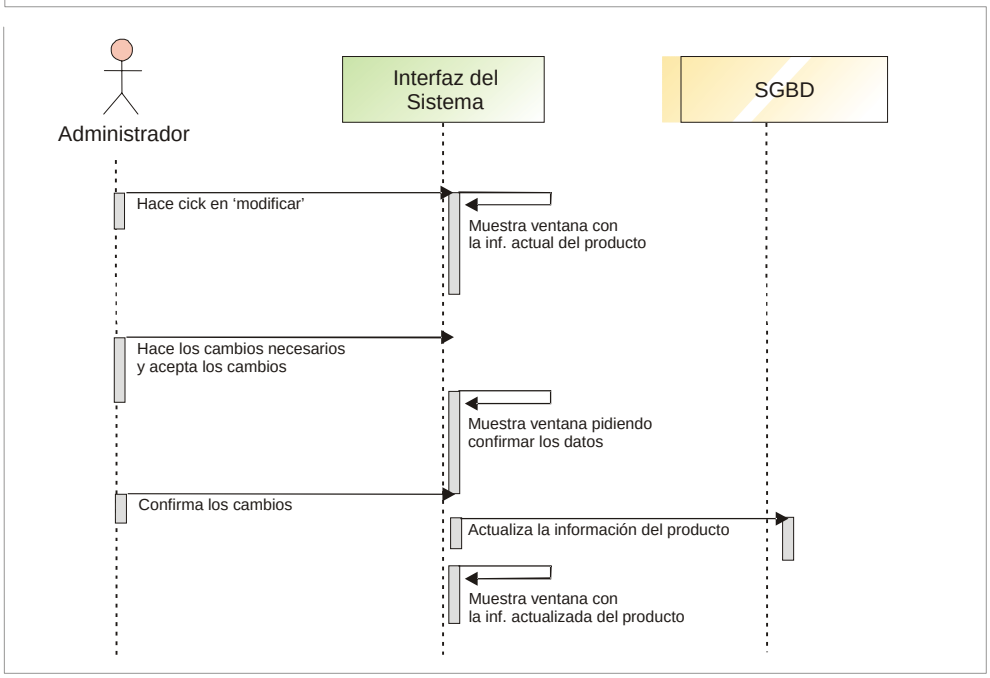
Diag. 3 - Agregar Producto



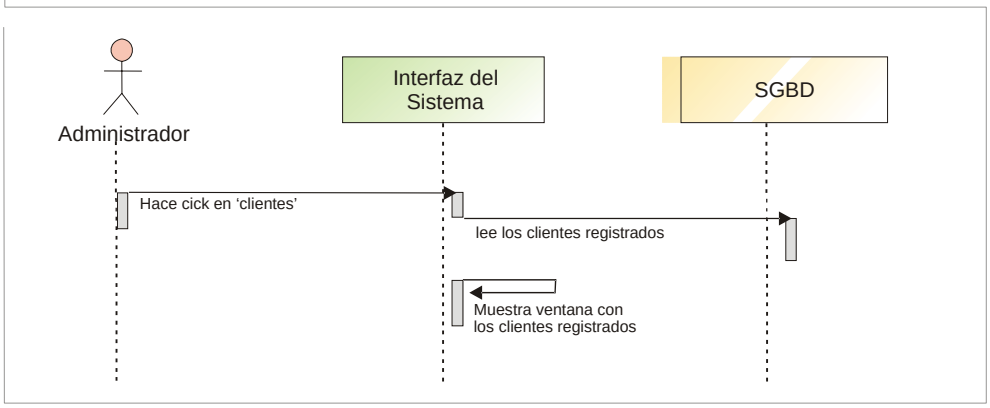
Diag. 4 -Eliminar productos



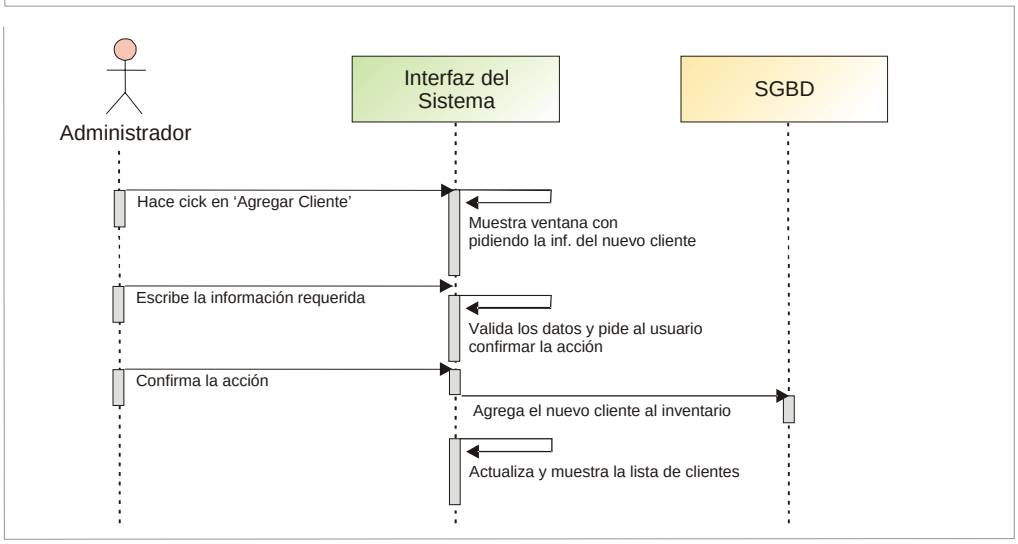
Diag. 5 - Actualizar inf. deproductos



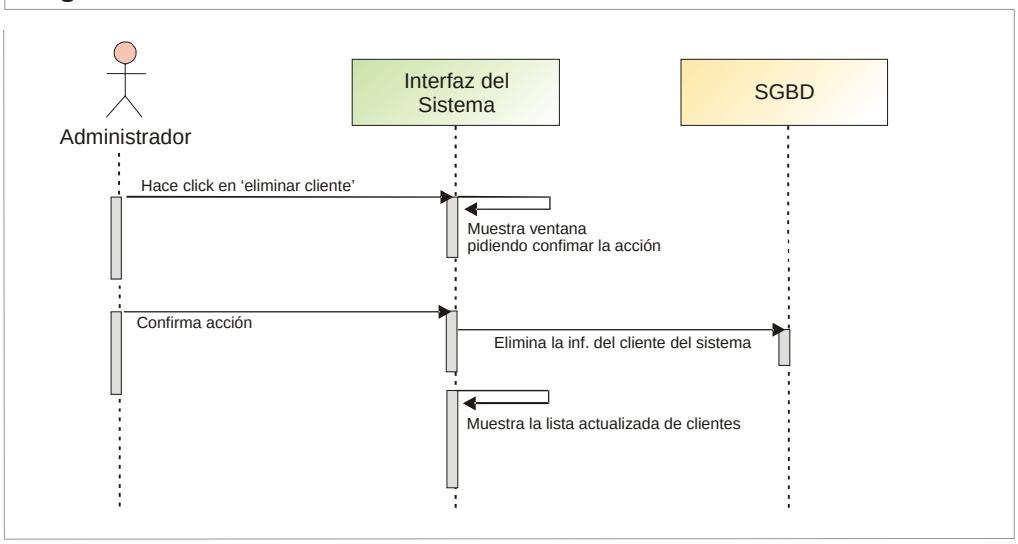
Diag. 6 - Administrar Clientes



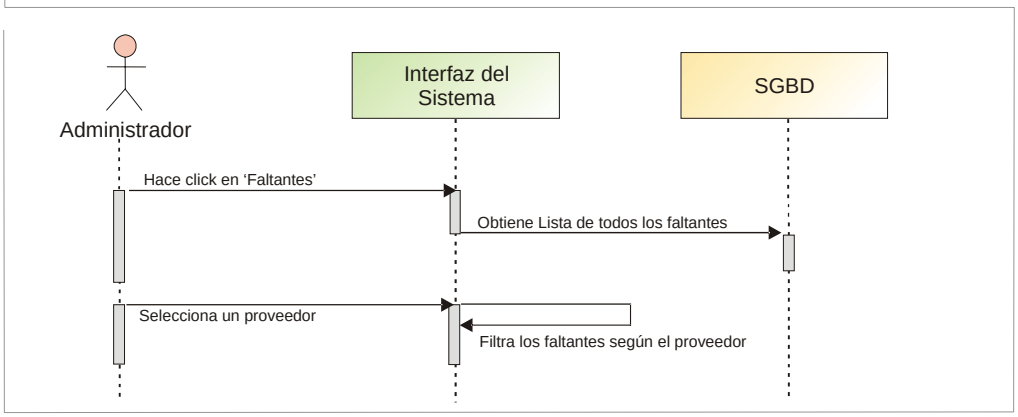
Diag. 7 - Agregar Cliente



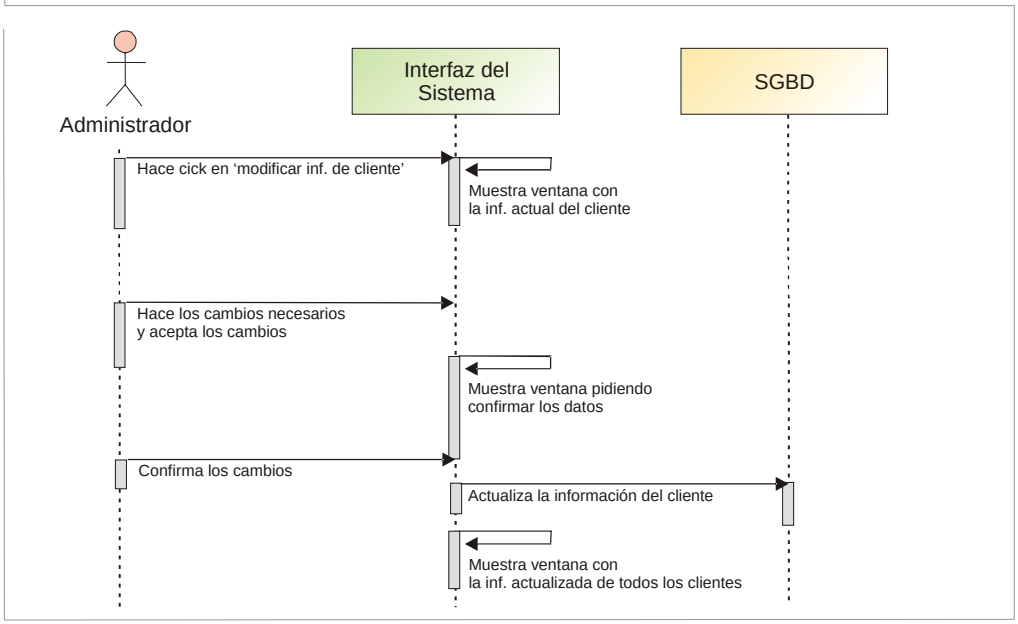
Diag. 8 - Eliminar clientes



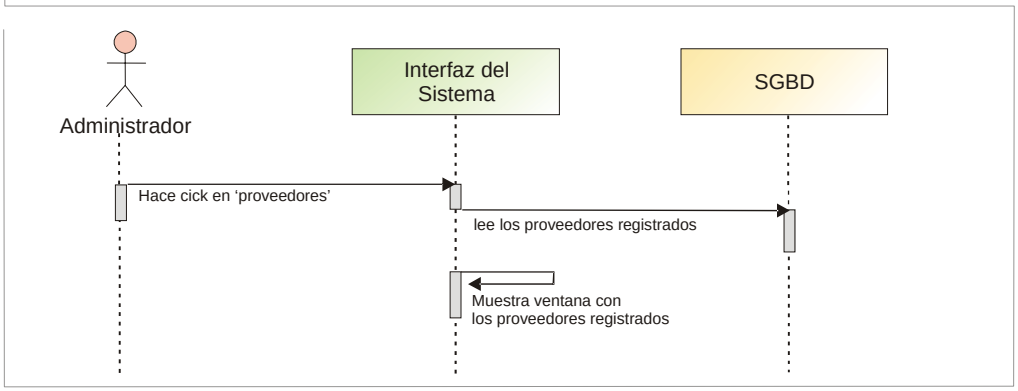
Diag. 9 - Faltantes



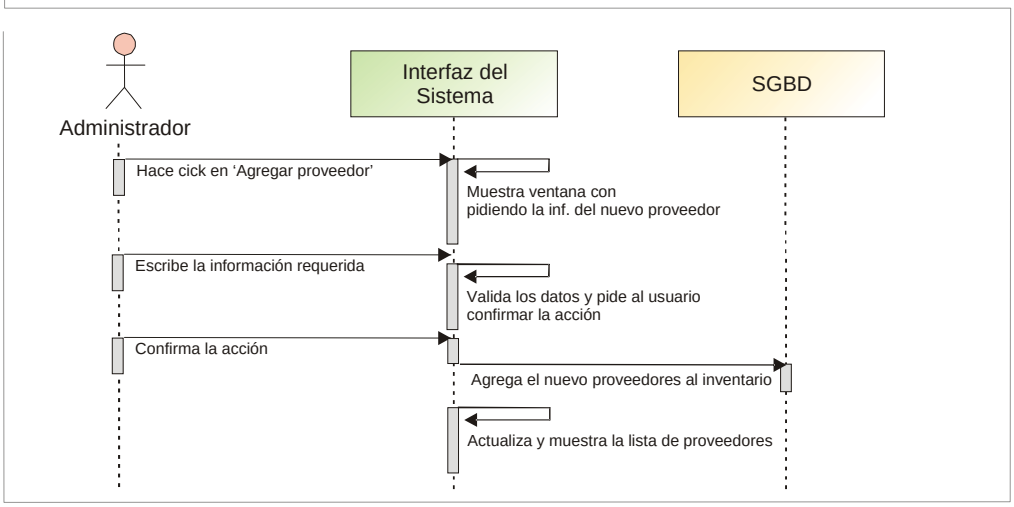
Diag. 10 - Actualizar Inf. de un Cliente



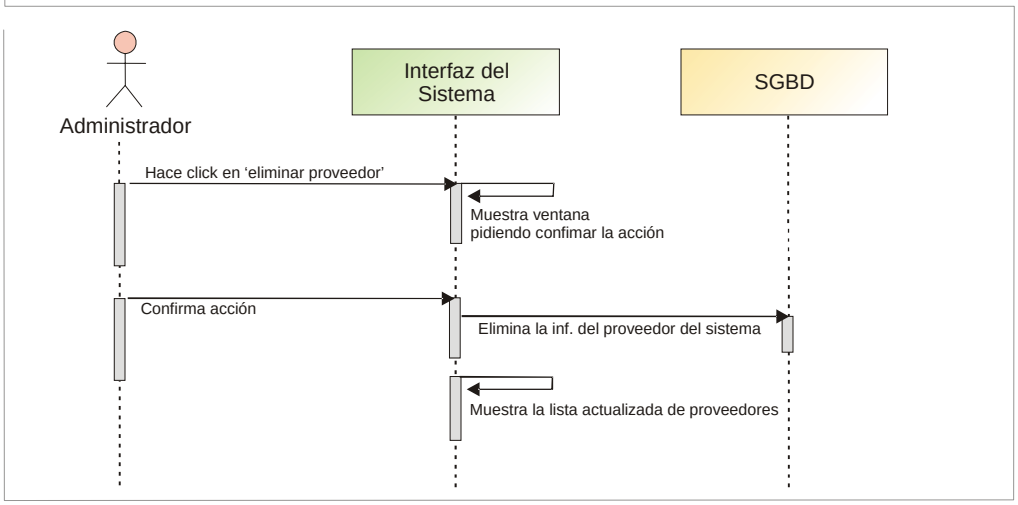
Diag. 11 - Administrar Proveedores



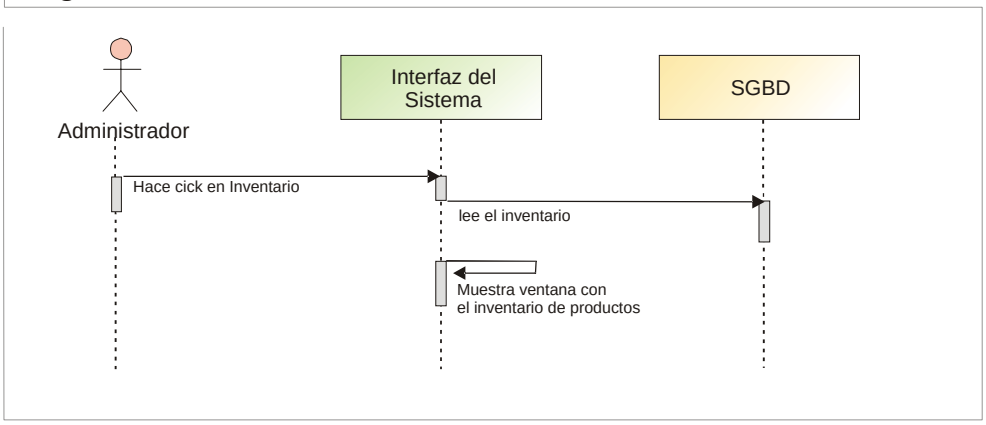
Diag. 12 - Agregar un proveedor



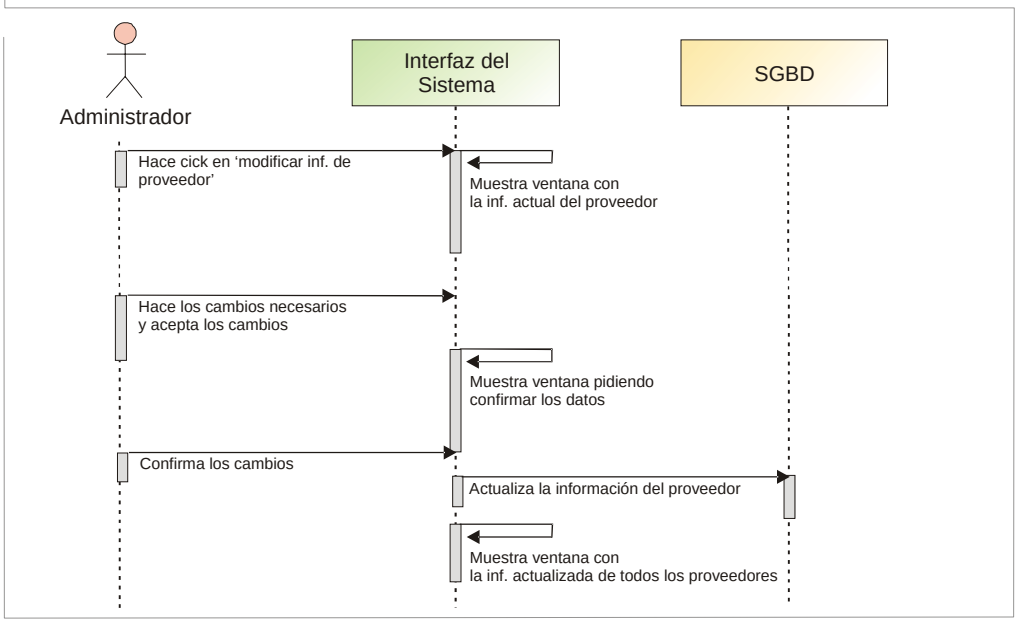
Diag. 13 - Eliminar un proveedor



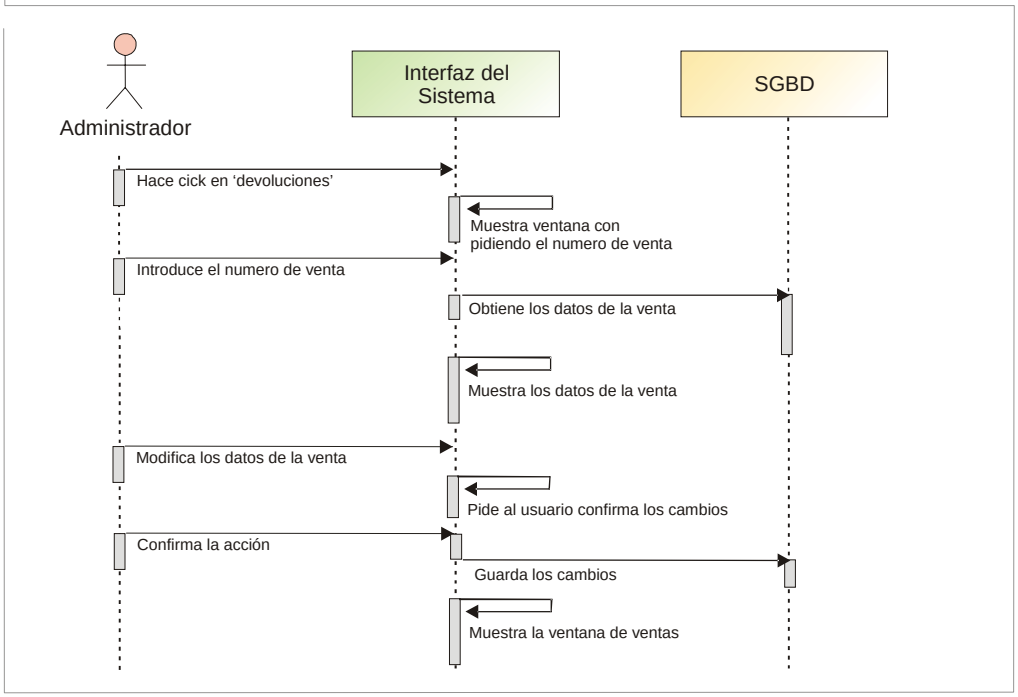
Diag. 14 - Administrar Inventario



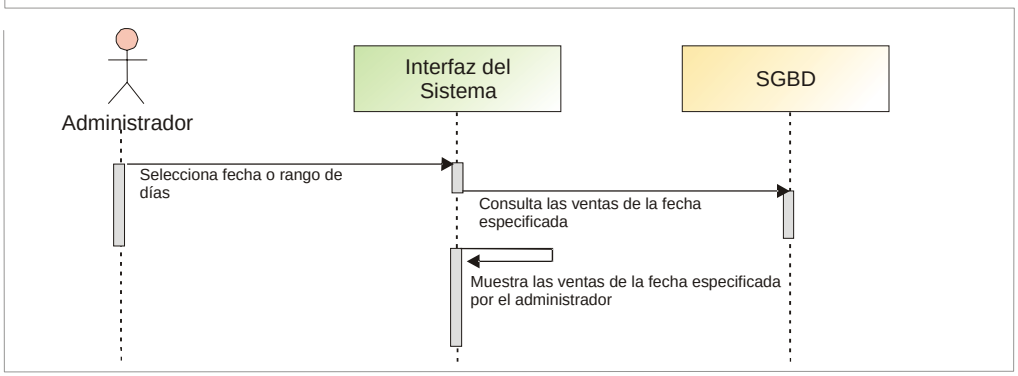
Diag. 15 - Actualizar la inf. de un proveedor



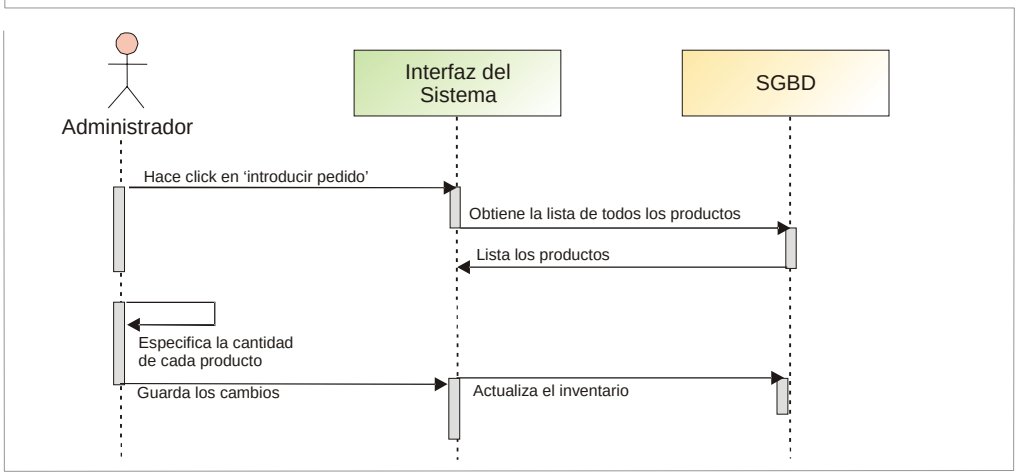
Diag. 16 - Devoluciones



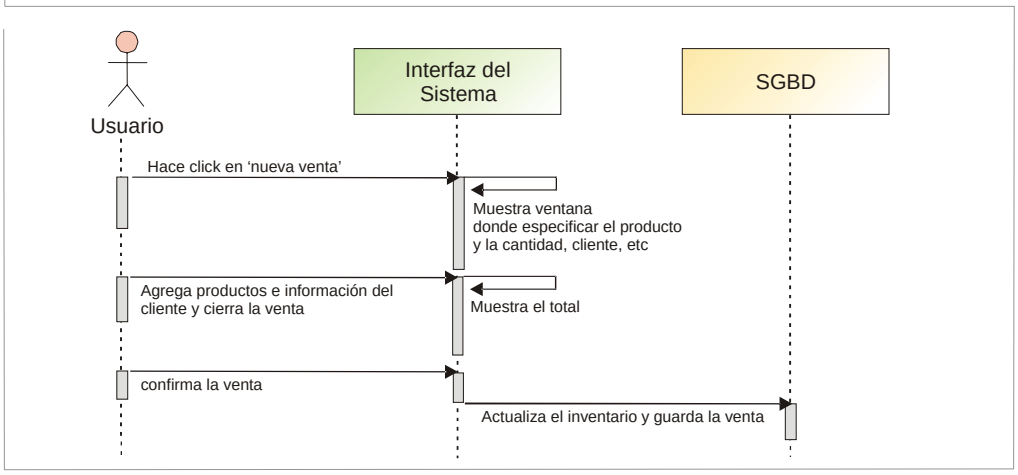
Diag. 17 -Reportes



Diag. 18 - Introducir Pedido



Diag. 19 - Venta



Capitulo 3

3 Diseño del Sistema

CAPITULO 3.
Diseño del Sistema

3.1. Diseño Conceptual

3.1.1 Entidades:

- **Productos**: Clave, descripción, notas, id_proveedor$_{FK}$, almacén, exhibición, pecio_compra, precio_publico, precio_mayoreo, precio_especial, faltante, categoría$_{FK,}$ Marca$_{FK}$,
- **Categorías**: clave, nombre
- **Proveedores**: Id_proveedor, empresa, contacto, dirección, teléfono_empresa, teléfono_contacto, fax_empresa, fax_contacto, numero_ cliente, e-mail
- **Clientes**: Id_cliente, nombre, dirección, teléfono
- **Ventas**: Id_venta, fecha, cliente$_{FK}$, Vendedor $_{FK}$
- **Pedidos**: Id_producto $_{FK}$, id_venta $_{FK}$, Cantidad, precio_compra, precio venta, Tipo_precio$_{FK}$
- **Vendedores:** id_vendedor, nombre, usuario, pwd, permisos
- **Precios:** clavep, nombre
- **Marcas:** clave, nombre

3.1.2 Relaciones*:*

- ✓ Una Venta *es hecha a un* Cliente
- ✓ Una Venta *es hecha por un* Vendedor
- ✓ Una Venta *contiene uno o más* pedidos
- ✓ Un Pedido *se compone de* productos
- ✓ Un pedido *es vendido a un tipo de* precio
- ✓ Un producto *se clasifica en una* categoría
- ✓ Un producto *es de* una marca
- ✓ Un Proveedor *provee* productos

3.2 Diseño Lógico

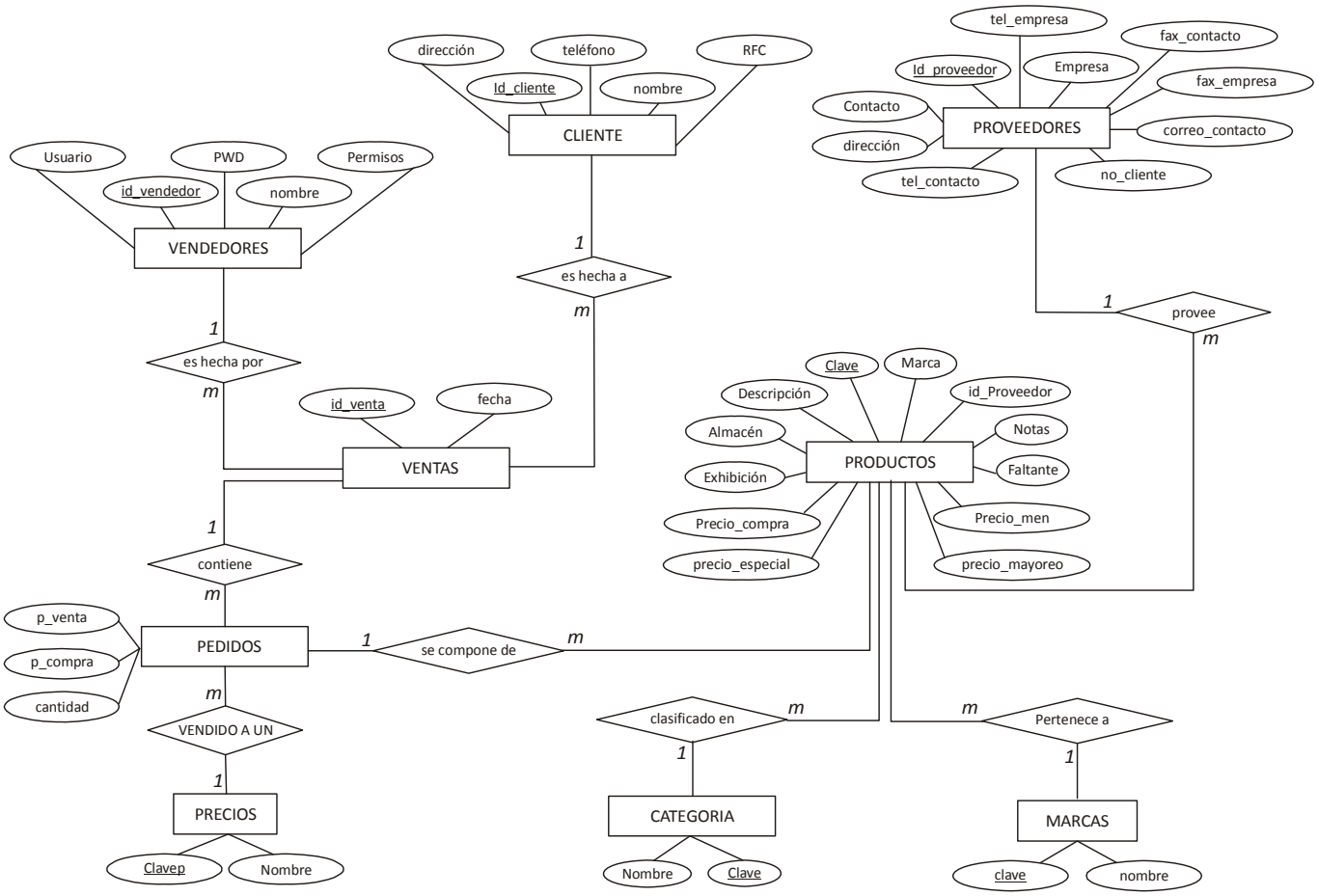


Fig.3.1 – Diseño lógico de la base de datos

3.3 Diagrama Entidad Relación

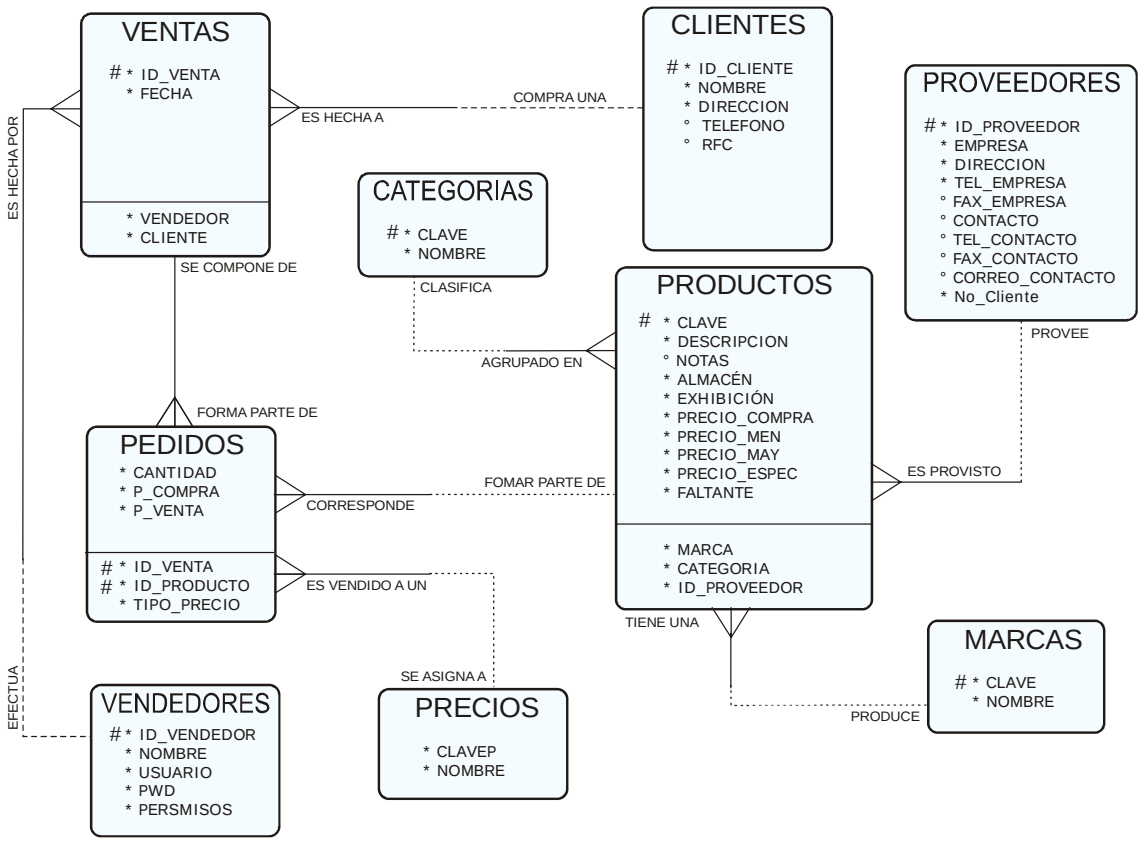


Fig. 3.2 – Diagrama ER

3.4 Normalización

3.4.1 Diagrama de Dependencia Funcional

VENTA
id_venta → fecha, monto, tipo
PEDIDO
Id_venta, id_producto → cantidad, tipo_precio, p_compra, p_venta

CLIENTE
id_cliente → nombre, dirección, teléfono, rfc

PRODUCTOS
clave → descripción, marca, notas, almacén, exhibición, precio_compra, precio_men
precio_may, precio_espec

CATEGORIA
clave → nombre

PROVEEDOR

id_proveedor → empresa, dirección, tel_empresa, fax_empresa, contacto, tel_contacto, fax_contacto, correo_contacto, no_cliente

3.4.2 Formas Normales

1. Comprobación de la primera forma Normal: Todos los campos no-clave son totalmente dependientes de la clave primaria.
2. Comprobación de la segunda forma normal: No hay atributos funcionalmente dependientes de otras no - claves.
3. Comprobación de la tercera forma normal: Ninguna no-clave es transitivamente dependiente de otras no-claves.
4. Forma de Boyce/Codd: Cada determinante es clave aspirante (de hecho, cada determinante es llave primaria).

3.5 Diccionario de Datos

Entidad: Productos

Objetivo: Tabla propia del sistema en la cual se guardan los datos de cada artículo y servicio que se ofertan en el establecimiento, esta es la información más importante para nuestro sistema, ya que de la información de esta tabla derivaremos muchos casos de uso.

Nombre	Descripción del Campo	Tipo	Tamaño
Clave	Contiene un identificador único para cada producto o servicio	Caracter	7
Descripción	Nombre detallado del producto	Caracter	60
Marca	Nombre del fabricante	Caracter	15
Notas	Observaciones relevantes sobre el artículo	Caracter	60
Id_Proveedor	Clave del proveedor que surte esta pieza	Entero	2
Almacén	Existencias del producto en el almacén	Entero	6
Exhibición	Existencias del producto en exhibición	Entero	2
Precio_Compra	Precio al que la empresa adquiere el producto del proveedor	Real	-
Precio_men	Precio de venta menudeo	Real	-
Precio_May	Precio de venta mayoreo	Real	-
Precio_Esp	Precio de venta especial	Real	-
Categoría	Organización de los artículos	Entero	2
Faltante	Menos de estas piezas en existencia el producto se considerará como faltante	Entero	2

Entidad: Proveedores

Objetivo: Tabla propia del sistema que almacena los datos de cada proveedor para generar pedidos.

Nombre	Descripción del campo	Tipo	Tamaño
Id_proveedor	Identificador único para cada proveedor	Entero	2
Empresa	Nombre del proveedor	Caracter	30
Contacto	Nombre de la persona que representa a la empresa	Caracter	40
Dirección	Dirección del proveedor	Caracter	60
Tel_empresa	Teléfono de la empresa	Caracter	18
Tel_contacto	Teléfono particular del contacto	Caracter	18
Fax_empresa	Fax de la empresa	Caracter	18
Fax_contacto	Fax particular del contacto	Caracter	18
correo	Correo electrónico del contacto	Caracter	40
No_cliente	Mi número de cliente	Entero	6

Entidad: Clientes

Objetivo: Tabla propia del sistema que almacena los datos de los clientes

Nombre	Descripción del campo	Tipo	Tamaño
Id_cliente	Identificador único para cada cliente	Entero	4
Nombre	Nombre del cliente	Caracter	40
Dirección	Dirección del cliente	Caracter	55
Teléfono	Teléfono del cliente	Caracter	18
RFC	Registro federal de causantes	Caracter	13

Entidad: Ventas

Objetivo: Tabla propia del sistema que almacena los datos de todas las ventas obtenidas.

Nombre	Descripción del campo	Tipo	Tamaño
Id_venta	Identificador único para cada venta	Entero	6
Fecha	Fecha de compra	Fecha	-
Cliente	Numero de cliente (llave foránea)	Entero	4
Vendedor	Número del vendedor (llave foránea)	Entero	2

Entidad: Categorías

Objetivo: Tabla propia del sistema que almacena los nombres de las categorías en que se organizan los productos.

Nombre	Descripción del campo	Tipo	Tamaño
clave	Identificador único para cada categoría	Entero	2
nombre	Nombre de la categoría	caracter	25

Entidad: Pedidos

Objetivo: Tabla propia del sistema que almacena cada pedido, se le incluye el identificador de la venta al que pertenece y la clave del artículo, las cuales fungen como llave primara

Nombre	Descripción del campo	Tipo	Tamaño
Id_venta	Numero de venta al que pertenece el pedido	Entero	6
Id_producto	Clave del producto vendido	Caracter	7
Cantidad	Artículos vendidos	Entero	4
Tipo_Precio	Precio al que fue vendido (mayoreo, normal, espec)	entero	1
P_compra	Precio al que se obtuvo el artículo	Real	-
P_venta	Precio al que se vendió	Real	-
Precio	Tipo de precio al que vendió	Entero	1

Entidad: Vendedores

Objetivo: Tabla propia del sistema que almacena la información acerca de los cajeros o vendedores. Cuando un cajero quiere acceder al sistema debe identificarse con un nombre de usuario y contraseña. Los permisos indicarán que puede y que no puede hacer dentro del sistema.

Nombre	Descripción del campo	Tipo	Tamaño
Id_vendedor	Llave primaria	Entero	2
Nombre	Nombre completo del vendedor	Caracter	45
Usuario	Usuario con el que entra el sistema	Caracter	20
Pwd	contraseña	Caracter	12
Permisos	Indica los permisos dentro del sistema	Entero	1

Entidad: Precios

Objetivo: Tabla propia del sistema que almacena el tipo de precio y su nombre. Aunque es poca la información que contiene, se utiliza para evitar inconsistencias sobre las tablas de pedidos y productos en caso de actualizar cualquiera de estas dos.

Nombre	Descripción del campo	Tipo	Tamaño
clavep	Identificador único para cada tipo de precio	Entero	1
nombre	Nombre que se mostrará	caracter	9

Entidad: Marcas

Objetivo: Tabla propia del sistema que almacena el tipo una clave y el nombre de una marca. Esta se utilizará para aplicar filtros sobre la tabla de productos.

Nombre	Descripción del campo	Tipo	Tamaño
Clave	Identificador único marca	Entero	2
nombre	Nombre de la marca	caracter	45

3.6 Implementación de la Base de datos.

El sistema se implementó sobre el gesto de bases de datos MySQL. Utilizando el diseño de la base de datos, y el diccionario de datos en conjunto con MySQL Administrator (del que hablaremos en el capítulo 4.1), tenemos que la implantación de la base de datos queda como se muestra en la figura 3.3.

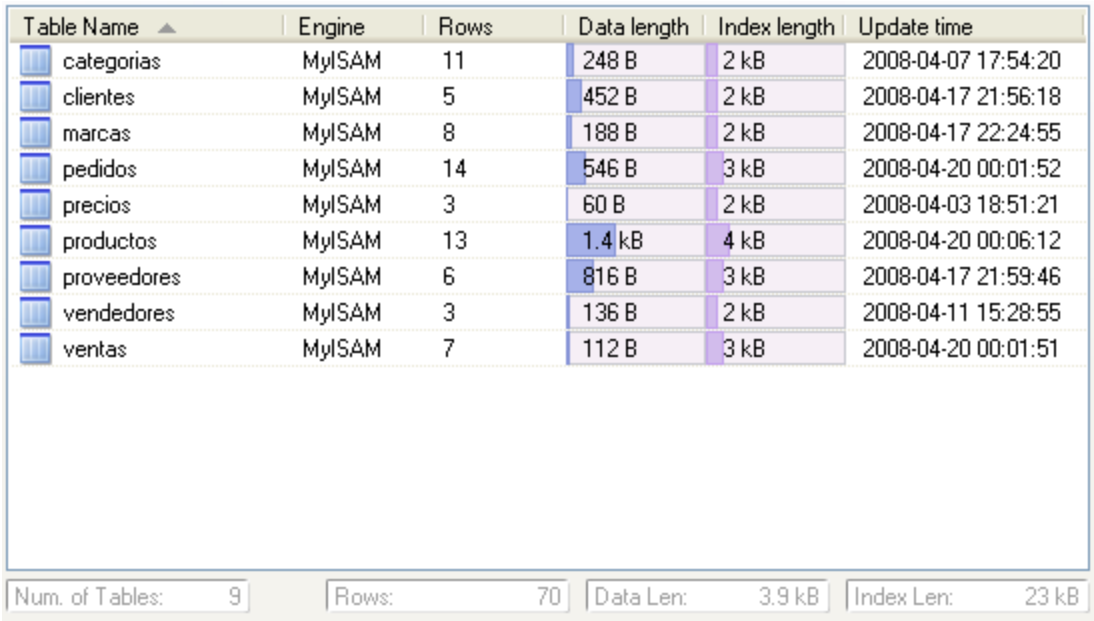
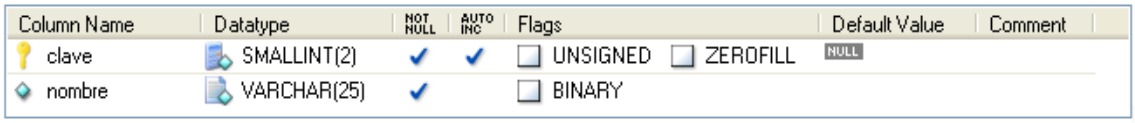


Table Name	Engine	Rows	Data length	Index length	Update time
categorias	MyISAM	11	248 B	2 kB	2008-04-07 17:54:20
clientes	MyISAM	5	452 B	2 kB	2008-04-17 21:56:18
marcas	MyISAM	8	188 B	2 kB	2008-04-17 22:24:55
pedidos	MyISAM	14	546 B	3 kB	2008-04-20 00:01:52
precios	MyISAM	3	60 B	2 kB	2008-04-03 18:51:21
productos	MyISAM	13	1.4 kB	4 kB	2008-04-20 00:06:12
proveedores	MyISAM	6	816 B	3 kB	2008-04-17 21:59:46
vendedores	MyISAM	3	136 B	2 kB	2008-04-11 15:28:55
ventas	MyISAM	7	112 B	3 kB	2008-04-20 00:01:51

Num. of Tables: 9 | Rows: 70 | Data Len: 3.9 kB | Index Len: 23 kB

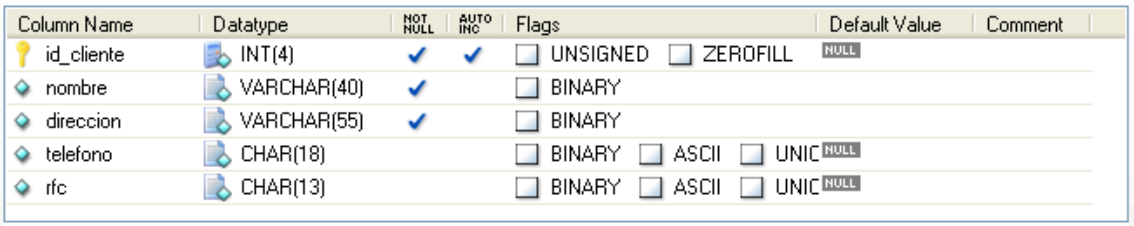
Fig. 3.3 – Implementación de la BD con MySQL Administrator

De la Fig. 3.4 a la fig. 3.12 se muestra a detalle cada tabla de la Base de Datos SISTAR vista desde MySQL Administrator.



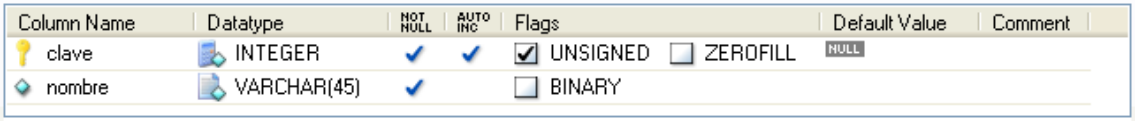
Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
clave	SMALLINT(2)	✓	✓	UNSIGNED ZEROFILL	NULL	
nombre	VARCHAR(25)	✓		BINARY		

Fig. 3.4 – La tabla Categorías



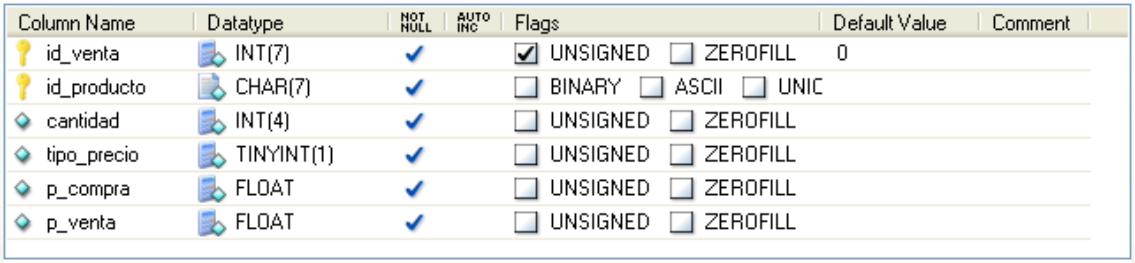
Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
id_cliente	INT(4)	✓	✓	UNSIGNED ZEROFILL	NULL	
nombre	VARCHAR(40)	✓		BINARY		
direccion	VARCHAR(55)	✓		BINARY		
telefono	CHAR(18)			BINARY ASCII UNIC	NULL	
rfc	CHAR(13)			BINARY ASCII UNIC	NULL	

Fig. 3.5 – La tabla Clientes



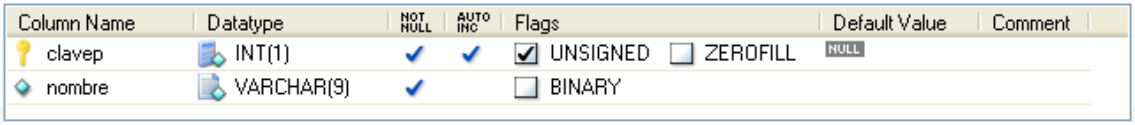
Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
clave	INTEGER	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL	
nombre	VARCHAR(45)	✓		☐ BINARY		

Fig. 3.6 - La tabla Marcas



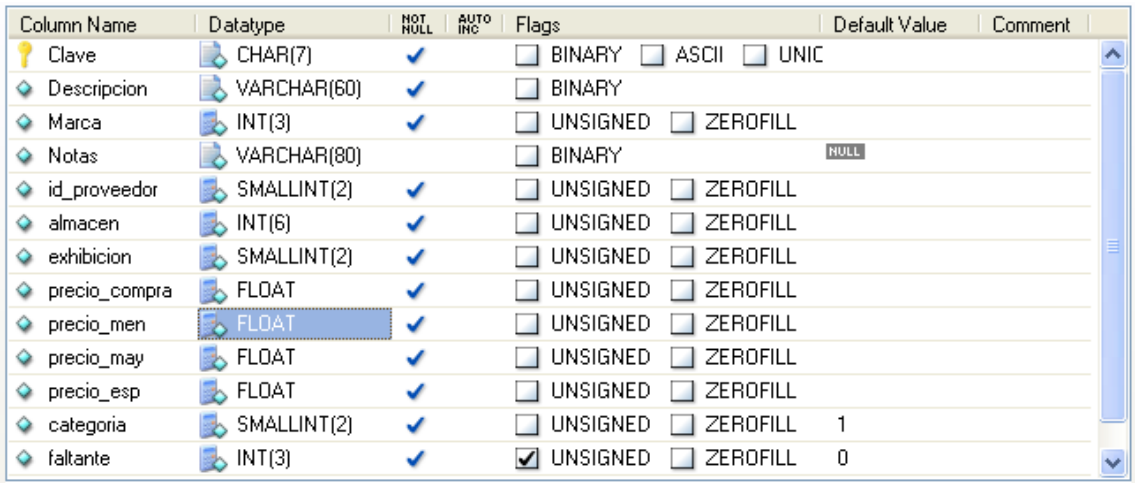
Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
id_venta	INT(7)	✓		☑ UNSIGNED ☐ ZEROFILL	0	
id_producto	CHAR(7)	✓		☐ BINARY ☐ ASCII ☐ UNIC		
cantidad	INT(4)	✓		☐ UNSIGNED ☐ ZEROFILL		
tipo_precio	TINYINT(1)	✓		☐ UNSIGNED ☐ ZEROFILL		
p_compra	FLOAT	✓		☐ UNSIGNED ☐ ZEROFILL		
p_venta	FLOAT	✓		☐ UNSIGNED ☐ ZEROFILL		

Fig. 3.7 – La tabla Pedidos



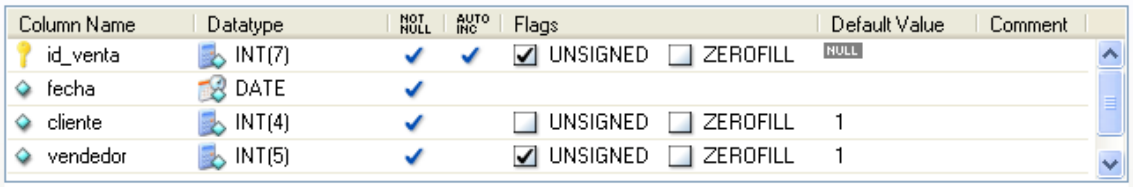
Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
clavep	INT(1)	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL	
nombre	VARCHAR(9)	✓		☐ BINARY		

Fig. 3.8 – La tabla Precios



Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
Clave	CHAR(7)	✓		☐ BINARY ☐ ASCII ☐ UNIC		
Descripcion	VARCHAR(60)	✓		☐ BINARY		
Marca	INT(3)	✓		☐ UNSIGNED ☐ ZEROFILL		
Notas	VARCHAR(80)			☐ BINARY	NULL	
id_proveedor	SMALLINT(2)	✓		☐ UNSIGNED ☐ ZEROFILL		
almacen	INT(6)	✓		☐ UNSIGNED ☐ ZEROFILL		
exhibicion	SMALLINT(2)	✓		☐ UNSIGNED ☐ ZEROFILL		
precio_compra	FLOAT	✓		☐ UNSIGNED ☐ ZEROFILL		
precio_men	FLOAT	✓		☐ UNSIGNED ☐ ZEROFILL		
precio_may	FLOAT	✓		☐ UNSIGNED ☐ ZEROFILL		
precio_esp	FLOAT	✓		☐ UNSIGNED ☐ ZEROFILL		
categoria	SMALLINT(2)	✓		☐ UNSIGNED ☐ ZEROFILL	1	
faltante	INT(3)	✓		☑ UNSIGNED ☐ ZEROFILL	0	

Fig. 3.9 – La tabla Productos



Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
id_venta	INT(7)	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL	
fecha	DATE	✓				
cliente	INT(4)	✓		☐ UNSIGNED ☐ ZEROFILL	1	
vendedor	INT(5)	✓		☑ UNSIGNED ☐ ZEROFILL	1	

Fig. 3.10 – Tabla Ventas

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
id_proveedor	SMALLINT(2)	✓	✓	☐ UNSIGNED ☐ ZEROFILL	NULL	
empresa	VARCHAR(30)	✓		☐ BINARY		
contacto	VARCHAR(40)			☐ BINARY	NULL	
direccion	VARCHAR(60)	✓		☐ BINARY		
tel_empresa	CHAR(18)	✓		☐ BINARY ☐ ASCII ☐ UNIC		
tel_contacto	CHAR(18)			☐ BINARY ☐ ASCII ☐ UNIC	NULL	
fax_empresa	CHAR(18)			☐ BINARY ☐ ASCII ☐ UNIC	NULL	
fax_contacto	CHAR(18)			☐ BINARY ☐ ASCII ☐ UNIC	NULL	
correo	VARCHAR(40)			☐ BINARY	NULL	
no_cliente	INT(6)	✓		☐ UNSIGNED ☐ ZEROFILL		

Fig. 3.10 – La tabla Proveedores

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
id_vendedor	INT(5)	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL	
nombre	VARCHAR(45)	✓		☐ BINARY		
usuario	VARCHAR(20)	✓		☐ BINARY		
pwd	VARCHAR(12)	✓		☐ BINARY		
permisos	INT(1)	✓		☑ UNSIGNED ☐ ZEROFILL	0	

Fig. 3.11 – Tabla Vendedores

3.7 Diagrama de clases

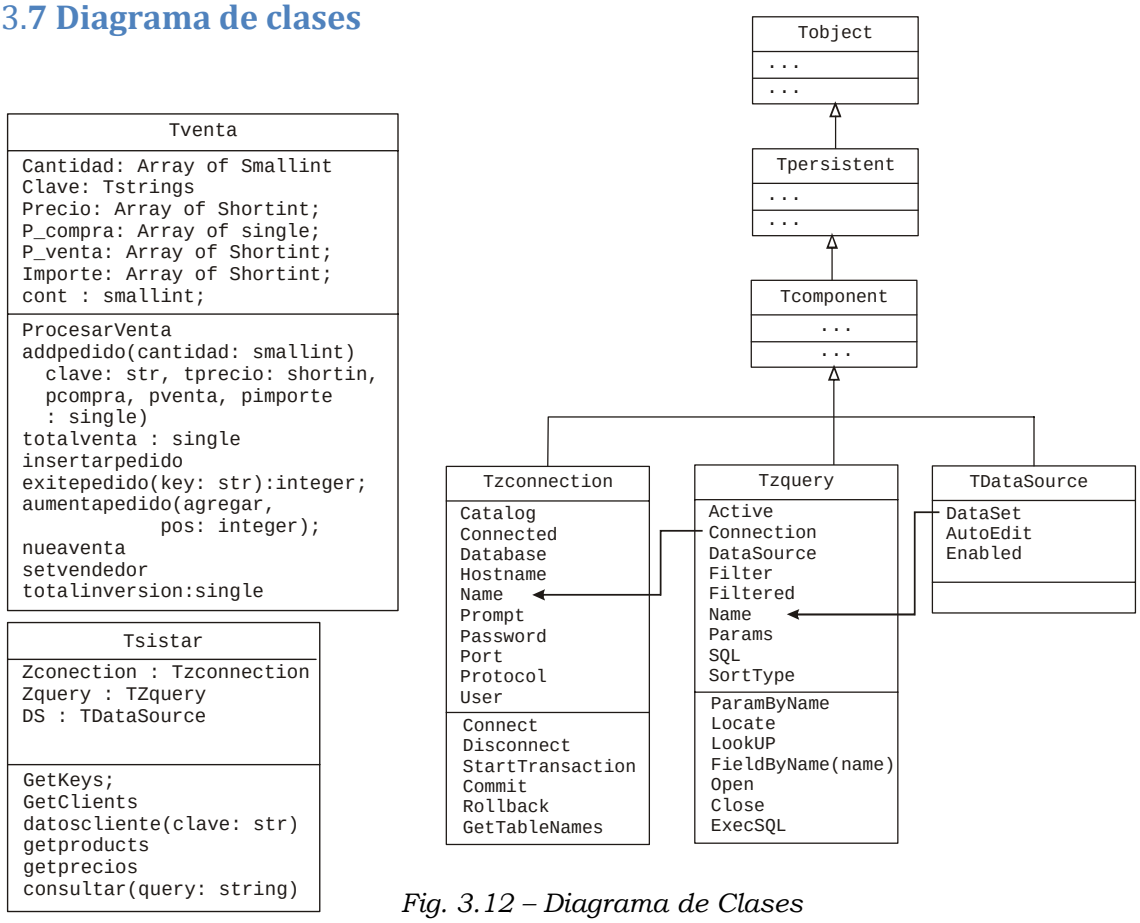


Fig. 3.12 – Diagrama de Clases

En el sistema utilizaremos las clases por defecto que vienen con Delphi más las clases de ZeosLib (disponibles en internet bajo licencia GPL). Las cuales son descendientes de las clases de Delphi TComponent, TPersistent y TObect. TSistar instancia a las 4 clases superiores.

3.8 Diagrama de componentes

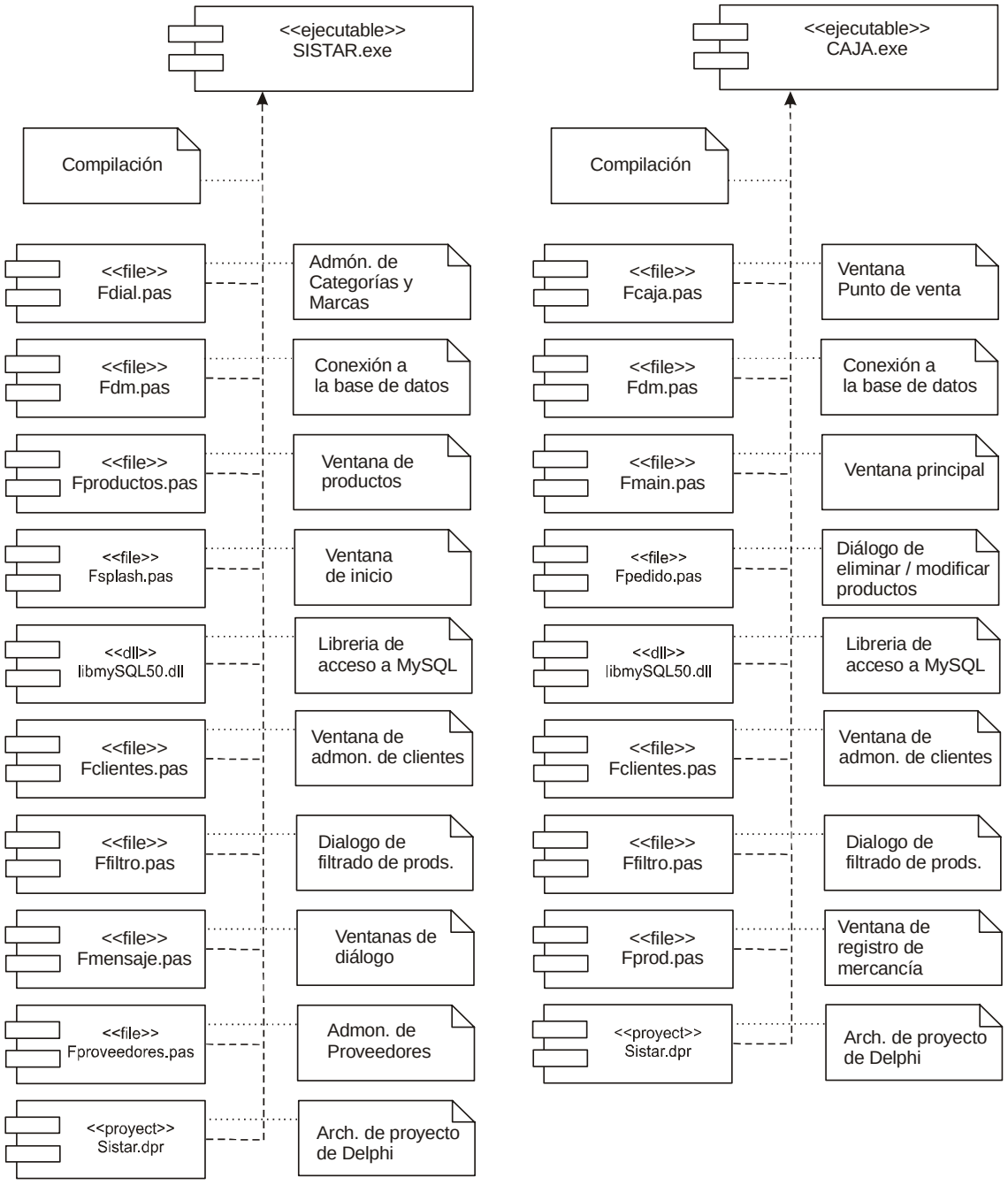


Fig. 3.13 – Diagrama de componentes

3.9 Diseño de interfaces.

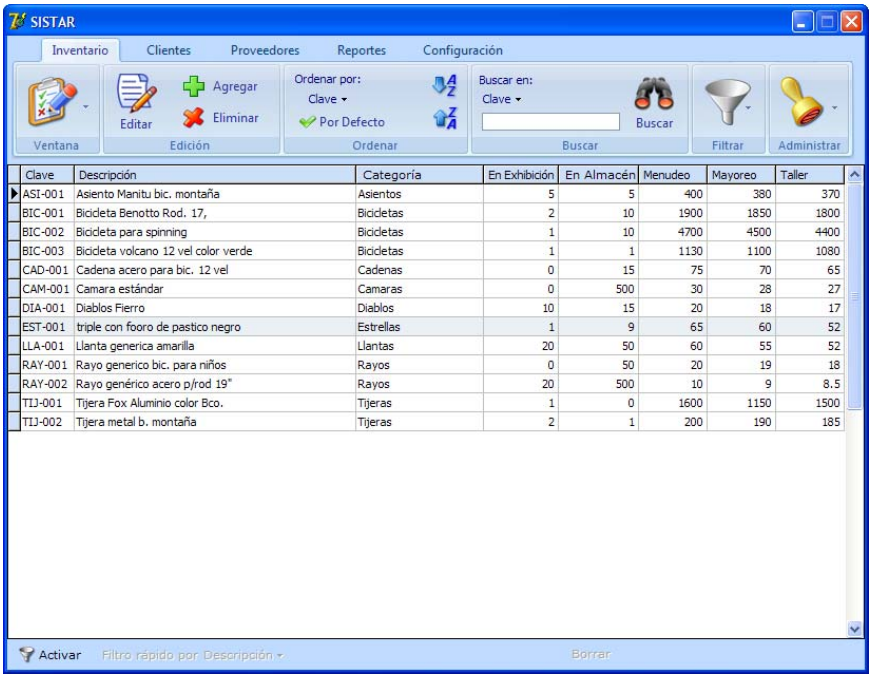
El proyecto produce dos programas ejecutables por separado, aunque trabajan de manera concurrente sobre la base de datos. El programa llamado SISTAR es el programa que lleva a cabo todas las funciones del administrador (ver análisis de requerimientos). El otro programa es CAJA y lleva a cabo todas las funciones de punto de venta.

Para ambos programas, el usuario debe identificarse, cada programa lanza una ventana al inicio donde solicita el nombre de usuario y contraseña. Además se pueden cambiar algunas configuraciones como el IP del servidor, el usuario y contraseña de la base de datos por si esta fuese cambiada de la original (fig 3.14).

3.9.1 Sistar Administrador

Fig.3.14 - Inicio del Sistema.

Esta ventana será diferente en diseño a la de CAJA pero iguales en función. El propósito es hacer más fácil al usuario identificar el programa que ha iniciado.



Clave	Descripción	Categoría	En Exhibición	En Almacén	Menudeo	Mayoreo	Taller
ASI-001	Asiento Manitu bic. montaña	Asientos	5	5	400	380	370
BIC-001	Bicicleta Benotto Rod. 17,	Bicicletas	2	10	1900	1850	1800
BIC-002	Bicicleta para spinning	Bicicletas	1	10	4700	4500	4400
BIC-003	Bicicleta volcano 12 vel color verde	Bicicletas	1	1	1130	1100	1080
CAD-001	Cadena acero para bic. 12 vel	Cadenas	0	15	75	70	65
CAM-001	Camara estándar	Camaras	0	500	30	28	27
DIA-001	Diablos Fierro	Diablos	10	15	20	18	17
EST-001	triple con fooro de pastico negro	Estrellas	1	9	65	60	52
LLA-001	Llanta generica amarilla	Llantas	20	50	60	55	52
RAY-001	Rayo generico bic. para niños	Rayos	0	50	20	19	18
RAY-002	Rayo genérico acero p/rod 19"	Rayos	20	500	10	9	8.5
TIJ-001	Tijera Fox Aluminio color Bco.	Tijeras	1	0	1600	1150	1500
TIJ-002	Tijera metal b. montaña	Tijeras	2	1	200	190	185

Fig. 3.15 - Ventana principal mostrando el inventario.

Una vez que se inicie el programa (cuando se satisfacen las condiciones de seguridad y conectividad) se muestra la pantalla principal (fig. 3.15) desde la cual parte el funcionamiento total de la parte administrativa del sistema.

Esta ventana es visualmente dividida en tres secciones horizontales. En la parte superior tenemos 5 pestañas (tabs) las cuales permite el acceso a cada parte del sistema. Debajo de esta tenemos la barra de herramientas y más abajo tenemos la rejilla con los datos.

Las 5 pestañas permiten acceder a todas las funciones del programa y permiten categorizar la información y las operaciones aplicables a esta. Las pestañas son: Inventario, clientes, proveedores y reportes. Trataremos a continuación cada pestaña con la información que presenta y las operaciones que permite realizar.

1. Inventario.

En la fig. 3.16 vemos el contenido total de la ficha Inventario. Ahora describiremos las funciones de la barra de herramientas.

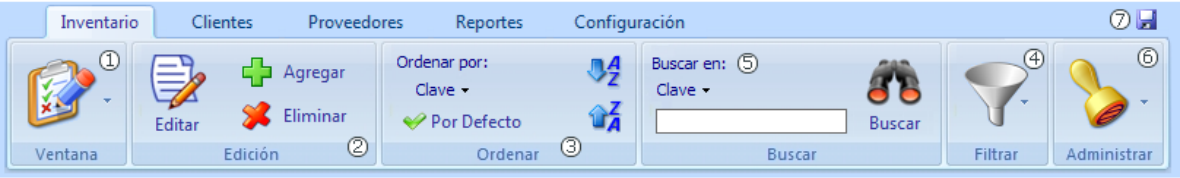


Fig. 3.16 – Barra de herramientas de la pestaña Inventario

Los comandos se organizan en paneles los cuales están etiquetados en la parte inferior de la barra ó cinta de herramientas y delimitados por un recuadro azul. La etiqueta en cada panel permite identificar rápidamente que tipo de acción efectuarán los comandos que contiene. A continuación trataremos cada uno de estos paneles:

1. El panel “ventana” es común a todas las pestañas y permite también intercambiar la vista entre estas. Muestra solo un botón que permite identificar (y cambiar) la pestaña actual.
2. El panel “Edición” (permite editar la información del producto seleccionado en la rejilla de abajo. También permite agregar o eliminar un producto. Los botones de Editar y Agregar al ser pulsados mostrarán una ventana emergente (fig. 3.17) de donde se podrán realizar los cambios deseados sobre la información del producto. El botón Eliminar provocará que se muestre un cuadro de dialogo para confirmar la acción.
3. El panel “ordenar” permite ordenar la información visible en la rejilla. Este comando no afecta el orden en la base de datos. El primer botón muestra un menú para seleccionar la columna sobre la que se va a aplicar el ordenamiento. Los botones a la derecha aplican la operación en orden ascendente y descendente.
4. El panel “filtrar” muestra un único botón que al ser presionado muestra una ventana donde el usuario puede especificar con mayor precisión la información que desea que el programa muestre (fig. 3.18a).

5. El panel “Buscar” permite buscar sobre los datos mostrados en la rejilla. Si el filtro avanzado esta activado solo buscará dentro de los datos visibles. Primero se selecciona la columna, se escribe la información a buscar y se hace clic en el botón de búsqueda.
6. El panel administrar muestra un menú para seleccionar le información que se desea administrar y estas opciones son ‘Categorías’ y ‘Marcas’. Dada la simpleza de esta información se decidió no anexarla entre las opciones las pestañas superiores y se integró como parte de la pestaña de inventario (fig. 3.18b).
7. En la parte superior derecha se encuentra un botón con el ícono de un disco de 3/1”, este comando es común a todas las pestañas y exporta el contenido de la rejilla activa a un archivo de Microsoft Excel.

Fig. 3.17 – Ventana de edición de productos.

a)

b)

Fig. 3.18 – Ventana de Filtro Avanzado y de Marcas

2. Clientes

La ficha clientes permite administrar toda la información relacionada con estos. En la figura 3.19 mostramos los comandos sobresalientes y a continuación describimos brevemente los comandos de la barra de herramientas.

Fig. 3.19 – Barra de herramientas de la pestaña de clientes

1. El panel de edición permite agregar o eliminar y modificar la información acerca del cliente. Para los dos últimos comandos el sistema muestra un cuadro de diálogo para confirmar la acción. El comando de editar muestra una ventana emergente para modificar la información y además eliminar o agregar nuevos clientes (fig. 3.20).

Fig. 3.20 – Ventana de edición de Clientes

2. El panel de ordenar muestra 4 botones cuyos comandos permiten ordenar los datos mostrados en la rejilla. Al hacer clic sobre uno de estos botones se muestra un menú para elegir si los datos serán ordenados en forma ascendente o descendente (fig. 3.21).

Fig. 3.21 – Comandos de orden

3. El panel de búsqueda selecciona la primera fila donde los datos coincidan al criterio de búsqueda.

4. La rejilla de datos muestra toda la información almacenada acerca de cada cliente. El comportamiento de esta es similar a la rejilla de la pestaña inventario. Esto es, cuando el usuario hace un doble clic sobre una fila, se muestra una ventana emergente para poder modificar los datos almacenados acerca de dicho cliente (fig. 3.20).

3. Proveedores.

La pestaña de proveedores (fig. 3.22) permite visualizar y administrar la información de todos los proveedores. Para grandes empresas que tienen agentes de ventas se especifica la información de estos como contactos. De esta manera podemos almacenar la información de la empresa y del contacto (si es que lo tiene, de lo contrario se omite esta información). Dado que la mayoría de las empresas asigna a sus clientes un número, en el sistema podemos almacenar este, de manera que cuando el programa nos genere una lista de pedido, se incluya automáticamente los datos de la empresa proveedora, de la refaccionaria y el número de cliente de esta.

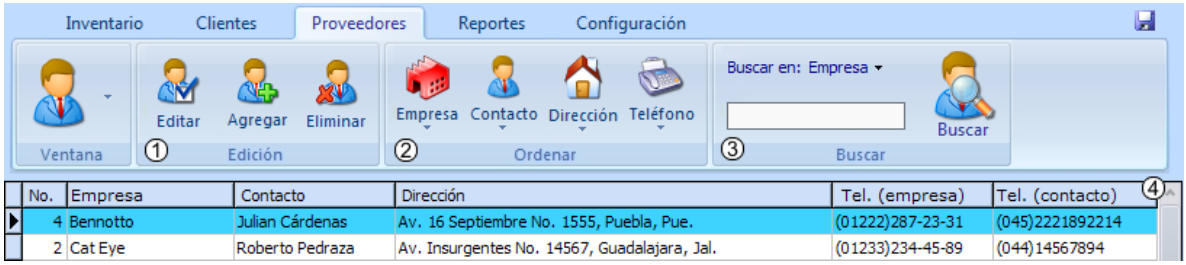


Fig. 3.22 – Barra de herramientas de la pestaña de clientes

A continuación describimos los comandos de la barra de herramientas:

1. El panel de edición funciona de manera similar al de la pestaña de clientes. El primer botón muestra una ventana emergente (fig. 3.23) que permite modificar la información del proveedor. El usuario puede crear un nuevo proveedor o eliminar del sistema el proveedor mostrado.

Fig. 3.23 – Ventana de edición de proveedores

2. Para ordenar los datos tenemos cuatro opciones dentro del este panel. Cada botón funciona de manera similar a los botones contenidos en la pestaña Clientes.
3. El panel de búsqueda localiza dentro de la rejilla el primer resultado que coincida con los dos criterios de búsqueda que se especifican: La columna sobre la que se va a buscar y el texto a buscar.
4. La rejilla funciona de manera análoga a las anteriores salvo que muestra la información de los proveedores.

4. Reportes.

La pestaña de reportes (fig. 3.24) permite ver las ventas detalladamente y los faltantes. La barra de herramientas proporciona comandos para ver las ventas de del día (por defecto), las de la semana, de un día o rango de fechas en específico ó por cliente.

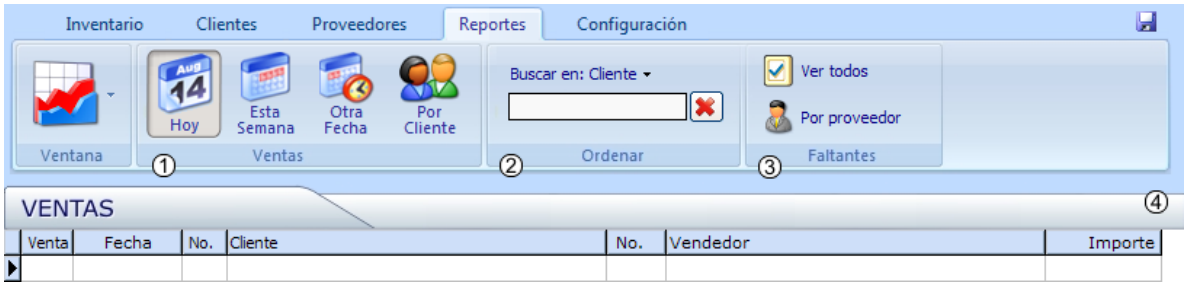


Fig. 3.24 – Barra de herramientas de La pestaña Reportes.

Esta funcionalidad en conjunto con la herramienta de búsqueda permiten hace consultas avanzadas, por ejemplo: ver las ventas de determinada fecha que hizo x vendedor ó las compras realizadas por determinado cliente las cuales hizo x vendedor ó en x días.

La parte de faltantes muestra todos los faltantes ó los muestra por un proveedor en específico con el fin de generar la lista de pedidos. Todos los reportes al igual que los datos de las demás pestañas pueden ser exportados a un archivo de Microsoft Excel ®.

A continuación describiremos a detalle cada comando:

1. El panel de ventas muestra cuatro botones, cada uno de estos afectará la información mostrada en la rejilla de ventas. Cuando accedemos a esta ficha el primer botón (Hoy) siempre está activo por defecto. De esta manera se listarán abajo las ventas realizadas durante el transcurso del día. El siguiente botón permite ver las ventas de la semana. El siguiente botón permite mostrar las ventas de otro día o en un intervalo determinado de tiempo. Al hacer clic el sistema muestra un cuadro de diálogo donde el usuario selecciona la fecha de inicio y fin (fig. 3.25a). Para facilitar la selección de fechas, cada recuadro tiene un botón que al hacer clic muestra un calendario (fig, 3.25b).

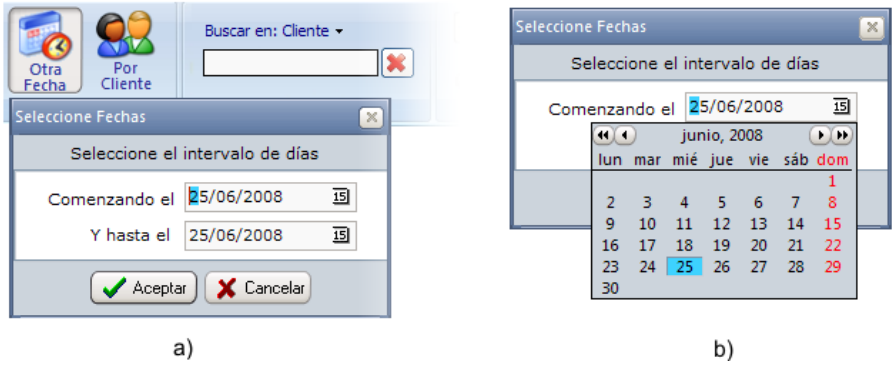


Fig. 3.25 – Mostrar Ventas por otra fecha

El siguiente botón permite ver las ventas realizadas a determinado cliente con la posibilidad de especificar un intervalo de tiempo para la consulta. En la fig. 3.26a se muestra la ventana emergente tras hacer clic sobre el botón. La ventana cuenta con una lista donde el usuario selecciona el nombre del cliente. Debajo de esta tenemos dos botones de radio. Si el primero esta activado los resultados serán la búsqueda de las ventas realizadas a x cliente desde su primera compra registrada. Si se activa el segundo botón de radio, se mostrará dos cajas de edición donde el usuario puede introducir el rango de tiempo sobre el que se va a buscar. Esto permitirá al usuario buscar las ventas hechas a determinado cliente durante el mes de abril de 2008, por ejemplo.

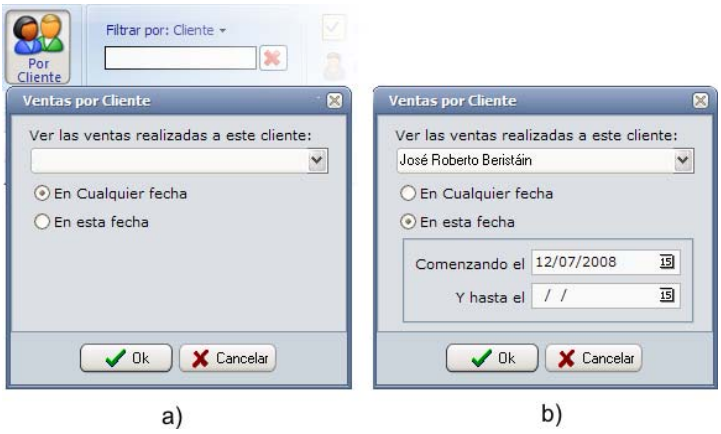


Fig. 3.26 – Mostrar ventas por cliente.

2. El panel de búsqueda realiza un filtrado rápido de los datos cuando se escribe algo dentro de la caja de texto que muestra. Este filtro solo trabaja sobre la información visible. Esto es, si el sistema está mostrando las ventas del día, cual el usuario escriba algo dentro el cuadro de texto, el sistema filtrara la información escrita sobre los datos que está mostrando. Arriba de este cuadro de texto hay un botón que muestra los diferentes criterios sobre los que se puede aplicar el filtro rápido: por cliente, vendedor y fecha.

3. El panel del faltantes muestra dos botones: Ver todos y por proveedor. Cuando el usuario haga clic sobre el botón *ver todos* el sistema muestra todos los faltantes. Si el usuario activa *por proveedor* el sistema mostrará una lista donde el usuario deberá elegir el nombre del proveedor.

5. Configuración

La última pestaña (fig. 3.27) permite ajustar ciertos parámetros del sistema. Básicamente son los vendedores y la dirección por defecto del servidor. Describiremos las secciones más importantes a continuación:

Fig. 3.27 – La pestaña Configuración.

1. En la sección de administrar vendedores se tiene una lista con todos los vendedores en registrados. Un vendedor necesita estar registrado aquí para poder iniciar sesión en SISTAR Caja. Se pueden agregar nuevos vendedores ó eliminar los que están seleccionados. Si se elimina un vendedor, las ventas realizadas por este pasarán a mostrador
2. A la derecha de la lista de vendedores tenemos tres cuadros de texto que permiten ver y editar la información de los vendedores, además de dos botones, uno que permite editar la información y el otro guardar los cambios.
3. En la sección de Servidor por defecto el usuario especifica el IP del servidor donde el sistema se conectará automáticamente al iniciar el programa.

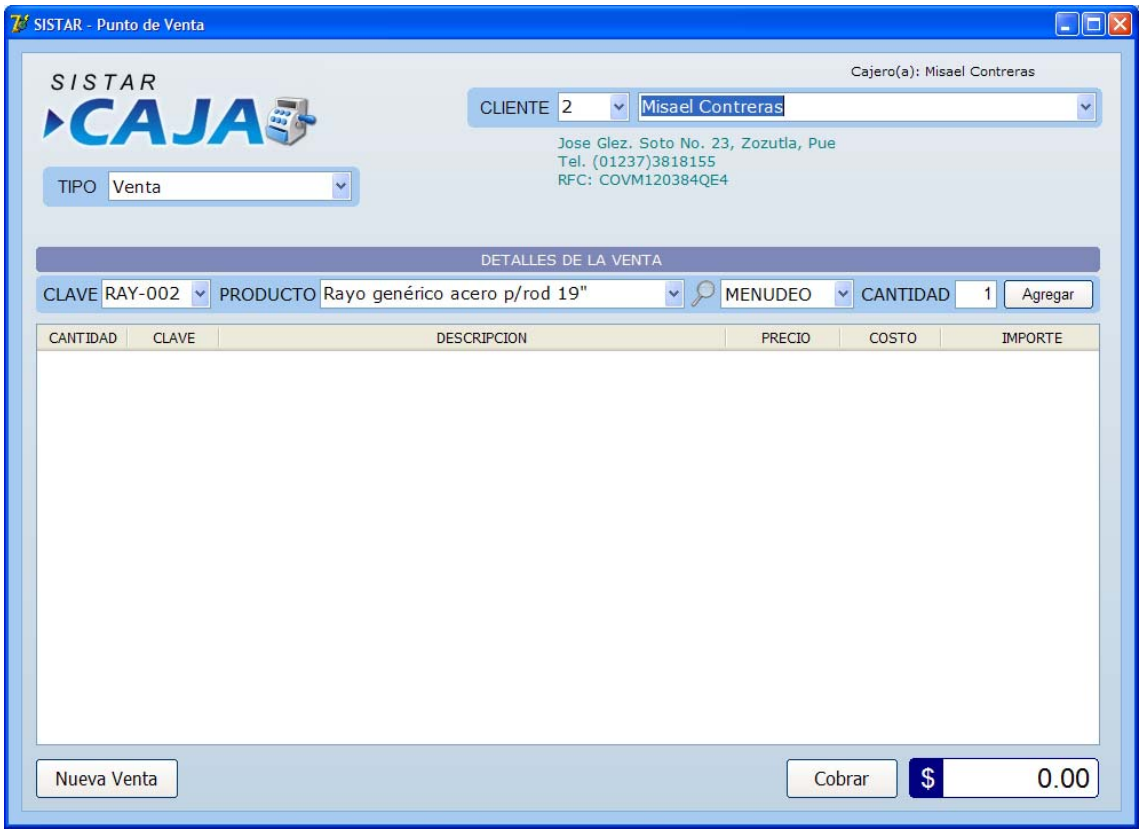
3.9.2 Sistar Caja

El proyecto está dividido en dos programas, del primero hemos visto hasta aquí. El otro programa es “SISTAR caja” que funciona únicamente como punto de venta.

Al iniciar el programa pide, al igual que en el anterior, un nombre de usuario y contraseña. Cada cajero debe identificarse para poder acceder al sistema de modo que para cada venta se registre quien la hizo (fig. 3.28).

Fig. 3.28 – Inicio del programa SISTAR Caja

Una vez iniciada sesión aparece la ventana principal (fig. 3.29). A continuación describiremos cada sección de la interfaz del programa.



3.29 - Ventana principal de SISTAR Caja

Debajo del logotipo del programa se puede seleccionar el tipo de transacción. Si es una venta cuando esta se termine el sistema la registrará y los cambios afectarán al inventario. SI es una cotización el sistema no almacenará nada ni hará cambios en la base de datos (ver fig. 3.30).

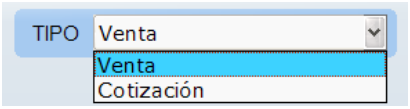


Fig. 3.30 – Selección del tipo de transacción

En la parte superior derecha el usuario debe elegir el cliente (fig. 3.31). Esto puede hacerlo por su número de cliente o por su nombre. Si el cliente es ocasional y no se desea registrarlo se debe especificar la venta al cliente *mostrador* que es el cliente por defecto. De lo contrario se puede ingresar un nuevo cliente al sistema utilizando el botón de *añadir cliente.*

Fig. 3.31 - Selección de cliente

Debajo del recuadro cuyo título es "detalles de la venta" sigue el panel de selección de productos. La primera opción del panel es seleccionar el artículo a vender acorde a su clave. Se ofrece una lista para escoger o escribir la clave del producto a vender. A continuación viene la etiqueta de producto y junto a esta una lista con los productos. Cuando se elije un producto por su clave o por su nombre estas dos listas se sincronizarán automáticamente (ver fig. 3.32).

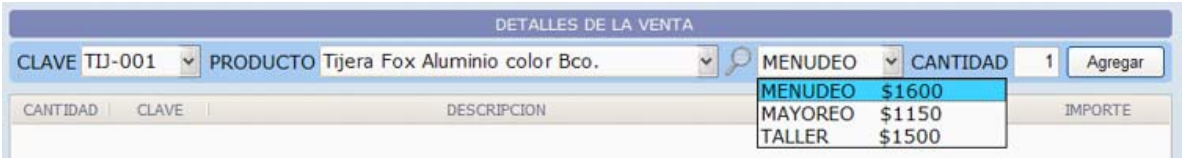


Fig. 3.32 – Panel de venta

Si es difícil determinar el producto se ofrece una ayuda cuando se hace clic sobre el botón cuya figura es una lupa. Haciendo esto se mostrará una ventana que permite buscar con mayor facilidad el producto en cuestión. Esta ventana la trataremos posteriormente.

Una vez que el usuario especifica el producto debe elegir el tipo de precio. La tercera lista que se muestra permite elegir el tipo de precio para la venta (fig 3.33a). Para esto, cuando se despliega la lista también se muestra el costo para cada tipo de precio. A la derecha de la etiqueta *cantidad* el usuario puede introducir la cantidad de piezas a vender.

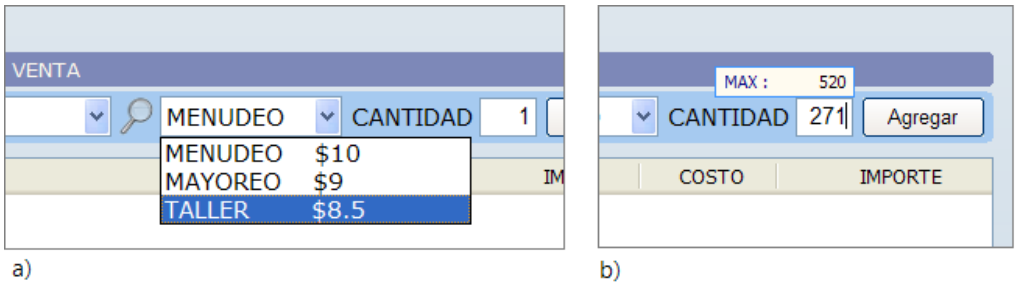


Fig. 3.33 – Selección del tipo de precio y especificación de cantidad

Cuando el usuario activa el control de edición de cantidad se muestra una leyenda informando acerca de las piezas disponibles para el artículo seleccionado. El usuario no podrá rebasar la cantidad de piezas en la venta a las que hay disponibles. Además, el programa solo lista los productos cuya existencia es mayor a una pieza (fig. 3.33b).

Para encontrar un artículo de una manera más cómoda el usuario puede hacer clic sobre el botón cuyo ícono es una lupa. A continuación el sistema mostrará una ventana emergente para seleccionar el producto a agregar.

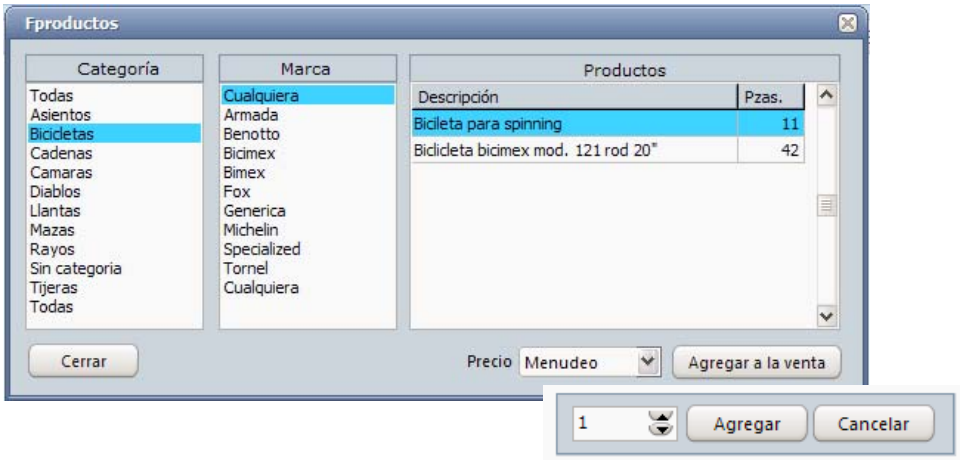
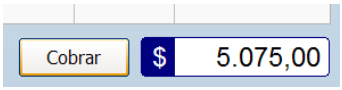


Fig. 3.34 – Búsqueda rápida de productos

En la fig. 3.34 podemos apreciar la ventana desde la cual el usuario puede buscar un producto por la categoría y por la marca especificándolas de las listas de la izquierda. Esto hará que el sistema muestre en la rejilla de la derecha los productos que coincidan con los criterios anteriores.

En la parte inferior, una vez que un producto es seleccionado, el usuario puede agregarlo a la venta especificando el precio. Cuando hace clic sobre el botón *Agregar a la venta* se muestra el un panel donde el usuario especifica la cantidad de piezas ó bien cancela la acción.

Una vez agregado el artículo a la venta, se va calculando el total de la venta, que se muestra en la parte inferior de la ventana (fig 3.35a). Si se desea, a partir de un artículo se puede dar por concluida la venta y cobrarla (fig. 3.35b).



a)

b)

Fig. 3.35 – Ventana de cobro

Capitulo 4

4 Desarrollo del Sistema Y pruebas

CAPITULO 4.
Desarrollo del Sistema y Pruebas.

4.1 Ambiente de las herramientas de implementación

Las herramientas utilizadas para el desarrollo de este proyecto fueron las siguientes:

4.1.1 MySQL y MySQL Administrator

El motor de Bases de Datos MySQL se instala únicamente en la máquina que fungirá como servidor. Aunque la potencia y estabilidad de MySQL es competente a los mejores SGBD como Oracle y SQL Server, el sistema en modo consola es un poco tedioso de utilizar, sobre todo en la creación de las tablas. Para evitar trabajar sobre el modo consola de MySQL utilizaremos *MySQL Administrator*[1], que es un frontend que se distribuye de manera gratuita, aunque por separado del gestor.

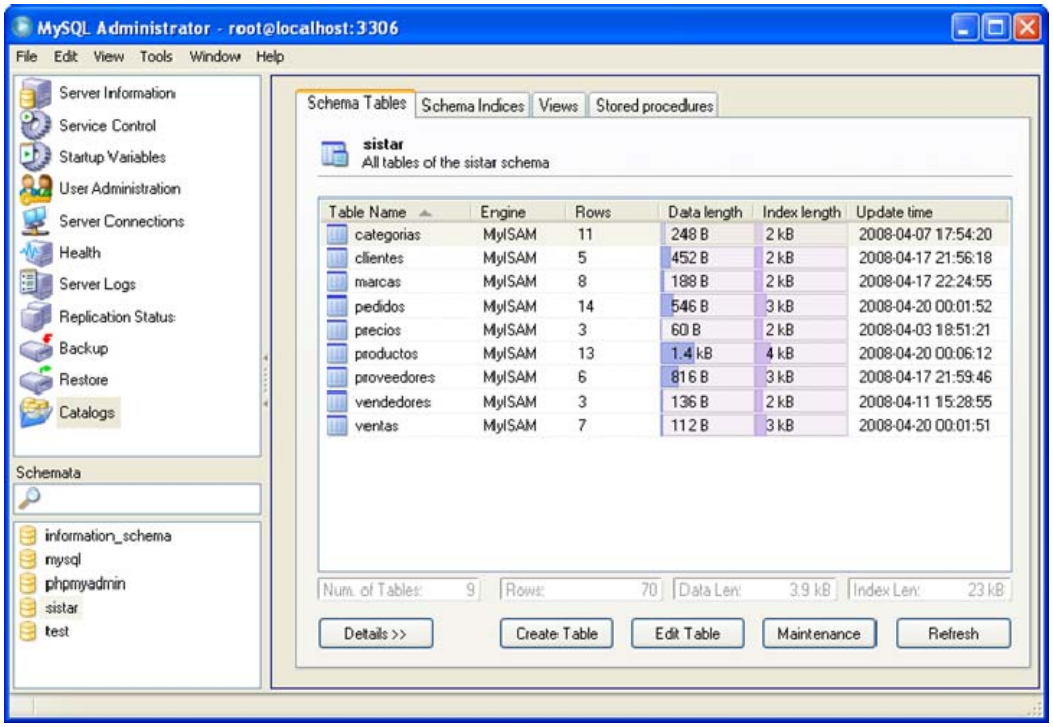


Fig. 4.1 – El entorno de MySQL Administrator

[1] MySQL Administrator está disponible en http://www.mysql.com/

Al iniciar MySQL Administrator nos pide ciertos parámetros de conexión que se refieren a la dirección del servidor, un nombre de usuario y contraseña. Si los datos son correctos el sistema inicia por completo.

MySQL organiza cada base de datos dentro de un catálogo, por ello para acceder a la base de datos (si ya ha sido previamente creada) debemos hacerlo por medio del catálogo, seguido por el nombre de la base de datos (fig 4.1).

4.1.2 Borland Delphi 7

El entorno de Delhi simplifica el proceso de desarrollar interfaces gracias a método drag&drop que permite diseñar los formularios de una aplicación con solo arrastrar y colocar los controles desde su paleta de componentes hasta nuestra aplicación.

La asignación de eventos en los controles permite una funcionalidad completa al agregarle procedimientos a ejecutar dependiendo de la interacción que tenga la aplicación con el usuario (fig 4.2).

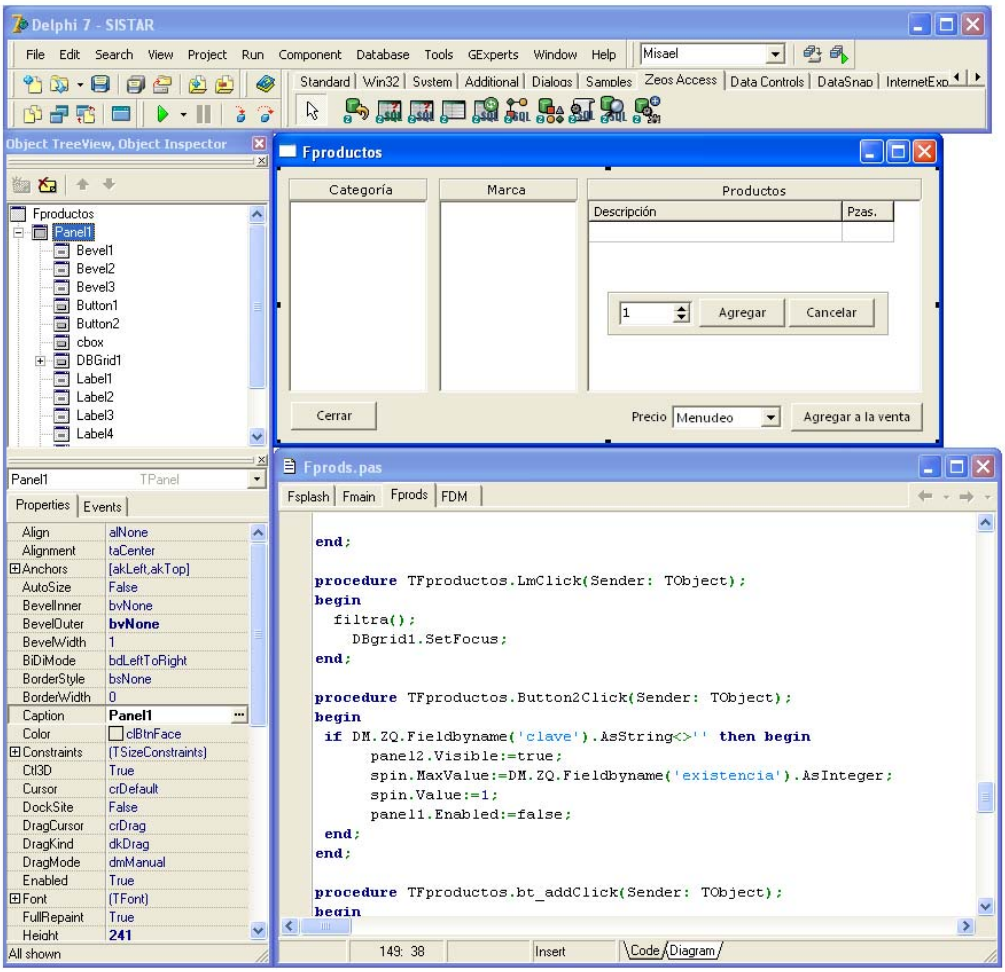


Fig. 4.2 – El entorno de programación de Delphi y la pestaña de Zeos Access

4.2 Conectividad a la Base de Datos.

La conexión al servidor de MySQL no puede hacerse en Deplphi de forma nativa. Para establecer una conexión desde nuestra aplicación a MySQL es necesario hacer uso de un controlador ODBC (Open Dabatabase Conectivity). Esta no es la única forma, una mejor opción es utilizar una librería de enlace dinámico (que se descarga desde la página de MySQL.com). En el capítulo 1 se habló sobre las grandes cualidades del IDE Delphi y entre estas, está el uso de Componentes, que permiten agregar al entorno librerías, funciones y controles desarrollados por terceros. Entre estas librerías se encuentra ZeosLib[2] que permite acceder a servidor de MySQL sin necesidad de utilizar ODBC. Esto permite que la conexión sea más rápida sin mencionar que facilita enormemente el programar la conexión desde Delphi.

Una vez que se instalan las librerías de ZeosLib, también se agregan de manera automática 5 controles en una nueva pestaña llamada Zeos Acces a la paleta de componentes. (fig. 4.2) Estos 5 controles en conjunto con una librería de enlace dinámico (que se distribuye con los mismos) nos permiten acceder al Servidor MySQL y ejecutar instrucciones sobre este, desde simples consultas hasta Scripts. Para establecer una conexión con la base de datos que utilizará nuestra aplicación se deben agregar los siguientes controles al programa y configurar las propiedades de estos de la siguiente manera.

ZConection. Este componente engloba la conexión a la base de datos. Los demás controles lo utilizan para efectuar sus operaciones sobre la base de datos. Las principales propiedades a configurar son el catálogo, el nombre de la base de datos, nombre de usuario y contraseña.

ZQuery. A través de su propiedad SQL asignamos la consulta a realizar. Esta consulta puede ser de cualquier tipo. Imaginemos el uso de ese componente como si estuviésemos introduciendo el comando directamente sobre la consola de MySQL. Cuando se utilizan instrucciones del lenguaje que no devuelven resultados (como UPDATE, INSERT, ALTER; etc.) utilizamos el procedimiento ExecSQL para ejecutar la sentencia especificada en la propiedad SQL, si la consulta devuelve un conjunto de resultados (con la instrucción SELECT) entonces utilizamos el procedimiento Open.

DataSource. Este componente es propio de Delphi y sirve como un enlace entre un conjunto de datos (por ejemplo, el resultado de un SELECT que ejecuta un Zquery) y otros componentes para poder visualizar y editar los datos. Para utilizar este componente únicamente debemos colocar en su propiedad *DataSet* el nombre del Zquery del cual queremos visualizar los datos.

[2] Disponible en www.sourceforge.net

DBGrid. Este es uno de los muchos componentes que podemos utilizar para ver y editar los datos de una base de datos, ya sea sobre una tabla únicamente o bien por un conjunto de datos. Este es un componente visual que se coloca sobre un form. Para utilizarlo solo necesitamos especificar en su propiedad *DataSource* el nombre del componente de tipo DataSource que va a enlazar los datos de un DataSet con la rejilla.

Utilizando estos componentes podemos implementar, aunque de manera muy primitiva, una aplicación que acceda a una base de datos, muestre y edite sus datos. Todo esto sin programar una línea de código en absoluto.

4.3 Código de Consultas

En ambas aplicaciones del proyecto (Caja y Administrador) se hacer gran cantidad de consultas, que, además de ser dinámicas, son complejas y el algoritmo que genera las mismas es aún más complejo.

En este apartado se especifican algunas consultas que se consideran más relevantes. Las consultas se encuentran almacenadas dentro de la propiedad SQL de un Zquery. El código está combinado entre OOP y SQL.

4.3.1 Consultas que realiza SISTAR Caja

En primer lugar, se realiza la conexión a la base de datos. Posteriormente se verifica la validez del usuario y contraseña. Siendo satisfechas las condiciones anteriores se muestra la ventana principal del sistema.

```
{La siguiente consulta es generada por el programa leyendo los datos que inglesa
el usuario para identificarse en el sistema}
cad:='select * from vendedores where usuario='+#39+ed_usuario.Text+#39;

{Abrimos la conexión}
DM.ZCsistar.Connect;

{ZQtemp es un Zquery. Este modelo de instrucciones se utiliza para ejecutar una
consulta, por lo que se omitirán en el futuro con el fin de no extender la
documentación.}
DM.ZQtemp.Close;
DM.ZQtemp.SQL.Clear;
DM.ZQtemp.SQL.Add(cad);
DM.ZQtemp.Open;

{Una vez realizada la consulta, verificamos los datos del usuario}
if not DM.ZQtemp.Eof then begin {existe el usuario?}
   {FieldByName es la manera de acceder a un campo, la conversión sigue después
    del paréntesis, AsString regresa el dato como cadena.}
   cad:= DM.ZQtemp.fieldbyname('pwd').AsString;
   if CompareText(ed_pwd.Text,cad) = 0 then begin
   nombre:=DM.ZQtemp.fieldbyname('nombre').AsString;
   idv:=DM.ZQtemp.fieldbyname('id_vendedor').AsString;
```

```
   form2.setvendedor(nombre,idv);
       Fmain.Form2.nuevaventa;
    end else  MessageDlg('Nombre de usuario o contraseña incorrecta', mtError,
[mbOK], 0);
  end else MessageDlg('Nombre de usuario o contraseña incorrecta', mtError,
[mbOK], 0);
{Es recomendable cerrar siempre las consultas}
DM.ZQtemp.Close;
```

El punto de venta necesita inicialmente la lista de los clientes a los que es posible realizar una venta, y el listado de los productos disponibles. Esta consulta se realiza constantemente debido a que puede haber cambios realizados por el programa de administración o desde otras terminales de Punto de Venta.

```
{Obtiene el listado de clientes con su respectiva clave}
ZQtemp.SQL.Add('SELECT id_cliente, nombre from clientes ORDER BY nombre');

{Cuando el usuario selecciona un cliente se consulta la información sobre este}
ZQtemp.SQL.Add('SELECT direccion, telefono, rfc from clientes where
id_cliente='+clave);

{Esta consulta obtiene el listado de artículos disponibles}
ZQtemp.SQL.Add('SELECT clave, descripcion from productos where
(almacen+exhibicion)>0');
```

Es importante aclarar que las consultas anteriores pertenecen a diferentes rutinas, las cuales toman los datos devueltos por cada consulta y los asignan a diferentes controles.

Una vez que se ha establecido el Vendedor y el Cliente, y que se obtiene el listado de artículos disponibles, el usuario puede proceder a agregar productos para realizar una venta. Cada vez que el usuario selecciona un producto, se obtiene la siguiente información necesaria para realizar la venta: Cantidad de piezas disponibles, el precio de menudeo, mayoreo y taller.

```
{Al seleccionar un producto, se consulta la siguiente información}
ZQtemp.SQL.Add('select precio_men, precio_may, precio_esp, (exhibicion+almacen)
as cant from productos where clave='+#39+key+#39);
```

Una vez que el usuario termina la venta, el sistema tiene que generar el conjunto de consultas para registrar la venta. Primero debe bloquear las tablas, luego obtener el número de venta, valor fundamental, pues el registro de la venta y los productos depende de este valor. Posteriormente debe insertar los datos de cada pedido en la tabla correspondiente, actualizar las existencias en la tabla de productos y finalmente liberar las tablas bloqueadas.

Mediante un complejo algoritmo, cuando se registra una venta, el sistema genera una consulta como la que sigue:

```
lock tables pedidos write, ventas write, productos write;
INSERT INTO ventas  VALUES('9','2008-04-21','2','3');
INSERT INTO pedidos VALUES('9','ASI-001','2','2','200','380');
INSERT INTO pedidos VALUES('9','TIJ-002','1','1','100','200');
INSERT INTO pedidos VALUES('9','CAD-001','1','1','50','75');
INSERT INTO pedidos VALUES('9','RAY-002','40','3','2','8.5');
UPDATE productos set almacen=1, exhibicion=5 where clave='ASI-001';
UPDATE productos set almacen=0, exhibicion=2 where clave='TIJ-002';
UPDATE productos set almacen=14, exhibicion=0 where clave='CAD-001';
UPDATE productos set almacen=456, exhibicion=20 where clave='RAY-002';
UNLOCK TABLES;
```

4.3.2 Consultas que realiza SISTAR Administrador.

Al igual que SISTAR Caja, inicialmente se establece la conexión. La diferencia reside en que la información de los usuarios del programa está almacenada en una tabla dentro de la base de datos. Para esta aplicación se utilizan los nombres de usuario y contraseña de la base de datos directamente. Esto es, los usuarios que se crean desde el propio gestor MySQL.

Es por ello que inicialmente no hay una consulta sino un intento de conexión, si la conexión falla, se captura la excepción y el programa muestra nuevamente la ventana para recibir los parámetros de la conexión. De lo contrario el sistema inicia.

Cuando el sistema inicia satisfactoriamente, hace una consulta para obtener el inventario. Es importante notar que no selecciona todos los datos de los productos, solamente los datos usados en primera instancia. Esto con el fin de dar una apariencia más organizada al usuario y reducir la cantidad de datos que el servidor debe suministrar por la red.

```
select productos.Clave, productos.Descripcion, productos.almacen,
productos.exhibicion, productos.precio_men, productos.precio_may,
productos.precio_esp, categorias.nombre
from productos join categorias
on (productos.categoria=categorias.clave);
```

Con estos datos, el sistema muestra en la rejilla de la pestaña Inventario la información básica de cada producto (fig 3.2). Cuando el usuario decide editar la información de un producto el sistema consulta toda la información acerca de este y la muestra en la ventana de edición (fig. 3.1).

```
Select * from productos where Clave=:pkey
```

Una manera de ir actualizando la consulta es a través de parámetros. En la consulta anterior se especifica el parámetro ‘:Pkey’. Cada parámetro en la propiedad SQL de un Zquery va precedido de dos puntos. De esta manera antes de realizar la consulta se especifica el valor del (los) parámetro(s). Un ejemplo para el caso anterior sería algo así:

```
ZQedit.ParamByName(‘pkey’).AsString;
ZQedit.ParamByName(‘pkey’).Value=’BIC-001’;
```

Aquí entra otra ventaja que tiene Delphi en el manejo de base de datos y es el componente Tnavigator, el cual, cuando es asociado a un DataSet, en este caso ZQedit, permite actualizar, eliminar y moverse entre diversos registros utilizando métodos propios del navegador sin necesidad de hacer otra consulta, lo que implicaría muchas instrucciones más en el código. De esta manera, para actualizar los datos solo es necesario ejecutar la instrucción siguiente:

```
navegador.BtnClick(nbPost);
```

Para el manejo de Proveedores y Clientes, el método es prácticamente el mismo, por lo que no se considera necesario tratarlos explícitamente.

Un proceso al que vale la pena mencionar es el que genera una consulta con un filtro avanzado. Este filtro avanzando permite seleccionar información muy específica sin que el usuario tenga que realizar la consulta, ya que el sistema va generando la consulta de acuerdo a los parámetros que va ingresando el usuario (fig 3.3). Para esto, el sistema primero hace una consulta donde obtiene el listado de todos los proveedores, marcas y categorías. Cuando el usuario da click sobre el botón filtrar, este ejecuta el siguiente procedimiento.

```
{Determina si un producto debe cumplir con todas las condiciones o solo alguna
de estas}
if radio1.Checked=true then union:='AND ' else union:='OR ';

consulta.Add('select productos.Clave, productos.Descripcion, productos.almacen,
productos.exhibicion, productos.precio_men, productos.precio_may,
productos.precio_esp, categorias.nombre from productos join categorías on
productos.categoria=categorias.clave) WHERE');

anterior:=false;

If check_categoria.Checked then begin {filtrar por categoría}

cond:='productos.categoria='+lista_nocat.Items.Strings[lista_nocat.itemindex];
      consulta.Add(cond); anterior:= true; end;

if check_marca.Checked then begin {filtrar por marca}
      if anterior then cond:=union else cond:='';
      cond:= cond+'productos.id_proveedor='+
             lista_noprov.Items.Strings[lista_noprov.itemindex];
      consulta.Add(cond); anterior:= true; end;

 if check_proveedor.Checked then begin {filtrar por proveedor}
      if anterior then cond:=union else cond:='';
      cond:=cond+'productos.Marca='+
            lista_nomarca.Items.Strings[lista_nomarca.itemindex];
      consulta.Add(cond); anterior:= true; end;
```

```
if check_existencia.Checked then begin {filtrar por existencia}
      if anterior then cond:=union else cond:='';
      cond:=cond+'(productos.almacen+productos.exhibicion)';
      case cbox1.ItemIndex of
        0 : cond:=cond+'='+mask.Text;
        1 : cond:=cond+'<'+mask.Text;
        2 : cond:=cond+'>'+mask.Text;
      end;
      consulta.Add(cond); end;

  ZQF.Close; ZQF.SQL:=consulta; ZQF.Open;
```

Con este procedimiento es posible aplicar un filtro avanzado que puede combinarse con el filtrado simple y el ordenamiento (opciones disponibles en la barra de herramientas de la pestaña inventario) para obtener información muy precisa.

En la pestaña ventas se muestran dos rejillas, la rejilla superior muestra todas las ventas (de acuerdo a criterio seleccionado) con su respectivo importe. Además se muestra el corte de caja en la barra que está entre esta rejilla y la rejilla inferior. En la rejilla inferior se muestra los detalles de la venta que esté seleccionada en la rejilla superior (fig. 3.11). Inicialmente el programa ejecuta la siguiente consulta que muestra las ventas del día.

```
fecha:=#39+FormatDateTime('yyyy-mm-dd',Now)+#39;
DM.ZQventas.Close;
DM.ZQventas.SQL.add('

select
  id_venta, fecha,cliente, clientes.nombre, id_vendedor,
  vendedores.nombre,
  (
    select sum(cantidad*p_compra) from pedidos
    where pedidos.id_venta = ventas.id_venta
  ) as inversion,
  (
    select sum(cantidad*p_venta) from pedidos
    where pedidos.id_venta = ventas.id_venta
  ) as importe
from ventas join clientes on cliente=id_cliente join vendedores on
vendedor=id_vendedor where fecha='+fecha+ 'order by id_venta');
```

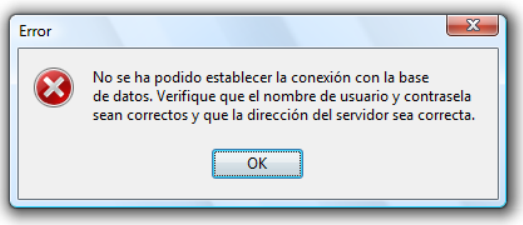
El corte de caja es calculado por el programa, sumando el resultado de todos los registros de los campos inversión e importe. La clausula “where” se modifica para mostrar las ventas según el criterio: por cliente, por periodo, por semana ó por día.

La última consulta que trataremos se efectúa cuando el usuario selecciona una venta. Cuando esto ocurre el sistema genera otra consulta, acorde a la venta seleccionada para ver los detalles de la misma.

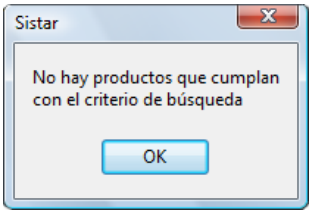
```
select cantidad, id_producto, descripcion, nombre, p_compra, p_venta,
(cantidad*p_venta) as importe, (cantidad*p_compra) as ganancia
from pedidos join productos on id_producto=clave join precios on
tipo_precio=clavep where id_venta=:no_venta;
```

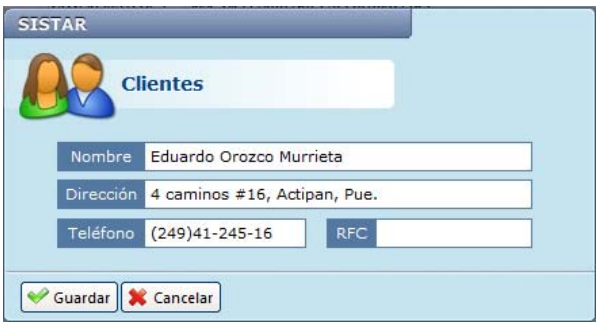
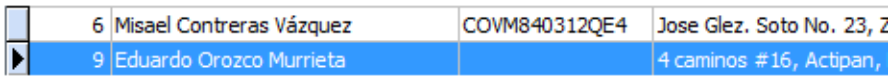
4.3.3 Pruebas

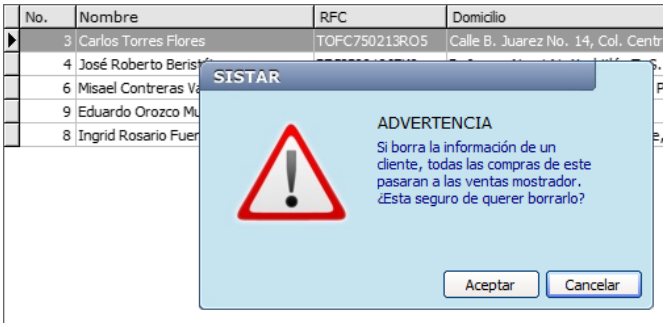
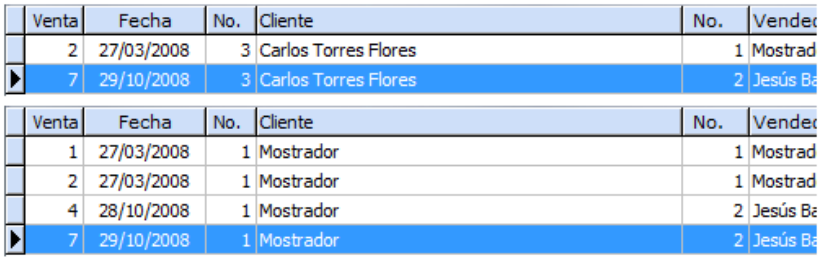
Se realizaron numerosas y extensas pruebas para cada función del sistema, pero dada la extensión del mismo solo mencionaremos las pruebas hechas a los casos de uso requeridos por el usuario.

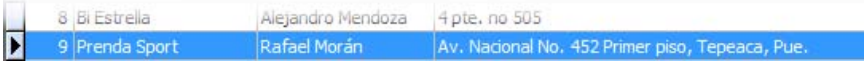
Caso de Experimento:	Acceso al sistema
Objetivo:	Verificar que el sistema restringe el acceso solo a usuarios registrados
Inf. De entrada:	Nombre de usuario (no registrado) y contraseña
Resultado esperado:	El sistema notificará que la información proporcionada es incorrecta
Resultado obtenido:	
Resultado de la prueba:	Satisfactoria

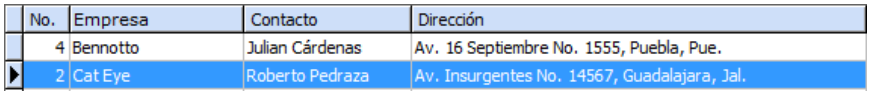
Caso de Experimento:	Agregar producto al inventario
Objetivo:	Verificar que el sistema almacena la información de nuevos productos
Inf. De entrada:	
Resultado esperado:	El sistema almacenará el nuevo producto y se listará en el inventario
Resultado obtenido:	
Resultado de la prueba:	Satisfactoria

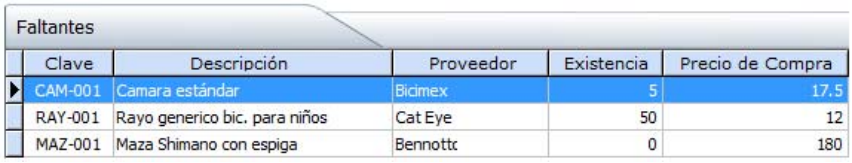
Caso de Experimento:	Generar filtros dinámicamente
Objetivo:	Verificar que el sistema permite al usuario generar filtros de información específica sobre el inventario
Inf. De entrada:	
Resultado esperado	De encontrar artículos coincidentes se listarán en la ventana de inventario, en otro caso notificará que no hubo coincidencias
Resultado obtenido	
Resultado de la prueba:	Satisfactoria

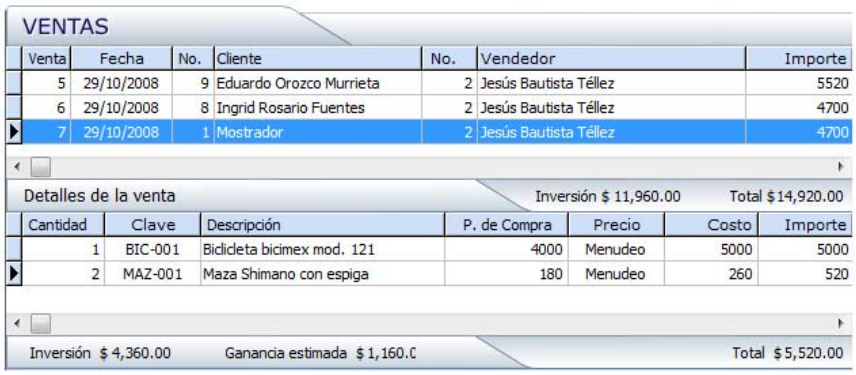
Caso de Experimento:	Agregar cliente
Objetivo:	Registrar un cliente nuevo
Inf. De entrada:	
Resultado esperado:	El sistema almacenará la información y mostrará el nuevo cliente en la lista de clientes
Resultado obtenido:	
Resultado de la prueba:	Satisfactoria

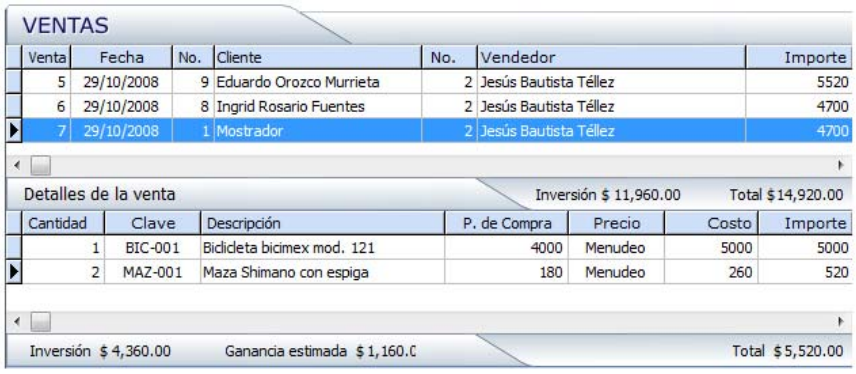
Caso de Experimento:	Eliminar Cliente del sistema
Objetivo:	Eliminar la información de un cliente y verificar la integridad referencial
Inf. De entrada:	
Resultado esperado:	Se eliminará la información del cliente y las ventas hechas a este cliente se registrarán como ventas de mostrador
Resultado obtenido:	
Resultado de la prueba:	Satisfactoria

Caso de Experimento:	Agregar proveedor
Objetivo:	Agregar la información de un nuevo proveedor
Inf. De entrada:	
Resultado esperado:	Se guardará la información y se mostrará en la lista de proveedores
Resultado obtenido:	
Resultado de la prueba:	Satisfactoria

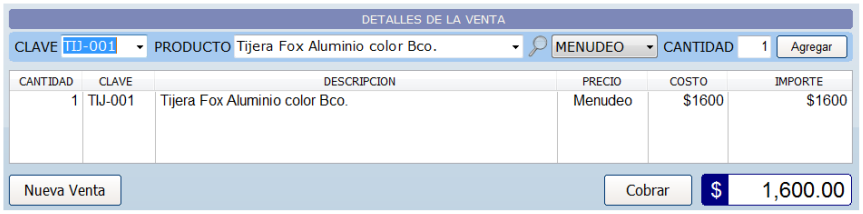
Caso de Experimento:	Eliminar un proveedor del sistema
Objetivo:	Eliminar la inf. De un proveedor y verificar la integridad referencial sobre el inventario
Inf. De entrada:	El proveedor a eliminar  No. | Empresa | Contacto | Dirección 4 | Bennotto | Julian Cárdenas | Av. 16 Septiembre No. 1555, Puebla, Pue. 2 | Cat Eye | Roberto Pedraza | Av. Insurgentes No. 14567, Guadalajara, Jal.
Resultado esperado:	Se eliminará la inf. Del proveedor y los artículos vinculados a este se les asignará proveedor sin especificar
Resultado obtenido:	 Desc. Camara estándar Marca Generica Notas con pivote de tornillo completo Proveedor Cat Eye Categoría Camaras Precio Compra 17.5 Desc. Camara estándar Marca Generica Notas con pivote de tornillo completo Proveedor Sin Especificar Categoría Camaras Precio Compra 17.5
Resultado de la prueba:	Satisfactoria

Caso de Experimento:	Consultar faltantes
Objetivo:	Verificar que el sistema enlista todos los artículos faltantes
Inf. de entrada:	Ninguna
Resultado esperado:	Se mostrarán las ventas del día y el corte de caja
Resultado obtenido:	 Faltantes Clave | Descripción | Proveedor | Existencia | Precio de Compra CAM-001 | Camara estándar | Bicimex | 5 | 17.5 RAY-001 | Rayo generico bic. para niños | Cat Eye | 50 | 12 MAZ-001 | Maza Shimano con espiga | Bennottc | 0 | 180
Resultado de la prueba:	Satisfactoria

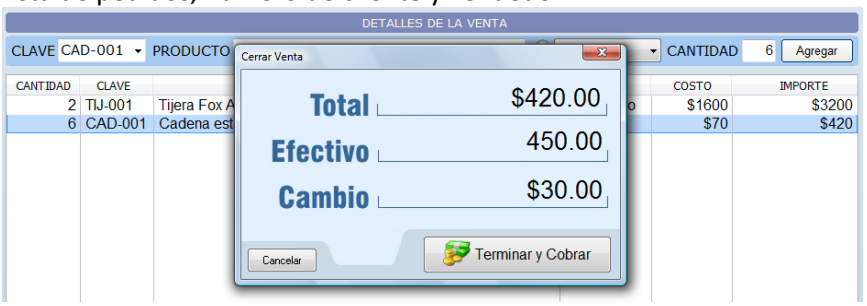
Caso de Experimento:	Consultar reporte del día
Objetivo:	Verificar que el sistema muestra las ventas hechas durante el día en curso, desglose a detalle cada venta, el importe, total, y corte de caja del día.
Inf. De entrada:	Ninguna
Resultado esperado:	Se mostrarán las ventas del día y el corte de caja
Resultado obtenido:	
Resultado de la prueba:	Satisfactoria

Caso de Experimento:	Consultar reporte de la semana
Objetivo:	Verificar que el sistema muestra las ventas hechas durante la semana en curso, desglose a detalle cada venta, el importe, total, y corte de caja del día.
Inf. De entrada:	Ninguna
Resultado esperado:	Se mostrarán las ventas del día y el corte de caja
Resultado obtenido:	
Resultado de la prueba:	Satisfactoria

Caso de Experimento:	Agregar Vendedores
Objetivo:	Verificar la creación de usuarios de SISTAR Caja, es decir, vendedores
Inf. de entrada:	Datos del vendedor 
Resultado esperado:	Se almacenará la información del nuevo vendedor y se enlistará en la sección de vendedores
Resultado obtenido:	
Resultado de la prueba:	Satisfactoria

Caso de Experimento:	Insertar pedido
Objetivo:	Añadir un pedido a una venta
Inf. de entrada:	Clave, cantidad y tipo de precio del artículo
Resultado esperado:	Se anexará a la lista de pedidos el nuevo artículo, se modificará el importe de la venta y, si la venta estaba en ceros, habilitará el botón de cobro
Resultado obtenido:	
Resultado de la prueba:	Satisfactoria

Caso de Experimento:	Registrar venta
Objetivo:	Cerrar una venta y reflejar los cambios en el inventario y corte de caja

Inf. de entrada:	Lista de pedidos, numero de cliente y vendedor 
Resultado esperado:	En SISTAR Caja se iniciará una nueva venta, en SISTAR Administrador se actualizaran el inventario y corte de caja
Resultado obtenido:	 
Resultado de la prueba:	Satisfactoria

Conclusiones

CONCLUSIONES

- El desarrollo de todo este trabajo deriva el diseño y análisis bajo la metodología RUP y del uso de UML para expresarlos sistemáticamente. En primera instancia la metodología RUP es frustrante porque cada iteración inicia desde la fase de análisis y dichas iteraciones nunca finalizan realmente. Sin embargo, a medida que se trabaja en esta metodología podemos notar que cada ciclo nos acerca al producto final y que podemos hacer tantos ciclos como creamos necesarios para definir a detalle todos los aspectos de sistema.
- Se desarrollo e implementó el sistema SISTAR utilizando el modelo cliente servidor, dicho sistema permite realizar de manera ágil el proceso de compra y venta de refacciones, además de otras funcionalidades que ayudarán a administrar todo el ámbito del negocio.
- También se diseñó una base de datos, la cual fue implementada en MySQL, donde se almacena la información para la interface desarrollada en Delphi.
- Utilizando la potencia del gestor MySQL se elaboraron las consultas para obtener reportes de caja, artículos faltantes, administrar la información de clientes, proveedores, introducir ventas y administrar el inventario.
- Se consideró necesario el poder extraer toda esta información para utilizarla en otras aplicaciones por lo que se implementó la funcionalidad de exportar la información a archivos de Microsoft™ Excel®.
- El acceso al front-end del sistema depende directamente del gestor MySQL, por lo que la medida de la seguridad de este será la que proporcione MySQL, el cual a pesar de ser un motor gratuito ha probado se confiable y estable.

Trabajo a Futuro.

Hay algunas funcionalidades que de implementarse podrían incrementar las prestaciones del sistema. Las más relevantes a considerar son:

1. Incorporar un Lector de código de barras. Cuando se hizo el análisis se evaluó que no era necesario implementarlo ya que la mayoría de piezas carecen de código de barras o son demasiado pequeñas para tener uno. Sin embargo, si el cliente en determinado momento estima necesario el incorporar esta funcionalidad al sistema se requiere implementar un procedimiento en el módulo SISTAR Caja y modificar ligeramente la base de datos para guardar esta información.
2. Implementar un modulo para llevar la contabilidad que, utilizando la base de datos, pueda extraer información sobre las actividades financieras e incorporar esta información en pólizas y cuentas.
3. Este sistema no se vende, solamente se licencia. Por lo que el sistema será ofertado en la página web del autor: www.jirehonline.net.

Referencias Bibliográficas

[1] Ambrosio, A. D. (2007). “Análisis y Diseño de Bases de datos”. Puebla: Diplomado en Bases de Datos, Facultad de Ciencias de la computación, BUAP.

[2] Castaño M., Piattini V. (2001) “Fundamentos y Modelos de Bases de Datos”. Madrid: Alfaomega Grupo Editor.

[3] Cota, A. (1994) "Ingeniería de Software". Soluciones Avanzadas. Julio de 1994. pp. 5-13

[4] Delphi. (2008, 5) de marzo. Wikipedia, La enciclopedia libre. Fecha de consulta: 03:23, abril 16, 2008 from http://es.wikipedia.org/w/index.php?title=Delphi&oldid=15608080.

[5] Information System. (n.d.). Computer Desktop Encyclopedia. Retrieved April 12, 2008, from Answers.com Web site: http://www.answers.com/topic/information-system-information-systems

[6] Jacobson, I. (1998). "Applying UML in The Unified Process" Presentación. Rational Software. Presentación disponible en http://www.rational.com/uml como UMLconf.zip

[7] León, S. G. (1996). “Ingeniería de Sistemas de Software”. Madrid: Editorial Isdefe

[8] Lewis, G. (1994). "What is Software Engineering?" DataPro (4015). Feb 1994. pp. 1-10.

[9] MySQL AB Corp. (2006). “MySQL reference manual” (Rev. 327). http://dev.mysql.com

[10] Pérez De Celis, M. C. (2007). “Ingeniería de Software”. Puebla: Diplomado en Bases de Datos, Facultad de Ciencias de la computación, BUAP.

[11] Raumbaugh J., Jacobson I., Booch G. (2000) “El lenguaje unificado de Modelado”. Madrid: Pearson Education.

[12] Rockart et. Al (1996) “Eight imperatives for the new IT organization Sloan Management review”. Massachusetts: MIT

[13] Silberschatz A., Korth H., Sudarshan S. “Fundamentos de Bases de Datos”, Cuarta Edición, Mc Graw Hill.

[14] Sistema de información. (2008, 9) de abril. Wikipedia, La enciclopedia libre. Fecha de consulta: 23:57, abril 12, 2008 from http://es.wikipedia.org/

[15] Schmuller, J. (2002) “Aprendiendo UML en 24 Horas”. Prentice Hall.

[16] Ullman (2002) “A first course in database systems” Adison-Wesley, Pearson Education.

Anexos

Anexo: Código Fuente del Sistema

Un programa en Delphi consiste en al menos un form. El programa SISTAR Caja y SISTAR Administrador consisten de varios forms. Cada form consiste en cuatro archivos, un *.dfm, *.pas *.dcu y *.dpp. De estos, el más importante es el .dfm y .pas. El archivo dfm es un archivo binario que almacena la información sobre los distintos controles que usa el form y los diferentes recursos como imágenes o sonidos. El archivo .pas contiene todo el código fuente.

A continuación incluimos el código fuente de algunas funciones del sistema. Delphi tiene la pequeña inconveniencia de que la codificación en OOP tiende a inflar demasiado el texto del código fuente, es por ello que no incluiremos todo el contenido de los archivos *.pas.

1. SISTAR Administrador.

Archivo: Fmain.pas.
Descripción: Archivo del código fuente de la pantalla principal del programa.

Nota: DM es un objeto de tipo módulo de datos, el cual almacena los componentes de ZeosLib *(véase: 4.2 Conectividad con la base de datos, pag. 82)*

Función: Editar productos.
Descripción: Enlista los artículos en la pestaña de Inventario y muestra la ventana de edición de producto.

```
procedure TFormMain.bt_editarClick(Sender: TObject);
var
 clave : string;
begin
  DM.ZQedit.Close;
  DM.ZQedit.SQL.Clear;
  Try
    clave:=DM.ZQproductos.FieldValues['Clave'];
    if clave='' then DM.ZQedit.SQL.Add('Select * from productos')
    else DM.ZQedit.SQL.Add('Select * from productos where Clave='+#39+clave+#39);
    DM.ZQedit.Open;
    DM.ZQmarca.Open;
    DM.ZQcategoria.Open;
    DM.ZQproveedor.Open;
  except
      MessageDlg('No se puede realizar la consulta esto puede deberse a que se ha
      desconectado el servidor, o la información está en uso por otro usuario.',
      mtError, [mbOK], 0);
    exit;
  end;
  oscurece();
  FormProductos.colorea(clGray);
  FormProductos.pnl_guardar.Visible:=false;
  formproductos.Showmodal;
  activa();
end;
```

Función: Eliminar producto.
Descripción: Esta función es llamada en el evento onClick() del botón eliminar producto en la pestaña de inventario. Muestra una ventana para que el usuario confirme la acción de borrado y de ser positiva continúa con el procedimiento de borrar el producto del inventario.

```
procedure TFormMain.btn_EliminarClick(Sender: TObject);
var
 clave : string;
begin

    DM.ZQup.Close;
    DM.ZQup.SQL.Clear;
 try
    clave:=DM.ZQproductos.FieldValues['Clave'];
    DM.ZQup.SQL.Add('Select * from pedidos where id_producto='+#39+clave+#39);
    DM.ZQup.Open;
 except
    exit;
 end;
    oscurece();
  {1}if not DM.ZQup.Eof then begin
    FormMensage.titulo.Caption:='ERROR';
    FormMensage.info.Caption:='No se puede eliminar el producto'+#13+'porque hay
ventas que'+#13+'lo contienen';
    FormMensage.ShowModal;
    activa();
    exit;
  {.1}end else begin
    FormMensage.titulo.Caption:='ATENCION';
    FormMensage.info.Caption:='¿Confirma usted que desea borrar'+#13+'El producto
seleccionado?';
    if  FormMensage.ShowModal = mrOk then begin
          //Procedemos a borrar el artículo.
         DM.ZQproductos.Close;
         DM.ZQup.Close;
         DM.ZQup.SQL.Clear;
         DM.ZQup.SQL.Add('DELETE FROM productos WHERE Clave='+#39+clave+#39);
         DM.ZQup.ExecSQL;
         DM.ZQproductos.Open;
     end;
  {:1}end;
  DM.ZQup.Close;
  activa();
end;
```

Función: Mostrar Venta
Descripción: Recibe el numero (id) de la venta a mostrar, hace la consulta y muestra la venta seleccionada a detalle, es decir, muestra la cantidad, artículos, precio e importe que corresponden a dicha venta así como el total, la inversión y ganancia estimada.

```
procedure TformMain.muestraventa(numero : integer);
var
 venta, ganancia : currency;
begin
  Dm.ZQpedidos.Close;
  Dm.ZQpedidos.ParamByName('no_venta').AsInteger:=numero;
  Dm.ZQpedidos.Open;
  if DM.ZQpedidos.Eof then begin
```

```
      lab_venta.Caption:='0.00';
      lab_inver.Caption:='0.00';
      lab_ganancia.Caption:='0.00';
      exit;
  end;
  venta:=0; ganancia:=0;
  DM.ZQpedidos.First;
  while not(DM.ZQpedidos.Eof) do begin
       venta:=venta+DM.ZQpedidos.fieldbyname('importe').AsCurrency;
       ganancia:=ganancia+DM.ZQpedidos.fieldbyname('ganancia').AsCurrency;
       DM.ZQpedidos.Next;
  end;
  lab_venta.Caption:= FormatFloat ('###,###,###,##0.00;-###,###,###,##0.00;0',
venta);
  lab_inver.Caption:= FormatFloat ('###,###,###,##0.00;-###,###,###,##0.00;0',
ganancia);
  lab_ganancia.Caption:= FormatFloat ('###,###,###,##0.00;-###,###,###,##0.00;0',
venta-ganancia);
end;
```

Función: Mostrar venta del día
Descripción: Enlista las ventas acaecidas durante el día y hace un corte de caja.

```
procedure TFormMain.Bt_hoyClick(Sender: TObject);
var
 consulta : Tstrings;
 fecha: string;
begin
 tabs.ActivePageIndex:=3;
 pnl_fprov.Visible:=false;
 bt_hoy.Down:=true;
 consulta:=Tstringlist.Create;
 fecha:=#39+FormatDateTime('yyyy-mm-dd',Now)+#39;
 consulta.Clear;
 consulta.Add('select id_venta, fecha,cliente, clientes.nombre, id_vendedor,
vendedores.nombre,');
 consulta.Add('(select sum(cantidad*p_compra) from pedidos where pedidos.id_venta
= ventas.id_venta) as inversion,');
 consulta.Add('(select sum(cantidad*p_venta) from pedidos where pedidos.id_venta
= ventas.id_venta) as importe ');
 consulta.Add('from ventas join clientes on cliente=id_cliente');
 consulta.Add('join vendedores on vendedor=id_vendedor');
 consulta.Add('where fecha='+fecha);
 consulta.Add('order by id_venta');

 DM.ZQventas.Close;
 DM.ZQventas.SQL:=consulta;
 reporte();
 consulta.Free;
 btnrep:=1;

end;
```

Función: Mostrar todas las ventas
Descripción: Muestra todas las ventas registradas en el sistema y hace el debido corte de caja.

```
procedure TFormMain.bt_vertodosClick(Sender: TObject);
begin
 DM.ZQuery.Close;
 DM.ZQuery.SQL.Clear;
 DM.ZQuery.SQL.Add('select productos.Clave, productos.Descripcion,');
```

```
 DM.ZQuery.SQL.Add('(productos.almacen+productos.exhibicion) as existencia,');
 DM.ZQuery.SQL.Add('productos.precio_compra, proveedores.empresa');
 DM.ZQuery.SQL.Add('from productos join proveedores');
 DM.ZQuery.SQL.Add('on (productos.id_proveedor=proveedores.id_proveedor)');
 DM.ZQuery.SQL.Add('where (productos.almacen+productos.exhibicion)<faltante');
 DM.ZQuery.Open;
 tabs.ActivePageIndex:=4;
 bt_vertodos.Down:=true;
 pnl_fprov.Visible:=false;
end;
```

Función: Mostrar las ventas de la semana:
Descripción: Enlista y obtiene corte de caja de las ventas acaecidas en la semana.

```
procedure TFormMain.Bt_semanaClick(Sender: TObject);
var
 consulta : Tstrings;
 f1,f2: string;
 fecha : Tdatetime;
begin
 tabs.ActivePageIndex:=3;
 pnl_fprov.Visible:=false;
 bt_semana.Down:=true;
 fecha:=StartOfTheWeek(now);
 f1:=#39+FormatDateTime('yyyy-mm-dd',fecha)+#39;
 fecha:=EndOfTheWeek(now);
 f2:=#39+FormatDateTime('yyyy-mm-dd',fecha)+#39;
 consulta:=Tstringlist.Create;
 consulta.Clear;
 consulta.Add('select id_venta, fecha,cliente, clientes.nombre, id_vendedor,
vendedores.nombre,');
 consulta.Add('(select sum(cantidad*p_compra) from pedidos where pedidos.id_venta
= ventas.id_venta) as inversion,');
 consulta.Add('(select sum(cantidad*p_venta) from pedidos where pedidos.id_venta
= ventas.id_venta) as importe ');
 consulta.Add('from ventas join clientes on cliente=id_cliente');
 consulta.Add('join vendedores on vendedor=id_vendedor');
 consulta.Add('where (fecha between '+ f1+ 'and '+f2+ ')order by id_venta');

 DM.ZQventas.Close;
 DM.ZQventas.SQL:=consulta;
 consulta.Free;
 reporte();
 btnrep:=2;

end;
```

Función: Mostrar ventas por fecha.
Descripción: Muestra las ventas en un determinado intervalo de tiempo con la opción de filtrarlas según el cliente.

```
procedure TFormMain.Bt_fechaClick(Sender: TObject);
var
 consulta : Tstrings;
 f1, f2: string;
begin

 pnl_fprov.Visible:=false;
 if FormFecha.ShowModal= mrOk then begin
     bt_fecha.Down:=true;
```

```
     consulta:=Tstringlist.Create;
     f1:=FormFecha.finicial;
     f2:=FormFecha.ffinal;
     consulta.Clear;
     consulta.Add('select id_venta, fecha,cliente, clientes.nombre, id_vendedor,
vendedores.nombre,');
     consulta.Add('(select sum(cantidad*p_compra) from pedidos where
pedidos.id_venta = ventas.id_venta) as inversion,');
     consulta.Add('(select sum(cantidad*p_venta) from pedidos where
pedidos.id_venta = ventas.id_venta) as importe ');
     consulta.Add('from ventas join clientes on cliente=id_cliente');
     consulta.Add('join vendedores on vendedor=id_vendedor');
     consulta.Add('where (fecha between '+ f1+ 'and '+f2+ ')order by id_venta');
     DM.ZQventas.Close;
     DM.ZQventas.SQL:=consulta;
     consulta.Free;
     reporte();
     if not tabs.ActivePageIndex=3 then tabs.ActivePageIndex:=3;
     btnrep:=3;
     intento:=0;
     timer1.Enabled:=true;
 end else begin
   intento:=0;
   timer1.Enabled:=true;
 end;

end;
```

Función: Mostrar faltantes por proveedor.
Descripción: Usando la información de la tabla Productos.faltante, determina si un artículo es faltante y lo enlista filtrando la información de acuerdo al proveedor especificado.

```
procedure TFormMain.bt_verxproveedorClick(Sender: TObject);
begin
 cbox_noprov.Items.Clear;
 cbox_prov.Items.Clear;
 Dm.ZQproveedor.Open;
 Dm.ZQproveedor.First;
 while not(DM.ZQproveedor.eof) do begin
        cbox_noprov.Items.Add(
Dm.ZQproveedor.fieldByName('id_proveedor').AsString );
        cbox_prov.Items.Add( Dm.ZQproveedor.FieldByName('empresa').AsString);
        Dm.ZQproveedor.Next;
 end;
 cbox_prov.ItemIndex:=0;
 Dm.ZQproveedor.Close;
 pnl_fprov.Visible:=true;
 cbox_prov.SetFocus;
 cbox_prov.ItemIndex:=0;
end;
```

Función: Mostrar faltantes
Descripción: Muestra todos los faltantes

```
procedure TFormMain.JvXPButton1Click(Sender: TObject);
var
 keyprov : string;
begin
 if cbox_prov.ItemIndex<0 then begin
   exit;
```

```
   bt_verxproveedor.Down:=false;
 end;
 keyprov:=cbox_noprov.Items.Strings[ cbox_prov.itemindex ];
 DM.ZQuery.Close;
 DM.ZQuery.SQL.Clear;
 DM.ZQuery.SQL.Add('select productos.Clave, productos.Descripcion,');
 DM.ZQuery.SQL.Add('(productos.almacen+productos.exhibicion) as existencia,');
 DM.ZQuery.SQL.Add('productos.precio_compra, proveedores.empresa');
 DM.ZQuery.SQL.Add('from productos join proveedores');
 DM.ZQuery.SQL.Add('on (productos.id_proveedor=proveedores.id_proveedor)');
 DM.ZQuery.SQL.Add('where (productos.almacen+productos.exhibicion)<faltante');
 DM.ZQuery.SQL.Add('and productos.id_proveedor='+keyprov);
 DM.ZQuery.Open;
 tabs.ActivePageIndex:=4;
 bt_verxproveedor.Down:=true;
 pnl_fprov.Visible:=false;
 btnrep:=5; intento:=0;
 timer1.Enabled:=true;

end;
```

Archivo: Ffiltro.pas
Descripción: Código fuente de la ventana que genera los filtros dinámicos sobre la pestaña inventario.

Función: Obtener datos.
Descripción: Obtiene la información de proveedores, marcas y categorías para mostrarlas en tres listas.

```
procedure TFormFiltro.obtenerdatos;
begin
 try
     ZQF.Close;
     ZQF.SQL.Clear;
     ZQF.SQL.Add('select * from categorias');
     ZQF.Open;
     while not(ZQF.eof) do begin
        lista_cat.items.add(ZQF.fieldByName('nombre').asString);
        lista_nocat.Items.Add(ZQF.fieldByName('clave').asString);
        ZQF.next;
     end;
     ZQF.Close;
     ZQF.SQL.Clear;
     ZQF.SQL.Add('select id_proveedor, empresa from proveedores');
     ZQF.Open;
     while not(ZQF.eof) do begin
        lista_prov.items.add(ZQF.fieldByName('empresa').asString);
        lista_noprov.Items.Add(ZQF.fieldByName('id_proveedor').asString);
        ZQF.next;
     end;
     ZQF.Close;
     ZQF.SQL.Clear;
     ZQF.SQL.Add('select clave, nombre from marcas');
     ZQF.Open;
     while not(ZQF.eof) do begin
        lista_marca.items.add(ZQF.fieldByName('nombre').asString);
        lista_nomarca.Items.Add(ZQF.fieldByName('clave').asString);
        ZQF.next;
     end;
```

```
    ZQF.Close;
  except;
    showmessage('No se puede accesar a la base de datos.');
    formFiltro.Close;
  end;
end;
```

Función: Construir consulta.
Descripción: Toma los diferentes parámetros que el usuario ha proporcionado mediante los controles del form y genera la consulta SQL, hace la consulta y dependiendo del resultado de esta muestra los artículos en la pestaña de inventario ó muestra una mensaje de advertencia en caso de que la consulta no genere ningún resultado.

```
procedure TFormFiltro.getsql;
var
 union, cond : string;
 consulta : Tstrings;
 anterior : boolean;
begin
 consulta:= TstringList.Create;
 if radio1.Checked=true then union:='AND ' else union:='OR ';
 consulta.Add('select productos.Clave, productos.Descripcion,
productos.almacen,');
 consulta.Add('productos.exhibicion, productos.precio_men,
productos.precio_may,');
 consulta.Add('productos.precio_esp, categorias.nombre from productos join
categorias');
 consulta.Add('on (productos.categoria=categorias.clave) WHERE');
 anterior:=false;

 If check_categoria.Checked then begin

cond:='productos.categoria='+lista_nocat.Items.Strings[lista_nocat.itemindex];
      consulta.Add(cond); anterior:= true; end;

 if check_marca.Checked then begin
      if anterior then cond:=union else cond:='';

cond:=cond+'productos.id_proveedor='+lista_noprov.Items.Strings[lista_noprov.item
index];
      consulta.Add(cond); anterior:= true; end;

 if check_proveedor.Checked then begin
      if anterior then cond:=union else cond:='';

cond:=cond+'productos.Marca='+lista_nomarca.Items.Strings[lista_nomarca.itemindex
];
      consulta.Add(cond); anterior:= true; end;

 if check_existencia.Checked then begin
      if anterior then cond:=union else cond:='';
      cond:=cond+'(productos.almacen+productos.exhibicion)';
      case cbox1.ItemIndex of
        0 : cond:=cond+'='+mask.Text;
        1 : cond:=cond+'<'+mask.Text;
        2 : cond:=cond+'>'+mask.Text;
      end;
      consulta.Add(cond); end;

  ZQF.Close;
```

```
  ZQF.SQL:=consulta;
  ZQF.Open;
  if ZQF.Eof then showmessage('No hay productos que cumplan'+#13+'con el criterio
de búsqueda')
  else begin
        DM.ZQproductos.Close;
        DM.ZQproductos.SQL:=consulta;
        DM.ZQproductos.Open;
        consulta.Free;
        Formmain.bt_filtro.Pressed:=true;
        FormFiltro.Close;
        Fmain.filtro:=true;
  end;

end;
```

1. SISTAR Caja.

Archivo: Fmain.pas
Descripción: Código fuente de la pantalla principal.

Función(es): Obtener productos, claves y clientes
Descripción: Obtiene el nombre de cada producto, claves, el nombre y número de cada cliente para asignarlos en las listas de clientes y productos respectivamente.

```
procedure Tform2.getproducts;
begin
 cbox_clave.Items.Clear;
 cbox_desc.Items.Clear;
 with DM do begin
    ZQtemp.Close;
    ZQtemp.SQL.Clear;
    ZQtemp.SQL.Add('SELECT clave, descripcion from productos where
(almacen+exhibicion)>0');
    ZQtemp.Open;
    while not ZQtemp.Eof do begin
          cbox_clave.Items.Add(Zqtemp.FieldByName('clave').AsString);
          cbox_desc.Items.Add(Zqtemp.FieldByName('descripcion').AsString);
          ZQtemp.Next;
    end;
    ZQtemp.Close;
 end;
end;

procedure Tform2.getkey;
begin
 with DM do begin
    ZQtemp.Close;
    ZQtemp.SQL.Clear;
    ZQtemp.SQL.Add('SELECT max(id_venta) as maximo from ventas');
    ZQtemp.Open;
    ventaid:=ZQtemp.fieldbyname('maximo').AsInteger;
    ZQtemp.Close;
 end;
end;

procedure TForm2.getclients;
begin
```

```
 with DM do begin
    ZQtemp.Close;
    ZQtemp.SQL.Clear;
    ZQtemp.SQL.Add('SELECT id_cliente, nombre from clientes ORDER BY nombre');
    ZQtemp.Open;
    ZQtemp.First;
    cbox_nocliente.Items.Clear;
    cbox_clientes.Items.Clear;
    while not ZQtemp.Eof do begin
      cbox_nocliente.Items.Add(Zqtemp.FieldByName('id_cliente').AsString);
      cbox_clientes.Items.Add(Zqtemp.FieldByName('nombre').AsString);
      ZQtemp.Next;
    end;
    ZQtemp.Close;
 end;
end;
```

Función: Obtener datos del cliente.
Descripción: Cuando un cliente es seleccionado, ya sea de la lista de clientes o de claves, esta función es ejecutada para extraer el resto de información del cliente.

```
procedure Tform2.datoscliente(clave: string);
begin
 with DM do begin
    ZQtemp.Close;
    ZQtemp.SQL.Clear;
    ZQtemp.SQL.Add('SELECT direccion, telefono, rfc from clientes where
id_cliente='+clave);
    ZQtemp.Open;
    if not ZQtemp.Eof then begin
      Lab_cliente.Caption:=Zqtemp.FieldByName('direccion').AsString + #13+
      'Tel. '+Zqtemp.FieldByName('telefono').AsString + #13 +
      'RFC: '+Zqtemp.FieldByName('rfc').AsString;
    end;
    ZQtemp.Close;
 end;
end;
```

Función: obtener precios
Descripción: Cuando se selecciona un producto, inmediatamente es invocada esta función para mostrar los tres precios del producto y así poder mostrarlos cuando el usuario desee seleccionar el tipo de precio.

```
procedure Tform2.getprecios;
var
 key : string;
begin
 if cbox_clave.ItemIndex<0 then exit;
  key:=cbox_clave.Items.Strings[cbox_clave.ItemIndex];
  with DM do begin
    ZQtemp.Close;
    ZQtemp.SQL.Clear;
    ZQtemp.SQL.Add('select precio_men, precio_may, precio_esp,
(exhibicion+almacen) as cant from productos');
    ZQtemp.SQL.Add('where clave='+#39+key+#39);
    ZQtemp.Open;
    if not ZQtemp.Eof then begin
      cbox_precio.Items.Clear;
      cbox_precio.Items.Add('MENUDEO
$'+Zqtemp.FieldByName('precio_men').AsString);
```

```
      cbox_precio.Items.Add('MAYOREO
$'+Zqtemp.FieldByName('precio_may').AsString);
      cbox_precio.Items.Add('TALLER
$'+Zqtemp.FieldByName('precio_esp').AsString);
      label_cant.Caption:=Zqtemp.FieldByName('cant').AsString;
      maxcant:=Zqtemp.FieldByName('cant').AsInteger;
      ed_cantidad.MaxValue:=maxcant;
    end;
    ZQtemp.Close;
 end;
end;
```

Función(es): Agregar pedido e insertar pedido
Descripción: Añade un pedido a la lista de pedidos y actualiza los detalles de la venta. La función primero analiza si el producto ya esta enlistado en los pedidos y solo aumenta su cantidad respetando el límite de existencia, de lo contrar inserta el producto como un nuevo pedido y para esto llama a la función insertar pedido.

```
procedure TForm2.BTaddClick(Sender: TObject);
var
 cantidad, pos, add : integer;
begin
  {Validando el pedido antes de agregarlo a la venta actual}
  if (cbox_clave.ItemIndex<0)or(cbox_precio.ItemIndex<0)or
       (ed_cantidad.Text='')or(cbox_desc.ItemIndex<0) then begin
cbox_desc.SetFocus; beep; exit; end;

  cantidad:=StrToInt(ed_cantidad.Text);
  if (cantidad=0)or(cantidad>maxcant) then begin ed_cantidad.SetFocus; beep;
exit; end;

  if venta.cont=300 then begin
         MessageDlg('Número máximo de artículos por venta alcanzado
(300)'+#13+#10+'Termine esta venta e inicie una nueva.', mtInformation, [mbOK],
0);
         exit;
  end;
  {Buscamos si ya hay un producto de la misma clave}
  pos:=existepedido(cbox_clave.Text);
  {Si pos >=0 el peiddo existe y solo lo aumentamos,  de lo contrario insertamos}
  if pos < 0 then insertarpedido
  else begin
     add:=Venta.cantidad[pos];
     if (cantidad+add)>maxcant then cantidad:=maxcant-add;
     aumentapedido(cantidad,pos);
  end;
end;

procedure Tform2.insertarpedido;
var
 cantidad : integer;
 clave, cadena   : string;
 precio : shortint;
 pcompra, pventa, importe: single;
 total : currency;
begin
   cantidad:=StrToInt(ed_cantidad.Text);
 {obtenemos los datos del artículo}
  with DM do begin
    ZQtemp.Close;
    ZQtemp.SQL.Clear;
```

```
    ZQtemp.SQL.Add('SELECT descripcion, precio_men, precio_may, precio_esp,
precio_compra from productos');
    ZQtemp.SQL.Add('where Clave='+#39+cbox_clave.Text+#39);
    ZQtemp.Open;
  end;
{1}if not DM.ZQtemp.Eof then begin
      cadena:=ed_cantidad.Text+' ; '+cbox_clave.Text+'; '+cbox_desc.Text;
      case cbox_precio.ItemIndex of
           0 : begin
               cadena:=cadena+';Menudeo';
               precio:=1;
               pventa:=DM.ZQtemp.fieldbyname('precio_men').AsFloat;
               end;
           1 : begin
               cadena:=cadena+';Mayoreo';
               precio:=2;
               pventa:=DM.ZQtemp.fieldbyname('precio_may').AsFloat;
               end;
           2 : begin
               cadena:=cadena+';Taller';
               precio:=3;
               pventa:=DM.ZQtemp.fieldbyname('precio_esp').AsFloat;
               end;
      end;
      pcompra:=DM.ZQtemp.fieldbyname('precio_compra').AsFloat;
      cadena:=cadena + ';$'+floattostr(pventa)+'
;$'+floattostr(pventa*cantidad)+' ;';
      clave:=cbox_clave.Text;
      importe:=cantidad*pventa;
      addpedido(cantidad, clave, precio, pcompra, pventa, importe);
      Grid.Items.Add( cadena );
      total:=totalventa();
      label_total.Caption:=FormatFloat ('###,###,###,##0.00;-
###,###,###,##0.00;0',total);
{.1}end;
  DM.ZQtemp.Close;
  cbox_clave.SetFocus;
end;
```

Función: Borrar pedido

Descripción: Quita un pedido de la lista de pedidos y actualiza los detalles de la venta.

```
procedure TForm2.delpedido(num : Smallint);
var
 i : integer;
begin
  i:=num;
  with venta do begin
        while(i<cont)do begin
                cantidad[i]:=cantidad[i+1];
                precio[i]:=precio[i+1];
                p_compra[i]:=p_compra[i+1];
                p_venta[i]:=p_venta[i+1];
                importe[i]:=importe[i+1]; inc(i);
        end;
        clave.Delete(num);
        dec(cont);
  end;
  Grid.Items.Delete(num);
  label_total.Caption:=FormatFloat ('###,###,###,##0.00;
###,###,###,##0.00;0',totalventa());
end;
```