UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

"ARAGON"

"DISEÑO Y APLICACION DE UN SISTEMA DE BASE DE DATOS"

T E S I S P R O F E S I O N A L

Que para obtener el Título de:

INGENIERO EN COMPUTACION

P r e s e n t a n

CARLOS BERNAL CHAVEZ

JOSE FERNANDO HILERIO ALFARO

San Juan de Aragón, Edo. de Méx. **1995**

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

DEDICATORIAS

Con todo mi amor y respeto a mis padres, porque me dieron la vida, el apoyo y la motivación para que llegara al fin de mis estudios profesionales.

A mis hermanos, por el gran amor quer les profeso.

A mi familia en general, por los importantes consejos que me han brindado siempre.

A mis amigos, por la compañía y consejos que me han brindado en los momentos difíciles.

Carlos Bernal Chávez

A mis padres con mucho cariño, por haberme brindado su amor, apoyo y comprensión para lograr este objetivo profesional, además de la dicha de vivir esta hermosa vida.

A mis hermanos, quienes se que me siguen alentando durante mi desarrollo profesional.

A mi familia y amigos, por sus importantes consejos en los momentos difíciles de mi carrera.

José Fernando Hilerio Alfaro

AGRADECIMIENTO

Agradecemos al Ingeniero

Martín Ordoñez Rosales
ENEP Aragón
UNAM

Por su valiosa colaboración para la realización de este trabajo de tesis.

CARTAS DE ACEPTACION

UNIVERSIDAD NACIONAL AVENOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGÓN
DIRECCION

CARLOS BERNAL CHAVEZ
P R E S E N T E .

En contestación a su solicitud de fecha 2 de febrero del año en curso, presentada por JOSE FERNANDO HILERIO ALFARO y usted, relativa a la autorización que se le debe conceder para que el señor profesor Ing. MARTIN ORDOÑEZ ROSALES pueda dirigirle su trabajo de Tesis denominado "DISEÑO Y APLICACION DE UN SISTEMA DE BASE DE DATOS", con fundamento en el punto 6 y siguientes del Reglamento para Exámenes Profesionales en esta Escuela, y toda vez que la documentación presentada por usted reune los requisitos que establece el precitado Reglamento; me permito comunicarle que ha sido aprobada su solicitud.

Aprovecho la ocasión para reiterarle mi distinguida consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPIRITU"
San Juan de Aragón, Edo. de Méx., Mayo 14 de 1993.
EL DIRECTOR

M. en A. CLAUDIO C. MERRIFIELD CASTRO

c c p Lic. Alberto Ibarra Rosas, Jefe de la Unidad Académica
c c p Ing. Juan Gastaldi Pérez, Jefe de Carrera de Ingeniería en Computación
c c p Ing. Manuel Martínez Ortiz, Jefe del Departamento de Servicios Escolares
c c p Ing. Martín Ordoñez Rosales, Asesor de Tesis

CCMC!AIR/jj'

UNIVERSIDAD NACIONAL AVENMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGÓN
DIRECCION

JOSE FERNANDO HILERIO ALFARO
P R E S E N T E .

En contestación a su solicitud de fecha 2 de febrero del año en curso, presentada por CARLOS BERNAL CHAVEZ y usted, relativa a la autorización que se le debe conceder para que el señor profesor, Ing. MARTIN ORDOÑEZ ROSALES pueda dirigirle su trabajo de Tesis denominado "DISEÑO Y APLICACION DE UN SISTEMA DE BASE DE DATOS", con fundamento en el punto 6 y siguientes del Reglamento para Exámenes Profesionales en esta Escuela, y toda vez que la documentación presentada por usted reune los requisitos que establece el precitado Reglamento; me permito comunicarle que ha sido aprobada su solicitud.

Aprovecho la ocasión para reiterarle mi distinguida consideración.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPIRITU"
San Juan de Aragón, Edo. de Méx., a 14 de Mayo de 1993.
EL DIRECTOR

M. en I. CLAUDIO C. MERRIFIELD CASTRO

c c p Lic. Alberto Ibarra Rosas, Jefe de la Unidad Académica
c c p Ing. Juan Gastaldi Pérez, Jefe de Carrera de Ingeniería en Computación.
c c p Ing. Manuel Martínez Ortiz, Jefe del Departamento de Servicios Escolares
c c p Ing. Martín Ordoñez Rosales, Asesor de Tesis

CCMCIAIR/jj'

INDICE

INTRODUCCION

El propósito primordial de este trabajo de tesis no es diseñar o desarrollar una aplicación administrativa, sino poner en práctica los métodos y algoritmos que nos lleven al establecimiento de una arquitectura de un sistema de base de datos. Para llegar a este objetivo, en los primeros ocho capítulos se lleva a cabo una recopilación de los fundamentos y principios más importantes a considerar sobre la teoría de bases de datos.

En el primer capítulo se presenta una introducción a las bases de datos, en donde se describen las principales características y objetivos de un sistema de base de datos. Del mismo modo se explican los conceptos básicos de las bases de datos, tales como: lenguajes de definición de datos, lenguajes de manejo de datos, manejo de base de datos, administrador de base de datos, usuarios de bases de datos, arquitectura de un sistema de bases de datos, diccionario de datos y diagrama de flujo de datos.

En el segundo capítulo se describen a detalle las definiciones y características principales de los diagramas de flujo de datos y del diccionario de datos.

En el tercer capítulo se presenta la teoría del modelo relacional, en el cual se describen los principios que lo rigen, su estructura y además se explican las relaciones entre los datos (tuplas) representados en tablas.

En el cuarto capítulo se presenta la teoría de otro de los modelos de bases de datos, llamado modelo jerárquico, se describe su arquitectura y principales características. Dentro de ellas se analizan los árboles jerárquicos, los cuales se representan en diagramas de estructura de árbol.

En el capítulo cinco se aborda el último modelo de bases de datos, llamado modelo de red. En este modelo los datos se representan por medio de una serie de entidades y sus relaciones por medio de ligas.

En el capítulo seis se presentan los fundamentos necesarios para el diseño lógico de bases de datos, de igual forma se exponen algunos aspectos importantes tales como: seguridad, concurrencia, caídas del sistema y tipos de fallas.

En el capítulo siete se presenta la teoría del diseño físico de bases de datos, entre los aspectos principales se describen los formatos de los registros y su especificación. Dentro de los formatos de los registros se analizan los registros de longitud fija y variable.

En el capítulo ocho se expone la teoría de las bases de datos distribuidas y bases de datos en microcomputadoras.

Debemos mencionar que nuestro principal interés está orientado a la manera en cómo se almacenan y accesan los datos más que en la funcionalidad y/o significado que estos puedan tener.

Para una mejor visión de nuestro planteamiento, se elaboró un sistema completamente funcional, cuyos alcances son muy limitados, pues como se mencionó, nuestro interés se enfoca a la estructura interna del sistema y no a la aplicabilidad real que este pueda tener. Por tanto, en el capítulo nueve se hace una presentación del análisis y diseño del sistema, así como de la arquitectura interna del mismo, además de los algoritmos diseñados para la captura y acceso de la información.

Los objetivos que se persiguen a través de la elaboración de este trabajo de tesis se mencionan a continuación:

1) Proporcionar los conceptos y principios en los que se fundamenta la teoría de las bases de datos, los cuales permitirán diseñar un sistema en el cual se ponga en práctica alguno de los modelos de las bases de datos.

2) Considerar algunos métodos de acceso a las bases de datos y poner en práctica el que consideremos más adecuado para nuestras necesidades.

3) Proporcionar un apoyo didáctico para los alumnos y profesores de la asignatura de bases de datos.

CAPITULO I

INTRODUCCION A LAS BASES DE DATOS

I.1. DEFINICION DE UNA BASE DE DATOS

En un sentido amplio, el término base de datos alude al conjunto entero de los datos almacenados por una empresa. Bajo una perspectiva más estrecha, la base de datos comprende únicamente aquella información que puede ser considerada como recurso común y que es tratada y está almacenada en un punto único de la red y puesta a disposición de todos los componentes de la empresa. Estos datos controlados y tratados de forma centralizada constituyen la base de datos común.

I.1.1. DEFINICION DE UN SISTEMA DE BASE DE DATOS (S.B.D)

Un Sistema de Base de Datos (S.B.D) consiste en un conjunto de datos relacionados entre sí y un grupo de programas para tener acceso a ellos.

Este grupo integrado de programas se utiliza para dar apoyo a bases de datos, y así, poder formar un Sistema de Base de Datos.

I.2. OBJETIVOS DE UN SISTEMA DE BASE DE DATOS

El objetivo primordial de un Sistema de Base de Datos es crear un ambiente en el que sea posible guardar y recuperar información en forma conveniente y eficiente.

Entre otros objetivos tenemos:

- El manejo de grandes cantidades de información.
- Cuidar la seguridad de la información almacenada en la base de datos.

- Proporcionar a los usuarios una visión abstracta de la información. Es decir, el sistema oculta ciertos detalles relativos a la forma de cómo los datos se almacenan y mantienen.

I.3. VENTAJAS Y DESVENTAJAS DE UN SISTEMA DE BASE DE DATOS

Las ventajas de los sistemas de base de datos repercuten en todas las etapas del ciclo de vida del sistema.

Entre las ventajas de un sistema de base de datos tenemos:

a) La etapa de diseño. Está facilitada por la estructura modular y por el hecho de que todos los programas de aplicación y los subsistemas de integridad y privacidad se refieren a las bases de datos en términos lógicos. Cada uno de estos módulos puede ser diseñado independientemente y los diseñadores no necesitan preocuparse de los detalles de implantación de la estructura física de almacenamiento utilizada.

b) La presencia de esquemas conceptuales y de bases de datos facilitan el empleo del sistema. Los usuarios finales pueden referirse a estos esquemas para aclarar su comprensión del área de aplicación y para descubrir qué datos estan almacenados en la base de datos.

c) El módulo de transformación lógica a física. Facilita en gran medida el mantenimiento del sistema. Tal como se mencionó, la introducción de nuevos equipos y/o sistemas operativos tiene impacto mínimo sobre el sistema si este módulo esta presente.

En la práctica pocos sistemas existentes se adecúan a la arquitectura ideal. Esto se debe en gran parte a la dificultad para diseñar el módulo de transformación lógica a física.

En general, los programas de aplicación comunican directamente con la estructura de almacenamiento de datos y los controles de privacidad e integridad estan inmersos en estos programas de aplicación.

Por otro lado, las bases de datos no son un claro beneficio. De hecho, las bases de datos tienen muchas desventajas, sobre todo las bases de datos a grandes escalas.

Algunas de estas se estudian a continuación:

a) Experiencia insuficiente en bases de datos. Quizá la desventaja más seria es que la tecnología de las bases de datos es compleja, y muy pocas organizaciones tienen personal con la suficiente experiencia para implantar y manejar las bases de datos correctamente. Esto aumenta la probabilidad de implantación sin éxito o con éxito parcial de las bases de datos.

b) Costos más altos de procesamiento de datos. Las bases de datos son responsables en general por un aumento significativo en el procesamiento de datos en una organización. En gran medida se debe a los programas del DBMS, que son grandes, deben procesarse para accesar, obtener y actualizar los datos, y en parte se debe a la cantidad de trabajo asociado con el manejo y la reorganización de los sistemas de archivos complejos.

c). Aumento en las necesidades de hardware y software. La capacidad de memoria de acceso directo, una mayor capacidad de comunicaciones (incluyendo paquetes de software de comunicaciones), y una capacidad adicional en el procesador se requieren en la mayoría de los sistemas de bases de datos. Por supuesto, esto aumenta en forma sustancial los costos de hardware.

d) Seguridad e integridad de los datos. La seguridad y la integridad de los datos son los principales problemas propios de una base. La mayoría de los problemas de seguridad e integridad se relacionan con el hecho de que muchos usuarios tienen derecho de acceso a la base de datos y que las capacidades asociativas de esta

última permiten que todos los datos estén disponibles en una forma u otra para cada usuario. Deben implantarse sistemas de seguridad para evitar al personal no autorizado su acceso a secciones de la base de datos.

Un problema serio relacionado con el anterior es que los usuarios de microcomputadoras buscan tener acceso regular a las bases de datos para obtener datos con el fin de manipularlos con sus microcomputadoras. Existe el peligro de que estos usuarios *contaminen* la base de datos alterando los archivos sin darse cuenta y también existe la preocupación de que una microcomputadora puede copiar datos con rapidéz en un disco flexible, el cual puede sacarse del lugar sin ser detectado.

También deben implantarse controles para que los archivos de las bases de datos puedan recontruirse a partir de otras fuentes de datos establecidas, especialmente para afrontar esta contingencia, tales como listados de transacciones colocados en cintas magnéticas. Estos sistemas de recuperación de bases de datos son complejos y tienen altos costos continuos de mantenimiento , pero son esenciales. En muchas situaciones un archivo de bases de datos perdido o destruido que no puede recontruirse llevaría a la organización a la bancarrota.

I.4. CONCEPTOS BASICOS DE UNA BASE DE DATOS

I.4.1. LENGUAJES DE DEFINICION DE DATOS

Un *lenguaje de definición de datos* (en inglés: DDL, data definition language) es aquel que se encarga del almacenamiento y métodos de acceso al sistema de base de datos.

Los lenguajes de definición de datos tienen medios para declarar los formatos de los registros, las estructuras de los archivos y los derechos de acceso del programa. A algunos

programas se les concede la facultad de leer los datos pero no de alterarlos; a otros se les puede dotar de facultades para leerlos y alterarlos.

I.4.2. LENGUAJES DE MANEJO DE DATOS

Un *lenguaje de manejo de datos* (en inglés: DML, Data Manipulation Language) permite al usuario manejar o tener acceso a los datos que estén organizados por medio del modelo apropiado.

El objetivo principal de un DML es lograr una interacción eficiente entre personas y el sistema.

Existen básicamente dos tipos de DML:

1) De procedimientos, en donde se requiere que el usuario especifique cuáles datos quiere y cómo deben obtenerse.

2) Sin procedimientos, en donde se requiere que el usuario especifique cuáles datos quiere sin especificar, cómo obtenerlos.

Los DML sin procedimientos son más fáciles de implantar, y se utilizan con más frecuencia aunque estos son menos eficientes ya que el usuario no tiene que especificar la forma de obtención de los datos.

I.4.3. MANEJO DE BASE DE DATOS

Para la simplificación, acceso y manejo de los datos de manera eficiente, requerimos de un módulo o parte fundamental dentro del sistema de base de datos, a este módulo se le denomina *manejador de base de datos.*

Un manejador de base de datos constituye la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicaciones y las consultas hechas al sistema.

Entre las tareas principales de un manejador de base de datos tenemos:

a) Interacción con el manejador de archivos. En donde el manejador de base de datos traduce las diferentes proposiciones en lenguaje de manejo de datos (DML) a comandos de sistema de archivos de bajo nivel.

b) Puesta en práctica de la seguridad. En este punto la labor del manejador de base de datos es conservar y aplicar la seguridad de los datos en el sistema de base de datos.

c) Control de concurrencia. Cuando varios usuarios actualizan la base de datos en forma concurrente, es posible que no se conserve la consistencia de los datos, así es que el manejador de la base de datos se encarga del control de la interacción entre los usuarios.

I.4.4. ADMINISTRADOR DE BASE DE DATOS

Un *Administrador de Base de Datos* (en inglés: DBA, Data Base Administrator) es la persona que tiene el control centralizado de los datos del sistema.

Entre las principales funciones de un DBA tenemos entre otras:

1) Definición de esquema. En donde se realiza la creación del esquema original de la base datos.

2) Definición de la estructura de almacenamiento y del método de acceso. Es decir, la creación de las estructuras de almacenamiento y métodos de acceso apropiados.

3) Concesión de autorización para el acceso a los datos. En el cual se conceden los diferentes tipos de autorización para el acceso a los datos a los distintos usuarios de la base de datos.

4) Especificación de las limitantes de integridad. Estas limitantes se conservan en una estructura especial del sistema que consulta el manejador de la base de datos cada vez que se lleva a cabo una actualización en el sistema.

I.4.5. USUARIOS DE LA BASE DE DATOS

Existen cuatro tipos diferentes de usuarios de un sistema de base de datos y se distinguen por el modo en que ellos esperan interactúar con el sistema, dichos tipos de usuarios son:

a) Programadores de aplicaciones.

b) Usuarios casuales.

c) Usuarios ingénuos.

d) Usuarios especializados.

I.4.6. ARQUITECTURA DE UN SISTEMA DE BASE DE DATOS

La arquitectura principal de un sistema de base de datos se divide en módulos que se encargan de cada una de las tareas del sistema general.

Entre los módulos los más importantes son:

- **El manejador de archivos.** Se encarga de la asignación de espacio en disco, así como de las estructuras de datos que va a emplear para representar la información almacenada.

- **El manejador de base de datos.** Tiene como función la conexión entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicaciones y las consultas que se hacen al sistema.

- **El procesador de consultas.** Traduce las proposiciones en lenguaje de consulta a instrucciones de bajo nivel que puede entender el manejador de la base de datos.

- **El precompilador de DML.** Convierte las proposiciones en DML incrustadas en un programa de aplicaciones en llamadas normales a procedimientos en lenguaje huésped. El precompilador debe interactúar con el procesador de consultas para generar el código apropiado.

- **El compilador de DDL.** Convierte las proposiciones en DDL en un conjunto de tablas que contienen metadatos. Tales tablas se almacenan después en el diccionario de datos.

La figura 1.4.6.1. esquematiza la estructura o arquitectura de un sistema de base de datos (S.B.D).

1.4.7. DICCIONARIO DE DATOS

Un diccionario de datos es una estructura de datos que almacena la información relativa a la base de datos. Se trata de un archivo especial del banco de datos que contiene las definiciones descriptivas de todos los elementos de datos, registros y archivos usados por el sistema.

Se usa frecuentemente por lo que debe tenerse mucho cuidado de desarrollar un diseño apropiado y una implantación eficiente.

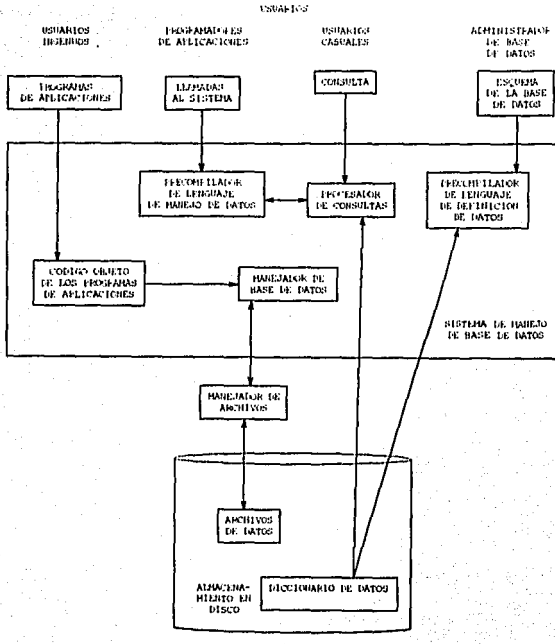


Figura 1.4.6.1. Estructura general de un Sistema de Base de Datos.

I.4.8. DIAGRAMA DE FLUJO DE DATOS

Un diagrama de flujo de datos es aquél en el que se representa toda una red de datos con varias transformaciones y flujo de datos.

El analista de sistemas usa los diagramas de flujo de datos para ilustrar el flujo y transformación de los datos a medida que son procesados por el sistema.

CAPITULO II

ANALISIS DEL CONTENIDO Y NATURALEZA DE LA INFORMACION

II.1. DIAGRAMA DE FLUJO DE DATOS

Los diagramas de flujo de datos auxilian al analista de sistemas a determinar en dónde los datos serán mantenidos de una transacción a la siguiente o almacenados permanentemente porque estos describen algunos aspectos del mundo fuera del sistema.

Estos indican el flujo de datos de un proceso a otro y ayuda a los analistas a determinar qué acceso inmediato a cada dato almacenado será utilizado por el usuario.

Los diagramas de flujo de datos son herramientas poderosas que pueden ser usadas por organizaciones para desarrollar arquitecturas de flujo de proceso para sus ambientes.

La administración puede utilizar esta herramienta para determinar dónde los datos son creados, dónde los datos van a ser usados, y quién utiliza el contenido de la información.

La administración puede utilizar los diagramas de flujo de datos para construir bases de datos completas y para almacenar los datos requeridos para las necesidades del usuario.

El administrador puede utilizar los procesos para transformarlos en flujo de datos.

El proceso puede ser descompuesto en funciones y actividades de las cuales los programas pueden ser codificados para manipular datos almacenados.

Los diagramas de flujo de datos son comunmentes utilizados en una base universal como el principal suministro de un proceso de análisis de sistema estructurado y de desarrollo.

Muchas organizaciones estan utilizando los diagramas de flujo de datos como base para un plan de sistema de negocios. Los diagramas de flujo de datos son también utilizados por varias organizaciones para demostrar e ilustrar sus necesidades de datos corporativo.

La administración puede utilizar la capacidad de separar procesos en varios niveles para determinar los requerimientos de procesamiento operacional de cada dato almacenado en la arquitectura del proceso. Por ejemplo, una organización esta utilizando los diagramas

de flujo de datos para determinar el procesamiento operacional (ordenamiento, grabación a otro medio de almacenamiento, borrado de archivos, creación de archivos de respaldo, etc.) este consumiendo mucho presupuesto y tiempo de la actividad operacional y corporativa.

La administración puede utilizar los diagramas de flujo de datos para auditar el diccionario de datos corporativo al igual que la actualización completa de éste.

Por ejemplo, un diccionario de datos completo puede tener datos acerca de todos los procesos y almacenar datos que existen en la organización. Para verificar el contenido y complemento del diccionario de datos existen los diagramas de flujo de datos. La suposición aquí es que los diagramas de flujo de datos son en sí complejos y representan la información completa y los procesos de la arquitectura de la corporación.

Los diagramas de flujo de datos pueden ser utilizados para la creación de especificaciones del desarrollo del sistema. El diagrama de flujo de datos muestra las fuentes y distinciones de los datos y por lo tanto indican los límites del sistema. Este identifica los nombres de las funciones lógicas, nombres de los elementos de datos, conexión de funciones a otras, almacenamiento de los datos y cómo cada función es accesada.

Cada diagrama de datos es analizado y sus estructuras y definiciones de sus componentes de datos son guardados en el diccionario de datos.

Cada función lógica puede ser descompuesta en más diagramas de flujo de datos, los contenidos de cada dato son analizados y almacenados en el diccionario de datos.

Estos documentos hacen más comprensivo el sistema y pueden ser utilizados por los administradores para construir sistemas o dar prioridad a la construcción de estos.

Los documentos y los diagramas de flujo de datos pueden tambien proveer un uso total en el mantenimiento de los sistemas existentes.

Finalmente los diagramas de flujo de datos pueden ser utilizados por los diseñadores de sistemas para preparar especificaciones funcionales que son:

a) Bien entendido y completamente acordado por usuarios.

b) Utilizado para configurar los requerimientos lógicos del sistema sin proponer implementación física.

c) Utiles en expresar preferencias comerciales.

Algunas organizaciones confirman que los diagramas de flujo de datos pueden prevenir muchos errores costosos en desarrollo de sistemas.

La figura 2.1.1 es un ejemplo de diagrama de flujo de datos.

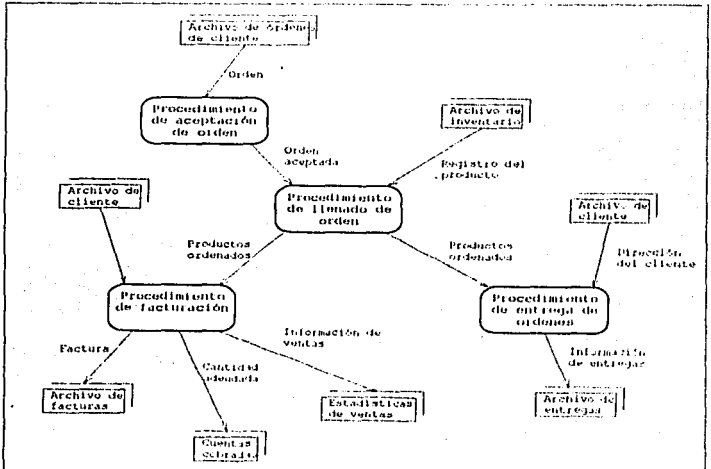


Figura 2.1.1. Ejemplo de un diagrama de flujo de datos.

II.2. DEFINICION Y CARACTERISTICAS DEL DICCIONARIO DE DATOS

Un esquema de base de datos se especifica por medio de una serie de definiciones que se expresan en un lenguaje especial llamado *lenguaje de definición de datos* (en inglés: DDL, Data Definition Language).

El resultado de la compilación de las proposiciones en DDL es un conjunto de tablas que se almacenan en un archivo especial llamado *diccionario de datos*.

Un diccionario de datos es un archivo que contiene metadatos, es decir, datos acerca de los datos. Este archivo se consulta antes de leer o modificar los datos reales en el sistema de base de datos.

El diccionario de datos almacena la información relativa a la estructura de la base de datos. Se usa constantemente, por lo que debe tenerse mucho cuidado de desarrollar un diseño apropiado y una implementación eficiente.

El diccionario de datos es la parte de la base de datos a la que se hace referencia más a menudo. Por ello, el manejador de buffer debe procurar no sacar los bloques del diccionario de datos de la memoria principal a menos que otros factores lo hagan necesario.

Algunas personas argumentan que el diccionario de datos también se puede poner en papel, ya sea a mano o a máquina. Pero el problema con un diccionario de datos manual es que es difícil satisfacer al diseñador que va a necesitar la definición de un campo de datos clasificado en diferentes maneras.

En un medio de base de datos, uno de los principales objetivos es el de que muchos usuarios compartan datos comúnes. Otro objetivo importante es proporcionar datos correctos a estos usuarios. Para realizar los objetivos de tener datos correctos, redundancia mínima y control del uso de los datos, es indispensable un mecanismo central de control. El diccionario de datos es un primer candidato para establecer y mantener estos controles.

Una ventaja adicional es que los sistemas establecidos que usan un diccionario de datos tienden a ser más efectivos y menos costosos de desarrollar.

El primer paso en el diseño de una base de datos es recabar información sobre la empresa, esto es, acerca del uso, relaciones y significado de los datos.

La herramienta que da la posibilidad de controlar y manejar la información sobre los datos en las fases de diseño, implantación, operación y expansión de una base de datos, es el diccionario de datos.

El diccionario de datos almacena información sobre los datos relativos al origen de estos, descripción, relación con otros datos, uso, responsabilidad y formato. Es la misma base de datos la que almacena datos sobre datos. El diccionario de datos es una guía y contiene el mapa de la ruta hacia la base de datos en lugar de *datos de la base.*

II.2.1. OBJETIVOS BASICOS DE UN DICCIONARIO DE DATOS

Uno de los objetivos básicos de un diccionario de datos es permitir el manejo y la documentación de los datos. Puesto que la base de datos sirve a varios usuarios, es vital que cada uno de ellos entienda precisamente qué son los datos y qué significan.

Es aconsejable comenzar a recabar la información sobre los datos en un diccionario de datos en el mismo día en que el proyecto se echa a andar. Tan pronto como el proyecto se pone en marcha, el diseñador empieza a hacerle a cada usuario preguntas tales como qué clase de sistema desea, qué información requiere del sistema y qué tipo de entrada puede proporcionar. Tan pronto como el usuario y el diseñador empiecen a platicar sobre las necesidades del primero, van a usar los nombres de los campos de datos. El diseñador y el usuario deben estar convencidos de que cuando usen algún término se refieran precisamente a lo mismo; de otra manera el diseñador puede construir un sistema que no es

el que el usuario desea. Esto destaca otro objetivo básico de un diccionario de datos; ayudar a establecer una comunicación efectiva entre el diseñador y el usuario, y entre usuarios.

El diccionario de datos se puede utilizar para almacenar la información sobre los campos de datos en un lugar central para establecer una comunicación efectiva entre todas las partes involucradas.

En la mayoría de las empresas, la dirección no tiene un control de los datos porque no hay una visión colectiva de estos. Para lograr este control, se tiene que recolectar la información sobre los datos en un lugar central. Sólo entonces la dirección puede empezar a controlar el uso de este recurso.

Así, los dos objetivos básicos de un diccionario de datos son la administración y el control de los datos como un recurso, en un lugar central, a través de las fases de diseño, realización y operación, así como el establecimiento de una comunicación efectiva entre todos los que estén interesados en la base de datos.

II.2.2. EL DICCIONARIO DE DATOS Y SUS INTERFACES

Consideraremos las interfaces de un diccionario de datos en un medio que dispone de un único sistema de manejo de la base de datos como en la figura 2.2.2.1.

En la etapa inicial de proceso del diseño de la base de datos, la función de administración de la base de datos interactuará con el diccionario de datos. Con la ayuda de un generador de informes, a la dirección y los usuarios se les proveerá de informes adaptados a las necesidades individuales de cada uno.

Los informes pueden contener información sobre lo siguiente:

a) Los campos de datos y las entidades.

b) Las relaciones entre los campos de datos y las entidades.

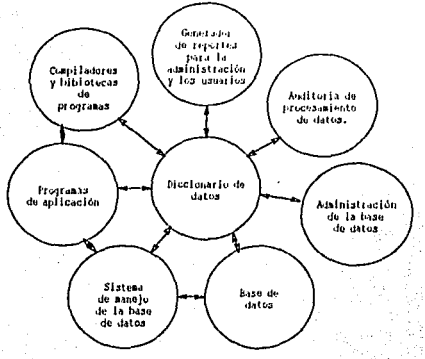


Figura 2.2.2.1. Interfaces del diccionario de datos en un medio de base de datos ideal.

c) Frecuencia del uso y las descripciones textuales de los elementos.

d) Información de control de acceso.

e) Informes de contabilidad del sistema.

f) Informes adecuados diseñados por los usuarios.

g) Informes sobre cambios.

h) Informes de error.

i) Informes de referencia cruzada.

La información sobre las relaciones de referencia cruzadas entre los campos de datos y los programas de aplicación se pueden almacenar en el diccionario de datos. La lista de

referencias cruzadas posibilitará a cualquier persona autorizada para informar a cualquier programador de aplicaciones o departamento sobre cualquier posible modificación antes de que se lleve a cabo, llegar a un acuerdo si es necesario y, entonces implementar el cambio. El cambio puede comprender contenido, formato o relaciones con otros campos de datos.

Los tipos de datos en la base de datos deberán actualizarse después de que el sistema de diccionario de datos tome nota de los cambios y los datos deberán actualizarse sólo cuando el DBMS los ha encontrado aceptables. Así, el diccionario de datos, el DBMS y la base de datos deberán formar un circulo estrecho.

El diccionario de datos puede ser utilizado como una herramienta efectiva para la función de administración de la base de datos en las fases de diseño, implantación y operación de la base de datos.

Un diccionario de datos es el lugar ideal para encontrar respuesta a preguntas como: dónde se usó, quién la usó y cuándo se usó, tomando en cuenta los campos de datos y las entidades.

Estas interfaces muestran que existen dos tipos de funciones del diccionario de datos. El primer tipo de uso lo realizan personas en sus funciones de: administrador de base de datos, programador de sistemas, analistas de sistemas, programadores de aplicaciones, dirección, usuario final y auditor. El otro tipo de uso lo realiza el software en las áreas: sistemas de manejo de bases de datos, compiladores, sistemas de operación y generadores de informes.

II.2.3. EL DICCIONARIO DE DATOS IDEAL

Lo siguiente es una lista de los requisitos deseables para formar un diccionario de datos ideal:

a) Modelo conceptual.

b) Modelo lógico.

c) Modelo interno.

d) Recursos de recuperación y de informe.

e) Captura de datos como entrada al diccionario de datos.

f) Información de control de acceso.

g) Apoyo de programas de servicio.

h) Generación de programas y código de descripción de datos.

i) Consistencia.

Un sistema ideal de diccionario de datos debe cumplir con los siguientes puntos:

1) El diccionario de datos debe apoyar a los modelos conceptual, lógico, interno y externo.

2) El diccionario de datos debe estar integrado en el sistema de manejo de base de datos.

3) El diccionario de datos debe contener varias versiones de la documentación, por ejemplo las versiones de prueba y la versión de producción.

4) El diccionario de datos debe prestar apoyo a la transferencia eficiente de información al sistema de manejo de la base de datos. Idealmente la conexión entre los modelos externos y el modelo interno debe hacerse en el momento de ejecución, cuando el diccionario de datos debe tomar la información de la versión de producción y crear en forma dinámica la descripción de la base de datos y la descripción de programa.

5) Un sistema de diccionario de datos debe mover la reorganización de la versión de producción de la base de datos como resultado de los cambios a la descripción de la base de datos. Similarmente cualquier cambio en las descripciones del programa deben reflejarse automáticamente en la biblioteca de descripción de programas con

la ayuda del diccionario de datos. Esto sucederá cuando el sistema de diccionario de datos sea una parte integral del DBMS. Los beneficios de usar un diccionario de datos están relacionados con la recopilación, especificación y manejo efectivos de los recursos totales de datos de una empresa. Un diccionario de datos debe ayudar al usuario de una base de datos a:

a) Comunicarse con otros usuarios.

b) Controlar los campos de datos de manera simple y efectiva, esto es, a introducir nuevos campos en los sistemas o a cambiar las descripciones de los campos.

c) Reducir la redundancia o inconsistencia de los datos.

d) Determinar el impacto de los cambios en los campos de datos sobre la base de datos total.

e) Centralizar el control de los campos de datos, como una ayuda en el diseño y en la expansión del diseño de la base de datos.

Además, un diccionario de datos ideal contiene información sobre otras entidades.

Almacena información sobre grupos de campos de datos, sobre la base de datos y sobre las referencias cruzadas entre los grupos de campos de datos y de las bases.

También indica qué programas se usan con la base de datos, y conserva información concerniente a los códigos de autorización y de seguridad.

CAPITULO III

MODELO RELACIONAL

III.1. PRINCIPIOS DEL MODELO RELACIONAL

El modelo de datos relacional representa la base de datos como un conjunto de tablas. Aunque las tablas son un concepto simple e intuitivo, existe una correspondencia directa entre el concepto de una tabla y el concepto matemático de una relación.

Algunos sistemas de gestión de base de datos relacionales son los siguientes (lo cual no significa que sean totalmente fieles a dicho modelo): *INGRES, SQL/DS, DB2, RDB, SUPRA, ULTRA, MIMER, ORACLE.*

En el modelo relacional una base de datos se compone de un cierto número de tablas llamadas *relaciones* (en inglés relations); cada columna de una tabla corresponde a un *atributo* (que equivale a un dato elemental) y al conjunto de valores que abarca un atributo se le llama *dominio* (en inglés domain). Al conjunto de los atributos de una relación se le llama intención de la relación, y al número de ellos, grado (en inglés degree) de la relación. A las filas de las relaciones se les llama *tuplas* o pares n-tuplas (duplas, cuádruplas, etc., según el grado de la relación). Se llama extensión de una relación al conjunto de la tuplas que tiene en un momento dado.

Un ejemplo de relación sería una llamada MATERIALES, que comprenda los datos a nivel de material, de los cuales sus atributos serían CODMAT (código de material), DENMAT (denominación del material) y CONMEN (consumo mensual) y los dominios respectivos serían el conjunto de serie de caracteres formados por dos letras y cuatro dígitos -por ejemplo-, el conjunto de series de caracteres de 40 posiciones y el conjunto de números de 6 dígitos. Un esquema de ésta relación sería el siguiente:

AM1404	CAL	000326
WB4330	ARENA	003260
EFO313	AGUA	032600

Ni el orden relativo de los atributos entre sí ni el de las tuplas tienen significado alguno. No se admiten dos atributos iguales en una misma relación (es decir, no se admite que haya dos o más valores del mismo atributo en la misma tupla, y una relación que cumpla esta condición se dice que está normalizada); tampoco se admite que haya dos tuplas iguales en la misma relación.

Que no se admitan tuplas iguales quiere decir que no se admitan tuplas que coincidan en los valores de todos sus atributos, pero puede ser que ni siquiera esté permitido que haya dos tuplas que tengan los mismos valores en unos determinados atributos, que pueden no ser todos; todo conjunto de atributos que cumpla esta condición se dice que es una clave candidata. Se dice que una clave candidata es no redundante si basta que se le suprima uno cualquiera de sus atributos para que deje de ser clave candidata; en lo que sigue sólo consideraremos claves candidatas no redundantes. A una de las claves candidatas escogida arbitrariamente se le llama *clave primaria*.

Se dice que un conjunto de atributos de una relación R es una clave foránea (en inglés foreign key) de la misma si los dominios de dichos atributos son los mismos que los de los atributos de la clave primaria de otra relación R'.

En la relación antes mencionada CODMAT es la única clave candidata no redundante, y por lo tanto será la clave primaria.

Las descripciones se suelen describir por medio de la siguiente notación: se da el nombre de la relación seguido de los nombres entre paréntesis de sus atributos, subrayándose los atributos componentes de la clave primaria. En el caso del ejemplo tendríamos:

MATERIALES (*CODMAT*, DENMAT, CONMEN)

En una base de datos relacional se suelen definir tres tipos de condiciones de integridad:

1) Que ninguna tupla pueda tener un valor nulo en ningún componente de la clave primaria.

2) Que no haya dos tuplas con igual clave primaria.

3) Que todo valor de una clave foránea aparezca también como valor de la clave primaria en una tupla de la relación R' antes mencionada; se dice que esto es una condición de integridad referencial.

III.2. ESTRUCTURA DE LAS BASES DE DATOS RELACIONALES

Los matemáticos definen a una relación como un subconjunto de un producto cartesiano de un listado de dominios. Esto corresponde casi exactamente a la tabla de la figura 3.2.1 . La única diferencia es que aquí se asignaron nombres a los atributos, mientras que los matemáticos se basan en nombres numéricos, usando el entero 1 para denotar el atributo cuyo dominio aparece en el primer lugar en el listado de dominios; para el atributo 2 cuyo dominio aparece en segundo lugar, etcétera.

Debido a que las tablas son básicamente relaciones se utilizan los términos matemáticos relación y tupla en vez de los términos tabla y columna.

En la relación depósito de la figura 3.2.1 hay siete tuplas. Sea la variable tupla t la primera tupla de la relación. Se utilizará la notación t[nombre-sucursal] para denotar el valor en t del atributo nombre sucursal.

nombre-sucursal	número-cuenta	nombre-cuentahabiente	saldo
Centro	101	Johson	500
Mianus	215	Smith	700
Perrydge	102	Hayes	400
Round Hill	305	Turner	350
Perrydge	201	Williams	900
Redwood	222	Lindsay	700
Brighton	217	Green	750

Figura 3.2.1. Relación depósito.

Así, t[nombre-sucursal] = centro.

De manera similar, t[número-cuenta] expresa el valor de t del atributo número-cuenta, etcétera.

Cuando se habla de una base de datos, debe diferenciarse entre el esquema de la base de datos; es decir, el diseño lógico de la base de datos y una instancia de la base de datos, que se constituye con la información contenida en la base de datos en cierto momento. Así, una variable en los lenguajes de programación corresponde al concepto de una instancia de una relación.

Es conveniente nombrar el esquema de una relación en la misma forma en que se dan nombres a las definiciones de tipo en los lenguajes de programación. Se toma la convención de usar minúsculas para las relaciones y la primera letra mayúscula para los esquemas de relaciones.

Siguiendo la convención anterior podemos indicar la relación depósito de la siguiente manera:

Esquema-depósito = (nombre-sucursal,número-cuenta,nombre-cuentahabiente, saldo)

En general, el esquema de una relación es una lista de atributos y sus correspondientes dominios. Supongamos que depósito es una relación con el esquema Esquema-depósito entonces se escribe: .

depósito(Esquema-depósito)

Cuando deseamos definir los dominios se utiliza la notación:

(nombre-sucursal:cadena, número-cuenta: entero, nombre cuentahabiente: cadena, saldo: entero)

para definir el esquema de relación para la relación depósito.

Una base de datos relacional consiste en un conjunto de tablas, que tienen asignado un nombre único.

Una columna de una tabla representa una relación entre un conjunto de valores.

Puesto que una tabla es un conjunto de estas relaciones, existe una correspondencia entre el concepto matemático de relación, del cual recibe su nombre el modelo de datos relacional.

Examinemos la tabla de depósito de la figura 3.2.1. tiene cuatro atributos: nombre-sucursal, número-cuenta, nombre-cuentahabiente, saldo. Para cada atributo, existe un conjunto de valores permitidos llamado dominio de ese atributo. Por ejemplo, para el atributo nombre-sucursal, el dominio sería el conjunto de todos los nombres de las sucursales.

Sea D1 este conjunto y sea D2 el conjunto de todos los números de cuenta, D3 el conjunto de todos los nombres de los cuentahabientes y D4 el conjunto de todos los saldos, cada una de las columnas de depósito debe componerse de una tupla de 4 (V1,V2,V3,V4) donde V1 es un nombre de sucursal (es decir, V1 está en el dominio D1); V2 es un número de cuenta (es decir, V2 está en el dominio de D2); V3 es un nombre de cuentahabiente (es

decir, V3 está en el dominio D3), y V4 es un saldo (es decir, V4 está en el dominio D4). En general, depósito va a contener únicamente un subconjunto del conjunto de todas las columnas posibles.

Una base de datos relacional supone que todos los datos dentro de una base de datos, potencialmente están relacionados entre sí.

Los usuarios definen las relaciones entre los datos mientras interactúan con el sistema de computadora. Se expresan estas relaciones en forma de tablas. Cuando los usuarios vean perspectivas diferentes de estos elementos de datos, pueden refinar el contenido de las tablas y secuenciarlo en varias formas.

Veamos el siguiente ejemplo, las relaciones en las bases de datos de pedidos de clientes se podrían listar en varias formas:

En orden del número de cliente, en orden de la cantidad del pedido con la mayor cantidad primero o en orden de la fecha del pedido con la fecha más antigua primero.

La siguiente tabla es formada con la base de datos antes mencionada. Los datos están ordenados según la fecha del pedido, con el pedido más antiguo primero.

Base de datos de pedidos de ventas al cliente
Orden con relacion a la fecha

Número de pedido	Número del cliente	Fecha de pedido	Cantidad de pedido
37629	394	10/06/82	$123.87
44908	105	11/15/82	$256.98
57923	814	12/01/82	$ 99.08
82999	249	01/31/83	$824.47

Otra relación que se podría definir para otros datos está en esta misma base de datos de pedidos de cliente es la tabla siguiente:

Se han ordenado los elementos de datos por orden del número de cliente.

Simplemente, reordenando esta tabla con relación al cliente, un director ve

Base de datos de pedidos de clientes Orden con relacion al cliente			
Número del cliente	Cantidad del pedido	Fecha del último pedido	Fecha del último embarque
104	$632.11	02/07/83	01/15/83
105	$256.98	11/15/82	10/30/82
106	$124.50	12/17/83	01/11/83
106	$743.25	02/19/83	01/11/83

perspectivas diferentes: qué clientes compran la mayor parte de un producto, qué clientes tienen pedidos de hace tiempo o permanentes, etc.

Los datos también pueden participar en más de una relación a la vez. Las relaciones se pueden clasificar en cualquier orden cuando las necesite el usuario del sistema de computadora. También se pueden combinar relaciones para obtener nuevas relaciones. Por ejemplo, un gerente podría utilizar una estructura relacional de datos para crear una lista de todos los clientes que tienen encargos por más de 500 dólares y que no han recibido un envío durante tres meses, combinando la tabla de pedidos con relación a la fecha y la tabla con relación al cliente en las cifras que los acompañan es posible crear todas las relaciones que hagan falta.

Como comentario final en éste capitulo es importante mencionar un lenguaje de consulta comercial llamado **SQL**, ya que su estructura esta basada en el modelo relacional. A continuación se da una breve introducción de éste importante lenguaje.

SQL se introdujo como lenguaje de consulta del Sistema R. El nombre **SQL** está formado por las iniciales en inglés de "lenguaje de consulta estructurado" (structured query language). Todavía se le conoce con su antiguo nombre, Sequel.

La estructura básica de una expresión en **SQL** se compone de tres cláusulas: **select** (elegir), **from** (de) y **where** (donde).

La cláusula **select** corresponde a la operación de proyección del álgebra relacional. Sirve para listar todos los atributos que se desean en el resultado de una consulta.

La cláusula **from** es una lista de las relaciones que se van a examinar durante la ejecución de la expresión.

La cláusula **where** corresponde al predicado de selección de álgebra relacional. Se compone de un predicado que incluye atributos de las relaciones que aparecen en la cláusula **from**.

CAPITULO IV

MODELO JERARQUICO

IV.1. ARQUITECTURA DEL MODELO JERARQUICO

En el *modelo jerárquico* los datos y las relaciones se representan por medio de registros y ligas. En el modelo jerárquico los registros se organizan para formar conjuntos de árboles, en vez de gráficas arbitrarias.

Una base de datos jerárquica consiste en un conjunto de registros que se conectan entre sí por medio de ligas. Una liga es una asociación entre dos registros exclusivamente.

Para ilustrar lo anterior, piénsese en una base de datos que representa a una relación cuentahabiente-cuenta en un sistema bancario. El tipo de registro cuentahabiente consta de tres campos: nombre, calle y ciudad. De manera similar el registro cuenta consiste de dos campos: número y saldo.

En la figura 4.1.1. se presenta un ejemplo de esta base de datos. Indica que el cuentahabiente Lowman tiene la cuenta 305, Camp las 226 y 177 y Kahn la 155.

Nótese que el conjunto de todos los registros de cuentahabientes y cuentas está organizado en forma de un árbol con raíz, en el cual ésta última es un nodo de trabajo.

Como se verá más adelante, una base de datos jerárquica está formada por un conjunto de árboles de este tipo, formando así un bosque.

El contenido de un registro específico puede repetirse en varios lugares. Por ejemplo, en el sistema bancario cuentahabiente-cuenta, una cuenta puede pertenecer a varios clientes. La información correspondiente a esa cuenta, o la relativa a los cuentahabientes a los que puede pertenecer, tendrá que repetirse. Esta repetición puede darse tanto en el mismo árbol de base de datos como en varios árboles distintos. La repetición de registros tiene dos desventajas principales:

1) Puede producirse una inconsistencia de los datos al llevar a cabo la actualización.

2) Será inevitable el desperdicio de espacio de memoria.

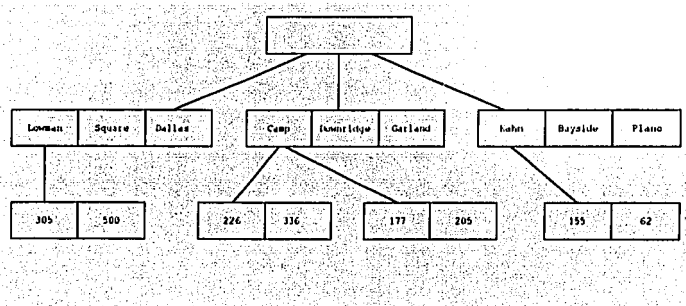


Figura 4.1.1. Ejemplo de base de datos.

IV.2. ESTRUCTURA DE DATOS DEL MODELO JERARQUICO

Una estructura de datos define relaciones entre registros sobre la base de que ciertos registros deben existir antes que otros. Por ejemplo, el registro maestro de un empleado que se genera cuando se contrata a un empleado, tiene que existir antes de que haya cualquier otro registro sobre el empleado dentro de la organización. Este es un tipo de estructura normal, y los términos dados a las relaciones ayudan a aclarar la idea. Un registro que tiene que existir antes de cualquier otro se llama registro-padre, y el registro dependiente se llama registro-hijo o registro de descendencia. Cuando haya más de un registro hijo, cada uno con independencia debiendo su existencia al mismo registro-padre se llaman fraternales.

El registro de nómina de un empleado y su registro de cumplimiento serán los registros hijos del registro maestro del empleado (figura 4.2.1).

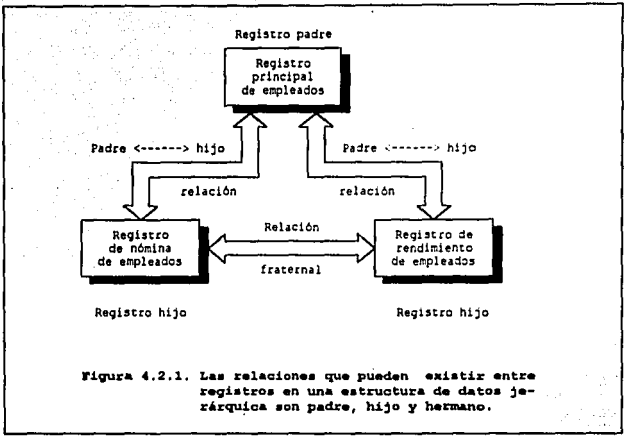


Figura 4.2.1. Las relaciones que pueden existir entre registros en una estructura de datos jerárquica son padre, hijo y hermano.

Los niveles de las relaciones padre-descendencia pueden ser bastante profundas.

Por tanto, mientras un registro puede ser hijo, en relación con un registro, puede ser padre de otros registros. En estructuras jerárquicas, el primer registro padre en la jerarquía se llama registro raíz. Los padres siempre tienen que existir antes que sus hijos. Los registros de nivel alto tienen que existir antes de que los registros de nivel bajo puedan añadirse a la estructura. Una razón por la cual ello es tan importante es que impide la obtención de registros no autorizados, tal como sería un registro de nómina para alguien que no ha sido contratado y por tanto, no tiene registro maestro de empleado (registro raíz).

Muchas aplicaciones de proceso de datos utilizan ficheros construídos a partir de juegos de registros padre e hijo, en donde los registros padre describen alguna información general sobre entidades o sucesos y los registros de descendencia describen los detalles sobre estas entidades y sucesos, sin duplicar ningún dato que existe en el registro-padre. El

evitar la duplicación crea una ventaja especial de conservación de espacio para estructuras jerárquicas de datos cuando se comparan con otros tipos de estructuras.

Cada registro en una estructura jerárquica tiene que tener su propia clave singular relativa a su pariente, pero no es necesario que sea singular a lo largo de la estructura entera. En el ejemplo de la nómina, es posible que dos personas que fueron contratadas en la misma fecha con el mismo pago tuvieran dos nóminas idénticas. Sin embargo, sus registros padre serían singulares puesto que representan a empleados diferentes. Por tanto, no hay mucha dificultad en distinguir un registro de nómina de otro, cuando ya se conoce la clave del registro padre. Como se puede ver en el ejemplo, cada registro descendencia es una jerarquía, además de tener su propia clave, se puede calificar aún más si especificamos las claves de sus padres a lo largo de la jerarquía. De este modo, se puede localizar la singularidad en partes pequeñas de la base de datos.

Como las estructuras jerárquicas permiten la localización de la singularidad, el sistema puede proporcionar protección y seguridad a pequeños subconjuntos de toda la base de datos. Se pueden establecer reglas en donde usuarios determinados del sistema de computadora sólo pueden tener acceso a algunos de los registros descendencia del registro-padre. Por ejemplo, se puede dar acceso al registro de entrenamiento al personal en entrenamiento mientras se les niega el acceso a la descendencia del registro de nóminas. Esta estructura facilita la administración de la tarea global de gestión de datos del sistema.

Un diagrama de estructura de árbol es el esquema de una base de datos jerárquica.

Este tipo de diagrama está formado por dos componentes básicos:

1) **Cuadros**, que corresponden a tipos de registros.

2) **Líneas**, que corresponden a ligas.

Un diagrama de estructura de árbol tiene la función de especificar la estructura lógica general de la base de datos. Un diagrama de estructura de árbol es similar a uno de

estructura de datos. La diferencia principal radica en que en este último los tipos de registro se organizan en forma de una gráfica arbitraria, mientras que en el primero se organizan en forma de un árbol con raíz.

Es necesario definir previamente lo que quiere decir el término árbol con raíz. En primer lugar, la gráfica no puede contener ciclos. En segundo lugar, las relaciones entre padre e hijo sólo pueden ser uno a muchos o uno a uno. La forma general de un diagrama de estructura de árbol se ilustra en la figura 4.2.2.

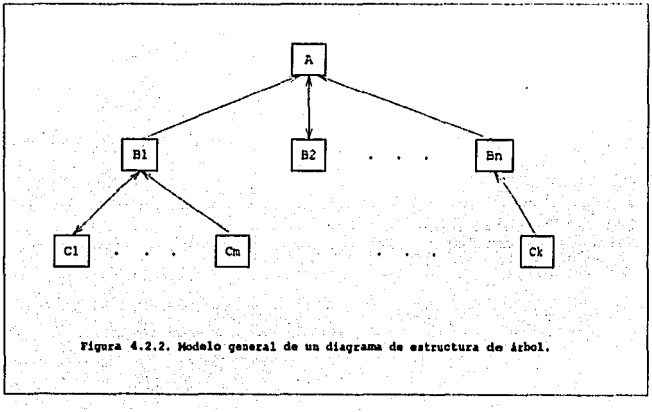


Figura 4.2.2. Modelo general de un diagrama de estructura de árbol.

Obsérvese que las flechas apuntan de hijos a padres. Es posible que una flecha apunte de un padre a un hijo, pero el hijo siempre debe tener una flecha que apunte a su padre.

El esquema de la base de datos se representa como un conjunto de diagramas de estructura de árbol. Para cada diagrama existe únicamente una instancia del árbol de base de datos.

La raíz de este árbol es un nodo instrumental. Los hijos de este nodo son instancias del tipo de registro correspondiente. Cada una de estas instancias puede tener a su vez varias instancias de diversos tipos de registro.

Considere un árbol genealógico en el cuál los padres pueden no tener hijos, tener uno o más de uno. Si hay hijos, estos mismos pueden tener sus propios hijos.

Los componentes de una base de datos que usa un modelo jerárquico como estructura fundamental se muestra en la figura 4.2.3.

La estructura jerárquica de árbol se construye con nodos y ramas. Un nodo es una colección de atributos de datos que describen a la entidad en ese nodo. El nodo más alto de una estructura jerárquica de árbol se conoce como *raíz.*

Los nodos dependientes se encuentran en niveles más bajos en el árbol. El nivel de estos nodos depende de su distancia del nodo raíz (figura 4.2.4.)

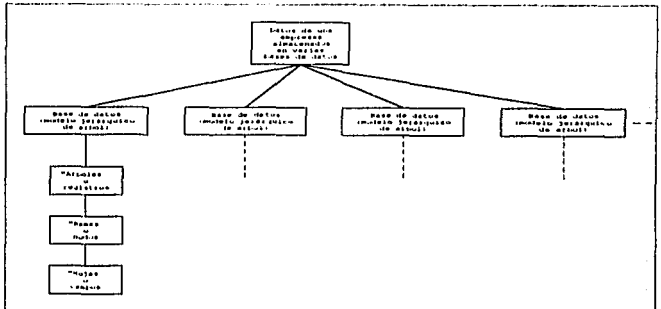


Figura 4.2.3. Los datos de una empresa pueden estar almacenados en varias bases de datos. Cada base de datos tiene un modelo jerárquico de árbol. Un numero de "ramas" o registros forman una base de datos. Un arbol componente de varios "ramas" o nodos. Cada rama tiene varias "hojas" o campos.

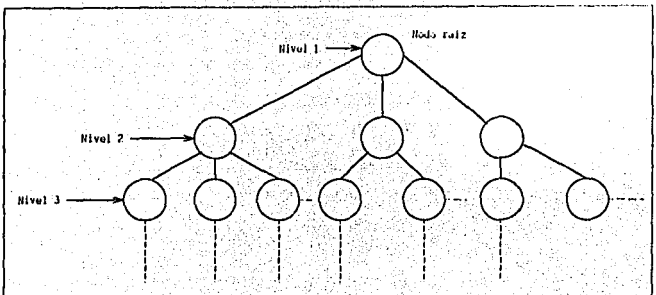


Figura 4.2.4. Una estructura jerárquica está constituida de nodos o ramas. Un nodo es una colección de atributos de datos que describen a la entidad en ese nodo. El nodo más alto de una estructura jerárquica de árbol se llama el nodo "raíz" (tipo de la entidad dominante). El nodo raíz está en el primer nivel. Los nodos dependientes (tipo de las entidades subordinandas) están en el segundo, tercer..., niveles.

Un modelo jerárquico es un modelo que organiza a los datos en una estructura jerárquica de árbol. Cada vez que aparece el nodo raíz da principio a un registro lógico de base de datos, esto es, una base de datos jerárquica esta constituida de varios árboles. En un modelo jerárquico los nodos que se encuentran en el nivel 2 se conocen como hijos del nodo que se encuentra en el nivel 1, y el nodo del nivel 1 se conoce como el padre de los nodos en el nivel 2. Los nodos en el nivel 3, que corresponden a un nodo en el nivel 2 se conocen como hijos del nodo del nivel 2, y el nodo en el nivel 2 se conoce como padre y así sucesivamente.

Una estructura jerárquica de árbol tiene que satisfacer las condiciones siguientes:

1) Un modelo jerárquico siempre comienza con una raíz.

2) Cada nodo consiste de uno o más atributos que describen a las entidades en ese nodo.

3. Los nodos dependientes pueden aparecer en dos niveles consecutivos. El nodo en el nivel precedente se convierte en el nodo padre de los nuevos nodos dependientes. Los nodos dependientes se pueden añadir tanto horizontal como verticalmente sin ninguna limitación (figura 4.2.4.). El nivel 1 sólo puede tener un nodo, el que llamamos nodo raíz.

4. Cada nodo que se presenta en el nivel 2 tiene que conectarse con uno y sólo uno de los nodos que se presentan en el nivel 1.

Cada nodo que se presenta en el nivel 3 tiene que conectarse con uno y solo uno de los nodos que se presentan en el nivel 2, y así sucesivamente. Debido a que como máximo sólo puede haber una conexión (o unión) entre dos nodos cualesquiera, los arcos no necesitan tener flechas de orientación.

5. Un nodo padre puede tener uno o varios nodos hijos bajo su dependencia. Si no tiene ningún nodo bajo su dependencia no es un nodo padre.

6. Cada nodo excepto la raíz, tiene que accesarse a través de su nodo padre. El nodo representado en una verdadera jerarquía debe recuperarse sólo a través de su padre, ya que en este radica el verdadero significado de la existencia de esos datos. Por lo tanto, la trayectoria de acceso es única para cada nodo dentro de un modelo jerárquico (figura 4.2.5.). Por ejemplo, el nodo I puede alcanzarse sólo a través de la trayectoria de A a G, de G a H y de H a I. El nodo D puede accesarse sólo a través de la trayectoria de A a B y de B a D. Por lo tanto, un modelo jerárquico consiste de trayectorias lineales.

7) Un nodo puede aparecer varias veces en cada nivel. Cada vez que aparece un nodo (excepto cuando aparece en el nodo raíz) se tiene que conectar con su nodo padre, esto es, puede presentarse muchas veces el nodo A. Cada vez que se

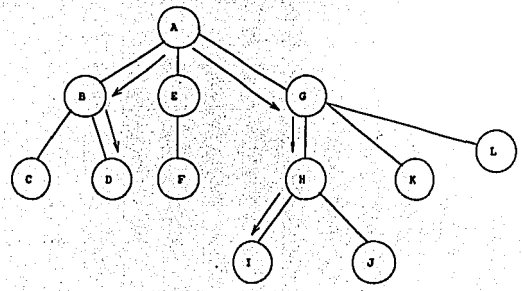


Figura 4.2.5. La trayectoria de acceso dentro de un modelo jerárquico es única. Por ejemplo, el nodo I sólo a través de la trayectoria de A a B y de B a D. Por lo tanto, un modelo jerárquico consiste en trayectorias lineales.

presenta el nodo A da principio a un registro lógico. Puede no aparecer, aparecer una o muchas veces el nodo B por cada vez que aparezca el nodo A, y así sucesivamente.

CAPITULO V

MODELO DE RED

V.1. ARQUITECTURA DEL MODELO DE RED

En el modelo relacional, los datos y las relaciones entre los datos se representan con una serie de tablas. El modelo de red se diferencia del relacional en que los datos se expresan por medio de una serie de *registros*, y las relaciones entre los datos mediante *ligas*.

Una base de datos de red consiste en una serie de registros que están conectados entre sí por medio de ligas (links). Todo registro es un conjunto de campos (atributos), cada uno de estos contiene únicamente el valor de un dato. Una liga es una asociación entre dos registros exclusivamente. Así pues, una liga puede considerarse como una forma restringida (binaria) de relación.

Para ilustrar esto, piénsese en una base de datos que representa una relación *cuentahabiente-cuenta* en un sistema bancario. Existen dos tipos de registro, *cuentahabiente* y *cuenta*.

En la figura 5.1.1 se muestra un ejemplo de base de datos. Puede saberse que Lowman tiene la cuenta 305, Camp las 226 y 177, y Kahn la cuenta 155.

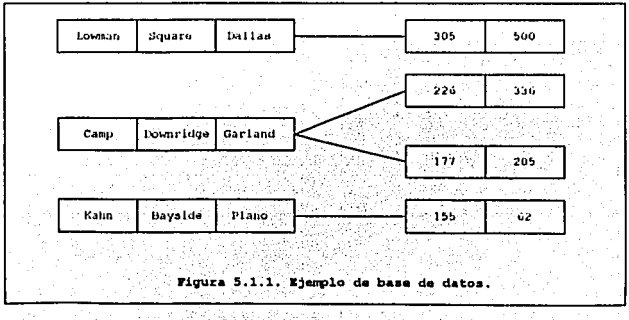


Figura 5.1.1. Ejemplo de base de datos.

V.1.1. DIAGRAMAS DE ESTRUCTURA DE DATOS

Un *diagrama de estructura de datos* es un esquema que representa el diseño de una base de datos de red. Estos diagramas eslan formados por dos componentes básicos:

* **Cuadros**, que corresponden a tipos de registro.

* **Líneas**, que corresponden a ligas.

Un diagrama de estructura de datos especifica la estructura lógica general de la base de datos.

V.1.2. RELACION BINARIA

Considérense dos conjuntos de entidades, *cuentahabiente* y *cuenta*, relacionados por medio de una relación binaria muchos a muchos, llamada *CthabCta*, que no tiene atributos descriptivos. La relación anterior especifica que un *cuentahabiente* puede tener varias cuentas y que cada una de estas puede pertenecer a varios cuentahabientes. El diagrama de estructura de datos correspondiente se ilustra en la figura 5.1.2.1. El tipo de registro *cuentahabiente* corresponde al conjunto de entidades *cuentahabiente*. Incluye los campos *nombre*, *calle* y ciudad. De manera similar, *cuenta* es el tipo de registro que corresponde al conjunto de entidades *cuenta*. Incluye los dos campos *número* y *saldo*. Finalmente, el conjunto de relaciones *CthabCta* se ha sustituido por la liga *CthabCta*.

La relación *CthabCta* es muchos a muchos. Si la relación *CthabCta* fuera de uno a muchos, de *cuentahabiente* a *cuenta*, entonces la liga *CthabCta* tendría una flecha apuntando hacia el tipo de registro *cuentahabiente* (figura 5.1.2.2a). De manera similar si la relación *CthabCta* fuera de uno a uno, entonces la liga *CthabCta* tendría dos flechas, una

apuntando hacia el tipo de registro *cuenta* y la otra apuntando hacia el tipo de registro *cuentahabiente* (figura 5.1.2.2b).

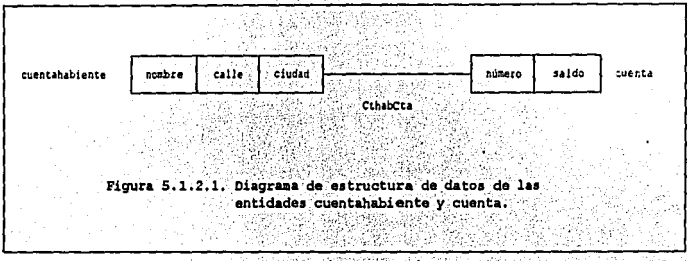


Figura 5.1.2.1. Diagrama de estructura de datos de las entidades cuentahabiente y cuenta.

Como en el diagrama de estructura de datos de la figura 5.1.2.1 la relación *CthabCta* es muchos a muchos, no se trazarán flechas en la liga *CthabCta*.

Por lo tanto, una instancia de una base de datos que correspondiera al esquema descrito podría contener varios registros *cuentahabiente* ligados a algunos registros *cuenta*, como se observa en la figura 5.1.2.2.

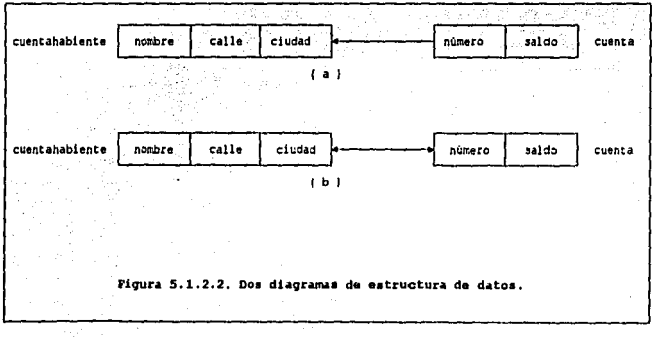


Figura 5.1.2.2. Dos diagramas de estructura de datos.

En la figura 5.1.2.3 se muestra un ejemplo de base de datos correspondiente al diagrama de estructura de datos de la figura 5.1.2.1. Puesto que la relación es muchos a muchos, se indica que Katz tiene las cuentas 256 y 347, y que la cuenta 347 es tanto a Katz como a Doner.

Finalmente, en la figura 5.1.2.4 se ilustra un ejemplo de base de datos que corresponde al diagrama de estructura de datos de la figura 5.1.2.3b. Puesto que la relación es uno a uno, una *cuenta* puede pertenecer únicamente a un cuentahabiente, y este puede tener exclusivamente una cuenta, como puede apreciarse en el ejemplo.

Si una relación entre dos entidades incluye atributos descriptivos, la transformación de estas entidades a un diagrama de estructura de datos es un poco más complicada. Esto se debe a que una liga no puede contener valores de datos. En este caso es preciso crear un nuevo tipo de registro y establecer las ligas como se indica a continuación.

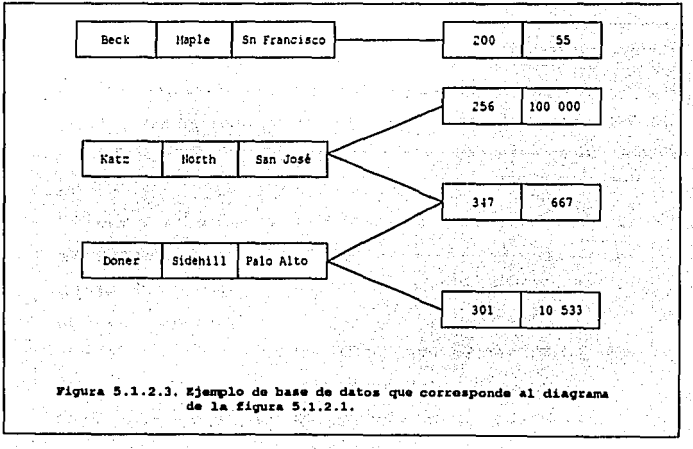


Figura 5.1.2.3. Ejemplo de base de datos que corresponde al diagrama de la figura 5.1.2.1.

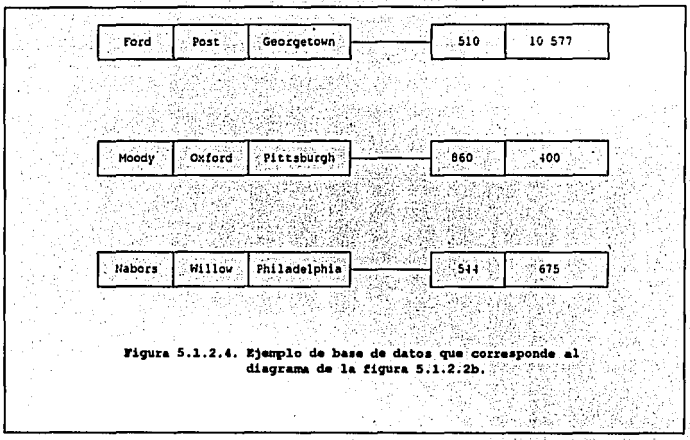


Figura 5.1.2.4. Ejemplo de base de datos que corresponde al diagrama de la figura 5.1.2.2b.

Supongamos que se agrega el atributo *fecha* a la relación *CthabCta* (de las dos entidades que se mencionaron al principio -*cuentahabiente* y *cuenta*-), para indicar la última vez que el *cuentahabiente* tuvo acceso a la *cuenta*. Al agregar un atributo a la relación anterior, para convertir esta relación con atributo a un diagrama de estructura de datos se necesita:

1) Sustituir las entidades *cuentahabiente* y *cuenta* por los tipos de registro cuentahabiente y cuenta, respectivamente.

2) Crear un tipo de registro nuevo, *fecha* con un solo campo que represente la fecha.

3) Crear las siguientes ligas muchos a uno:

* *CthabFecha* del tipo de registro *fecha* al tipo de registro *cuentahabiente*.

* *CtaFecha* del tipo de registro fecha al tipo de registro *cuenta*.

El diagrama de estructura de datos que resulta se muestra en la figura 5.1.2.5.

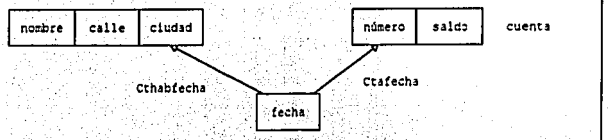


Figura 5.1.2.5. Diagrama de estructura de datos de la relación CthabCta con el atributo fecha.

En la figura 5.1.2.6 aparece una instancia de una base de datos que corresponde al esquema que se acaba de describir.

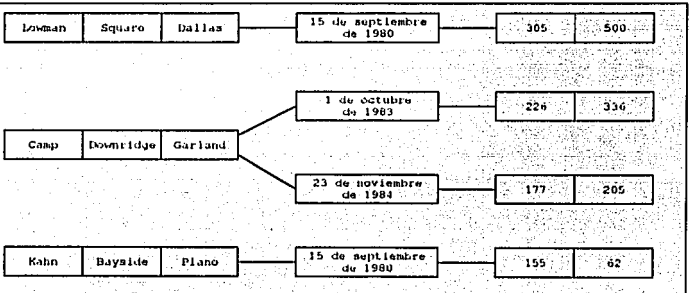


Figura 5.1.2.6. Ejemplo de la base de datos que corresponde al diagrama de la figura 5.1.2.5.

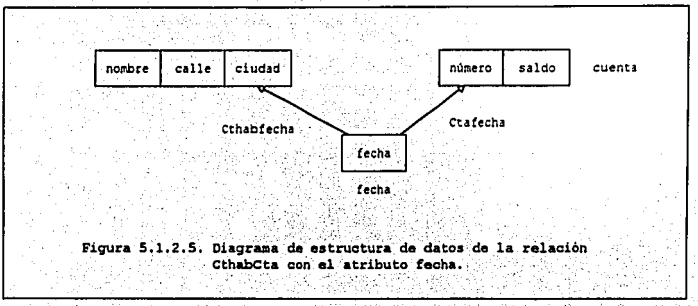


Figura 5.1.2.5. Diagrama de estructura de datos de la relación CthabCta con el atributo fecha.

En la figura 5.1.2.6 aparece una instancia de una base de datos que corresponde al esquema que se acaba de describir.

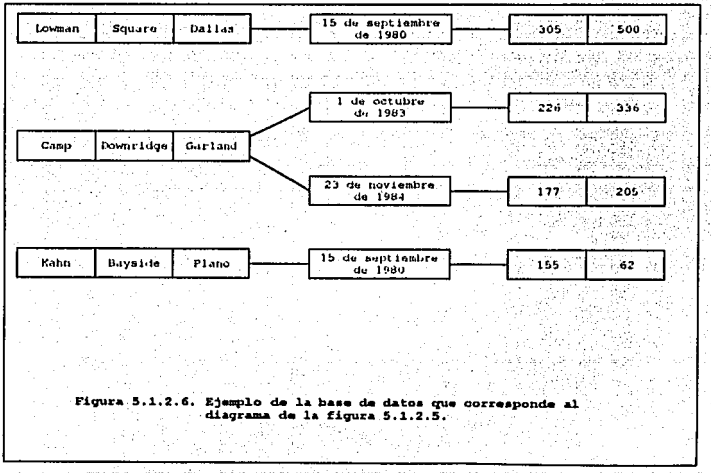


Figura 5.1.2.6. Ejemplo de la base de datos que corresponde al diagrama de la figura 5.1.2.5.

V.1.3. RELACIONES GENERALES

Considérense ahora tres conjuntos de entidades: cuenta, *cuentahabiente* y *sucursal*, relacionadas entre sí por medio de la relación general CCS sin atributos descriptivos.

Supóngase además que un *cuentahabiente* puede tener varias cuentas, cada una situada en una sucursal bancaria específica, y que una *cuenta* puede pertenecer a varios cuentahabientes distintos.

Puesto que una liga puede conectar solamente a dos tipos de registro diferentes, es preciso conectar a estos tres tipos de registro por medio de un nuevo tipo de registro ligado directamente a cada uno de estos tres registros, como se describe a continuación:

1) Sustituir los conjuntos de entidades *cuenta*, *cuentahabiente* y *sucursal* por los tipos de registro *cuenta*, *cuentahabiente* Y *sucursal*, respectivamente.

2) Crear un nuevo tipo de registro, *liga D* que puede carecer de campos o puede tener sólo uno que contenga un identificador único. Dicho identificar lo proporciona el sistema y no lo utiliza directamente el programa de aplicación. Este nuevo tipo de registro se denomina tipo de registro temporal (o *de liga* o de unión).

3) Crear las siguientes ligas del tipo muchos a uno:

* *CthabLiga* del tipo de registro *liga D* al tipo de registro *cuentahabiente.*
* *CtaLiga* del tipo de registro liga D al tipo de registro *cuenta.*
* SucLiga del tipo de registro *liga D* al tipo de registro *sucursal.*

El diagrama de estructura de datos que resulta se muestra en la figura 5.1.2.7.

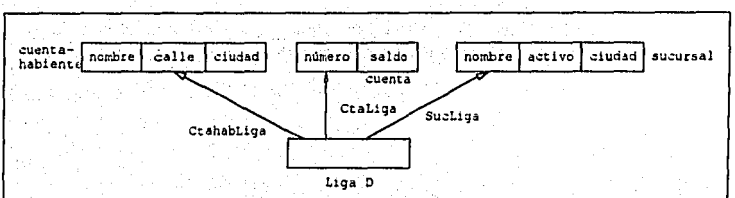


Figura 5.1.2.7. Diagrama de estructura de datos de las entidades cuenta, cuentahabiente y sucursal con su correspondiente Liga D, llamado registro temporal (o de liga o de unión).

En la figura 5.1.2.8 aparece una instancia de la base de datos que corresponde al esquema descrito.

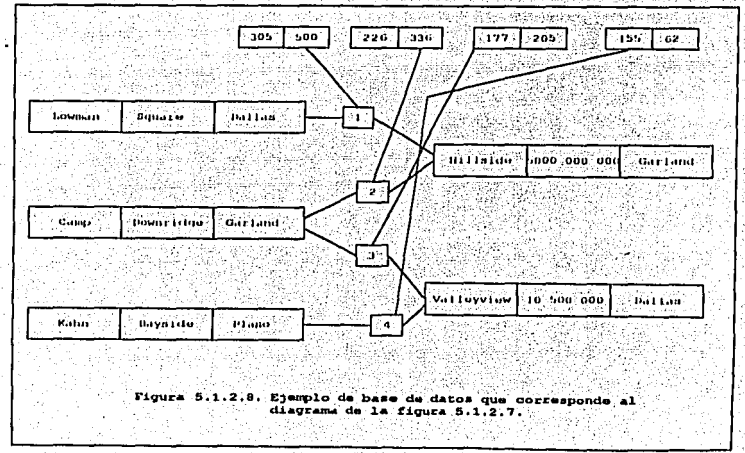


Figura 5.1.2.8. Ejemplo de base de datos que corresponde al diagrama de la figura 5.1.2.7.

CAPITULO VI

DISEÑO LOGICO DE BASES DE DATOS

VI.1. DISEÑO CONCEPTUAL

Para desarrollar una base de datos que satisfaga las necesidades de información actuales y futuras, se debe diseñar un *modelo conceptual*. Este modelo refleja las entidades y sus relaciones y esta basado en las necesidades de la organización de procesamiento de datos. Cuando se determinan las entidades y sus relaciones es necesario hacer un análisis de datos. Este análisis puede basarse en la información sobre datos, tanto para aplicaciones existentes como futuras.

La metodología para el diseño del *modelo conceptual* produce un diseño de base de datos independiente del enfoque de su realización, es decir, el *modelo conceptual* puede transformarse en un modelo relacional, jerárquico o de red.

La principal responsabilidad de un Administrador de Base de Datos (ABD) es diseñar un *modelo conceptual* de la empresa. Este modelo debe representar a las entidades de la empresa y las relaciones entre ellas.

El *modelo conceptual* no es la forma mediante la cual procesa la información un programador individual de aplicaciones. En lugar de esto, es una combinación de varios procedimientos usados para procesar los datos de varias aplicaciones.

El *modelo conceptual* es independiente de las aplicaciones individuales, independiente del sistema de manejo de base de datos, independiente del hardware usado para almacenar los datos e independiente del modelo físico de los datos en el medio de almacenamiento.

Cuando se diseña el *modelo conceptual*, se debe poner especial empeño en la estructura de los datos y de las relaciones entre los campos de datos de la empresa. Hasta este momento no debe haber relación con las fases de realización y operación de la base de datos.

La recolección de la información requiere de mucho tiempo, por lo que es necesario que la dirección tenga paciencia. El ABD debe iniciar un plan para completar está tarea. El ABD primero, debe usar un cuestionario o un vehículo similar para obtener de cada nivel administrativo (ejecutivo, funcional y operacional) una lista compuesta de los datos que necesita. Los diferentes niveles pueden procesar los datos o pueden almacenarlos. En segundo término, debe investigar los usos de oficina, operacionales y de procesamiento de datos de la empresa.

El cuestionario o el formulario para encuesta debe estar dirigido a todos los directores y supervisores de la empresa que necesiten datos o que los suministren a la base. Es muy importante que la encuesta la llene la dirección y no el personal subordinado. La encuesta debe reflejar las necesidades de la dirección. Debe ponerse a disposición de los suscriptores iniciales formularios para encuestas adicionales, de tal manera de que estos puedan aumentar la amplitud de la encuesta si la lista inicial no incluye a todos los usuarios.

El cuestionario debe requerir la siguiente información: nombre de la entidad, nombre del campo, descripción, atributos, fuente y sensibilidad (seguridad), valor o importancia de los datos y relaciones de los campos y entidades.

A continuación se trata una breve descripción de los principales puntos del cuestionario:

1) Nombre de las entidades y su descripción. Dar una lista del nombre y de cualquier sinónimo con los cuales se haga referencia a los campos de datos. Dar una descripción de lo que significa el nombre aunque el nombre aparentemente se defina o explique por sí mismo. Describir en forma general el uso o función de la entidad, el propósito al que sirve dentro de la unidad operativa o funcional de la unidad y todos los usuarios fuera de ésta.

2) Campos de datos. Para cada *pieza* de información en la unidad, proporcionar la siguiente información:

a) Nombre del campo y descripción. Enumerar los nombres, siglas y nemónicos. Dar una descripción completa del elemento.

b) Fuente de datos. Enumerar la(s) fuente(s) de origen según la unidad funcional y operativa.

c) Atributos del campo. Enumerar atributos numéricos, alfabéticos y textuales. Dar la unidad de medida conocida con los datos, tales como piezas, dólares y pies. Si existen limitantes a los rangos de valores aceptables del elemento, enumerarlos.

d) Uso del campo. Describir el uso.

e) Seguridad/Sensibilidad del campo. Enumerar todas las limitaciones asociadas con el nombre del campo. Estas limitaciones están generalmente relacionadas con restricciones al público, es decir, quién puede accesar, usar, leer y/o divulgar los datos.

f) Importancia/Valor. Mencionar la importancia de los datos.

g) Relaciones del campo. Enumerar las formas en las cuales éste campo se usa o relaciona con otros campos. Estos campos no necesitan estar limitados a la entidad específica que se describe.

3) Criterio de retención y almacenamiento. Describir la cantidad de tiempo y la forma en que los datos se guardan. Mencionar también si se conoce la razón o causa de la retención.

Junto con la realización de la encuesta, el ABD debe investigar los flujos de datos de la oficina, administrativos y del procesamiento de datos de la empresa. La investigación no debe verse como de vigilancia para asegurar la exactitud del cuestionario, más bien debe verse como un complemento al modelo de la empresa.

El siguiente proceso y con mucho el más importante que el ABD realiza es el análisis del depósito de los datos, que se construye a partir del cuestionario y la investigación del flujo de los datos. El análisis mismo se describe fácilmente, pero el proceso del análisis no

es rutinario o simplista. Ahora el ABD creará un mapa o diagrama de datos de las entidades y campos, mostrando las fuentes y entidades funcionales y recibiendo o usando funciones junto con sus entidades y campos funcionales, como se muestra en la figura 6.1.1.1.

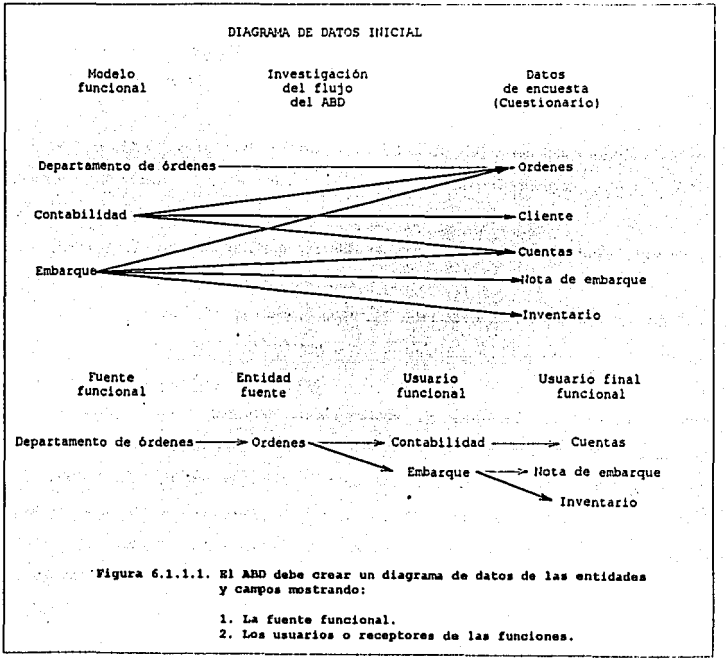


Figura 6.1.1.1. El ABD debe crear un diagrama de datos de las entidades y campos mostrando:

1. La fuente funcional.
2. Los usuarios o receptores de las funciones.

Se debe tener cuidado de no diseñar un modelo que satisfaga sólo las necesidades presentes. En los cuestionarios también pueden hacerse preguntas sobre posible campos

y/o relaciones a utilizar en un futuro, y considerando estos puntos, entonces continuar con el diseño del modelo conceptual. De esta manera el diseño resultante es fácilmente adaptable.

VI.2. EL MEDIO AMBIENTE EN LINEA

VI.2.1. SEGURIDAD

La información que se almacena en la base de datos debe protegerse contra el acceso no autorizado, la destrucción o alteración con fines indebidos, y la introducción accidental de inconsistencias.

Es más fácil evitar la pérdida accidental de consistencia de la información que impedir el acceso mal intencionado a la base de datos.

No es posible proteger de manera absoluta a la base de datos contra abusos ilícitos, pero puede hacerse que el costo sea tan alto que frustre casi todos los intentos de lograr acceso sin autorización a la base de datos.

Un usuario puede disponer de un modelo *personalizado* de la base de datos aprovechando el concepto *vistas*. Una vista puede ocultar los datos que el usuario no necesita conocer. La seguridad puede lograrse si se cuenta con un mecanismo que restrinja al usuario a su vista o vistas personales. Puede utilizarse una combinación de seguridad a nivel de relaciones y de vistas para limitar el acceso del usuario exclusivamente a los datos que necesite.

Un usuario puede contar con varios tipos de autorización sobre partes de la base de datos. La autorización es un modo de proteger al sistema de bases de datos contra el acceso no autorizado, o mal intencionado. Puede ser que un usuario al que se haya concedido algún tipo de autorización, la conceda a su vez a otros usuarios. No obstante, es

preciso cuidar mucho la forma como puede transmitirse la autorización entre usuarios para garantizar que se pueda revocar la misma en cualquier fecha posterior.

Es posible que todas las precauciones que tome el sistema de base de datos para autorizar el acceso no sean suficientes para proteger datos muy importantes. En tales casos se puede *cifrar* la información. Para leer información cifrada, es necesario que el lector conozca la forma de descifrar los datos.

VI.2.2. CONCURRENCIA

Uno de los conceptos mas importantes de los sistemas modernos es sin duda el de *multiprogramación o multitareas*. Si se ejecutan varias transacciones al mismo tiempo, puede compartirse el procesador entre ellas. Este esquema mejora la eficiencia total del sistema de cómputo, ya que se trabaja más en menos tiempo.

La idea de la multiprogramación es relativamente sencilla. Una transacción se ejecuta hasta un punto en el que debe de esperar (casi siempre a que se termine una operación de entrada o salida). En un sistema de cómputo multiprogramado, el procesador permanece ocioso.

Todo este tiempo de espera se desperdicia, pues el procesador no realiza un trabajo útil. En el paso de la *multiprogramación* se intenta aprovechar de manera productiva este tiempo. En un momento dado, se dispone de varias transacciones que deben ejecutarse.

Cuando es preciso que espere una transacción, el sistema libera al procesador de esa transacción y lo asigna a otra. Los beneficios que se obtienen de la *multiprogramación* son un mejor aprovechamiento del procesador y una productividad total de transacciones más alta. La productividad es la cantidad de trabajo que se realiza en un intervalo de tiempo dado.

Existen dos esquemas complementarios para implantar la *multiprogramación*:

1) Los sistemas no interactivos (Sistemas por lotes).

2) Los sistemas interactivos (Sistemas de tiempo compartido).

Las transacciones interactivas por lo general se componen de varias transacciones cortas, y el usuario espera el resultado de cada una. Por tanto, el tiempo de respuesta de las transacciones interactivas debe ser muy corto, a lo más del orden de segundos.

Puesto que las transacciones suelen ser cortas, sólo se requiere una pequeña cantidad de tiempo del procesador para cada usuario. Dado que el sistema cambia muy rápido de un usuario al siguiente, estos reciben la impresión de que tienen su propia computadora, siendo que en realidad una sola computadora se comparte entre varios usuarios.

Así, en un esquema de *multiprogramación* es posible ejecutar varias transacciones de manera concurrente. Como se verá, es necesario que el sistema controle la interacción entre las transacciones concurrentes para evitar que destruyan la consistencia de la base de datos. Este control se logra por medio de varios mecanismos a los que se denomina esquemas de control de concurrencia.

VI.2.3. RECUPERACION DE CAIDAS DEL SISTEMA

Un sistema de cómputo, como cualquier dispositivo mecánico o eléctrico, esta sujeto a fallas. Existen diversas causas de las fallas, por ejemplo:

1) Aterrizaje de cabezas en la unidad de disco. La información que recide en el disco se pierde.

2) Interrupción del suministro de energía. Se pierde la información almacenada en la memoria principal y los registros de uso general.

3) Errores de software. Los resultados generados pueden ser incorrectos, lo que resulta en respuestas erróneas a los usuarios y en que la base de datos entre en un estado de incongruencias.

Existen otras fuentes de error, por ejemplo, un incendio en el cuarto de computadoras, sabotaje, o hasta un agujero negro que pasara por el edificio en el que reside el sistema de cómputo. En todos estos casos se pierde información referente al sistema de base de datos.

Una parte integral de un sistema de base de datos es el subsistema de recuperación que se encarga de detectar las fallas y restaurar la base de datos al estado anterior a la ocurrencia de la falla.

VI.2.4. TIPOS DE FALLAS

Existen varios tipos de fallas que pueden presentarse en un sistema, y cada una de ellas debe manejarse de diferente manera. El tipo de falla más fácil de manejar es aquella que no resulta en pérdida de información en el sistema. Los más difíciles de resolver son los que resultan en perdidas de información.

Un esquema de recuperación se invoca a causa de diversos tipos de falla. En este apartado se consideran solamente los siguientes cuatro:

1) Errores lógicos. El programa no puede continuar su ejecución normal debido a condiciones internas, como pueden ser entradas inválidas, información no localizada, desborde o que se exceda el límite de los recursos.

2) Errores de sistema. El sistema ha entrado en un estado incorrecto por lo que el programa no puede continuar su ejecución normal. Sin embargo, es posible volver a ejecutar el programa.

3) Caída del sistema. El equipo funciona incorrectamente ocasionando la pérdida del almacenamiento volátil y el contenido del almacenamiento no volátil no se altera.

4) Falla de disco. Un bloque del disco pierde su contenido debido al aterrizaje de las cabezas o a fallas durante una operación de transferencia de información.

Algunas de las posibles acciones para recuperar información cuando se presenta alguna de las tres primeras fallas son:

a) Transacciones. Es una unidad de programa cuya ejecución conserva la consistencia de la base de datos. Para garantizar ésto, se requiere que las transacciones sean atómicas, es decir, que se ejecuten completamente todas las instrucciones implicadas en la transacción o que no se ejecute ninguna.

Nótese que el programador es el responsable de definir de manera correcta los diversos programas para que cada uno conserve la consistencia de la base de datos.

Una transacción es una unidad de programa que tiene acceso a varios datos y posiblemente los actualiza. La transacción lee una sola vez cada uno de estos datos y, en caso de que lo vaya a actualizar, escribe a lo más una sola vez cada dato.

Es posible que una transacción no termine siempre de ejecutarse correctamente. En este caso se dice que la transacción se *abortó*.

Por lo tanto una transacción abortada no debe afectar al estado de la base de datos.

Se dice que una transacción que completo correctamente su ejecución esta *cometida*.

Una *transacción cometida* que realiza actualizaciones transforma la base de datos a un nuevo estado consistente.

Se dice que una transacción está *cometida* sólo si llegó al estado cometido. De manera similar, se dice que una transacción se *abortó* sólo si ya entró en el estado abortado.

El diagrama de estados que corresponde a una transacción se ilustra en la figura 6.2.4.1 de el medio ambiente en línea (recuperación de caídas del sistema).

Una transacción se inicia en el estado *activo*, cuando llega a su última proposición entra en el estado parcialmente cometido. En este momento, la transacción ya término su ejecución, pero todavía es posible que tenga que abortarse, si no se ha hecho la salida a disco en sí. Así, una transacción cometida siempre podrá realizar sus escrituras externas, excepto en el caso de falla de disco.

Una transacción entra en el estado *fracasado* una vez que se determina que no puede procederse con la ejecución normal de la transacción. Una transacción así debe retroceder.

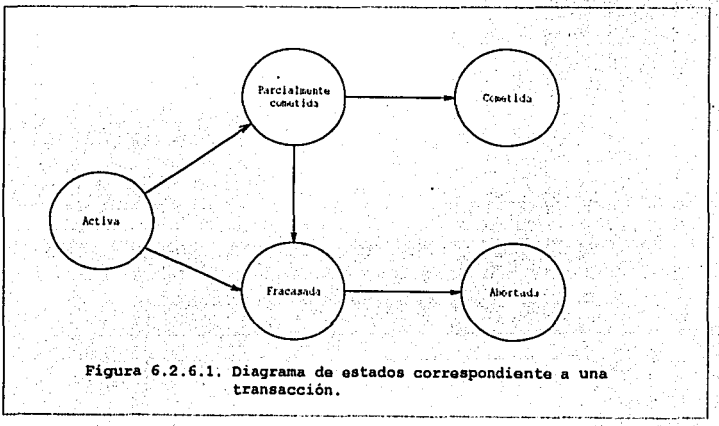


Figura 6.2.6.1. Diagrama de estados correspondiente a una transacción.

Cuando se haya efectuado el retroceso, la transacción estará en el estado abortado.

La transacción entra en estado cometido si se cometió parcialmente y se garantiza que nunca se va a abortar.

b) Bitácora incremental con actualizaciones diferidas. Durante la ejecución de una transacción se postergan todas las operaciones de grabación hasta que aquella quede

parcialmente cometida. Todas las actualizaciones del sistema se registran en la bitácora del sistema que debe mantenerse en almacenamiento estable. Una vez que la transacción está parcialmente cometida, la información de la bitácora asociada a la transacción se utiliza para ejecutar las grabaciones diferidas. Si el sistema se cae antes de que la transacción termine de ejecutarse, o si la transacción aborta, se hace caso omiso de la información de la bitácora. Empleando la bitácora, el sistema puede manejar cualquier falla que no resulte en pérdida de información de memoria no volátil. Cuando no ocurre ninguna falla, el esquema de recuperación consulta la bitácora y repite todas las transacciones cometidas.

c) Bitácora incremental con actualizaciones inmediatas. Todas las actualizaciones se aplican directamente a la base de datos, manteniendose en almacenamiento estable una bitácora incremental de todos los cambios que se hacen en el estado del sistema. Si se presenta una caída, la información de la bitácora se utiliza para restaurar el sistema a un estado consistente previo. Como en el caso anterior, la bitácora puede manejar cualquier falla que no resulte en pérdida de información de memoria no volátil.

d) Puntos de verificación. Cuando se presenta una falla del sistema, es necesario consultar la bitácora para determinar cuáles son las transacciones que es preciso repetir o anular. En principio, es necesario revisar toda la bitácora para determinar esto. Este enfoque tiene dos problemas principales:

* El proceso de búsqueda consume tiempo.
* La mayor parte de las transacciones que deben repetirse ya efectuaron sus actualizaciones de la base de datos, por lo que realmente no necesitan repetirse. Repetirlas no causa daño, pero provoca que la recuperación tarde más.

Para reducir este tiempo extra, se introducirá el concepto de puntos de verificación.

Durante la ejecución, el sistema mantiene la bitácora de la manera como se explicó anteriormente, pero además ejecuta periódicamente verificaciones, para lo que se requieren las siguientes acciones:

* Grabar en almacenamiento estable todos los registros de bitácora que están actualmente en memoria principal.

* Grabar en disco todos los bloques de buffer modificado.

* Grabar un registro de bitácora <checkpoint> en almacenamiento estable.

Con el mecanismo de puntos de verificación pueden refinarse los procedimientos de recuperación anteriores. Después de presentarse una falla, el procedimiento de recuperación examina la bitácora para determinar cuál fué la última transacción que comenzó a ejecutarse después del último punto de verificación. Dicha transacción puede localizarse si se examina la bitácora hacia atrás para encontrar el primer registro <checkpoint>.

La acción a realizar cuando ocurre una falla del cuarto tipo es:

Falla con pérdida de almacenamiento no volátil. Hasta ahora sólo se ha tomado en cuenta el caso en que una falla del sistema resulta en pérdida de información que reside en almacenamiento volátil, mientras que el contenido del almacenamiento no volátil permanece intacto. Aquí solamente se hablará de almacenamiento en disco, pero lo que diga puede aplicarse también a otros tipos de almacenamiento no volátil.

El esquema básico es vaciar periódicamente el contenido de la base de datos completa a memoria estable. Por ejemplo, podría vaciarse la base de datos en una o varias cintas magnéticas. Si ocurre una falla que resulte en la pérdida de uno o más bloques físicos de la base de datos, se utiliza el vaciado más reciente para restaurar la base de datos a un estado consistente previo. Una vez hecho esto, se utiliza la bitácora para transformar la base de datos al estado consistente más reciente.

VI.2.5. TIPOS DE ALMACENAMIENTO

Existen varios tipos de medios de almacenamiento que se distinguen por su velocidad relativa, su capacidad y su resistencia a las fallas.

1) Almacenamiento volátil. La información que reside en memoria volátil generalmente no sobrevive a las caídas del sistema. La memoria principal y la memoria caché son ejemplos de este tipo de almacenamiento.

2) Almacenamiento no volátil. La información que reside en memoria no volátil normalmente sobrevive a las caídas del sistema. Los discos y cintas magnéticas son ejemplos de éste tipo de almacenamiento. El disco se utiliza para almacenamiento en línea, mientras que la cinta se utiliza para información archivada. Los discos son más confiables que la memoria principal, pero menos confiables que la cinta magnética. Sin embargo, ambos están sujetos a fallas que pueden resultar en perdida de información.

3) Almacenamiento estable. La información que reside en almacenamiento estable nunca se pierde (aunque teóricamente no se puede garantizar su fiabilidad). Para implementar una aproximación de este tipo de almacenamiento es necesario repartir la información en varios medios de almacenamiento no volátil (generalmente disco) susceptibles a diferentes tipos de fallas, y actualizar la información de manera controlada.

VI.2.6. OPERACION POR PARTE DEL USUARIO

Uno de los objetivos primordiales de la base de datos es crear un ambiente para la recuperación de información y para almacenar información nueva en la base de datos.

Existen tres tipos diferentes de usuarios de un sistema de base de datos y se distinguen por el modo en que ellos esperan interactuar con el sistema.

1) Usuarios casuales. Son usuarios complejos que interactúan con el sistema sin escribir programas. En cambio, escriben sus consultas en un lenguaje de consulta de base de datos. Cada una de tales consultas se maneja a través de un procesador de consultas, cuya función es tomar una proposición en *lenguaje de manejo de datos* (DML) y descomponerla en instrucciones que pueda entender el manejador de bases de datos.

2) Usuarios ingénuos. Son usuarios poco complejos que interactúan con el sistema llamando alguno de los programas de aplicaciones permanentes escritos previamente.

3) Usuarios especializados. Son usuarios complejos que escriben aplicaciones para la base de datos que no embonan en el marco tradicional de procesamiento de datos. Entre dichas aplicaciones se cuentan los sistemas de diseño ayudado por computadora, los sistemas expertos, los sistemas de modelación ambiental, etc.

VI.3. ADMINISTRACION DE BASES DE DATOS

Una de las razones principales para contar con sistemas de manejo de bases de datos es tener un control centralizado tanto de los datos como de los programas que tienen acceso a ellos. La persona que tiene éste control centralizado sobre el sistema es el *administrador de base de datos* . Las funciones del administrador de la base de datos son entre otras:

1) Definición de esquemas. Es decir, la creación del esquema original de la base de datos. Esto se logra escribiendo una serie de definiciones que el compilador de DDL

traduce a un conjunto de tablas que se almacenan permamentemente en *el diccionario de datos*.

2) Definición de la estructura de almacenamiento y del método de acceso. Es decir, la creación de las estructuras de almacenamiento y métodos de acceso apropiados. Esto se lleva a cabo escribiendo una serie de definiciones que posteriormente son traducidas por el compilador del lenguaje de almacenamiento y definición de datos.

3) Modificaciones del esquema y la organización física. Ya sea la modificación del esquema de la base de datos o de la descripción de la organización física del almacenamiento. Estos cambios aunque son relativamente poco frecuentes, se logran escribiendo una serie de definiciones utilizadas, ya sea por el compilador de DDL o por el compilador del lenguaje de almacenamiento y definición de datos para generar modificaciones a las tablas internas apropiadas del sistema.

4) Concesión de autorización para el acceso a los datos. Es decir, conceder diferentes tipos de autorización para acceso a los datos a los distintos usuarios de la base de datos. Esto permite al administrador de la base de datos regular cuáles son las partes de la base de datos a la que van a tener acceso diversos usuarios.

5) Especificación de las limitaciones de integridad. Estas limitaciones se conservan en una estructura especial del sistema que consulta el manejador de bases de datos cada vez que se lleva a cabo una actualización en el sistema.

CAPITULO VII

DISEÑO FISICO DE BASES DE DATOS

VII.1. ESTRUCTURAS FISICAS

VII.1.1. FORMATOS DE LOS REGISTROS

VII.1.1.1. COMO ESPECIFICAR LOS FORMATOS DE LOS REGISTROS

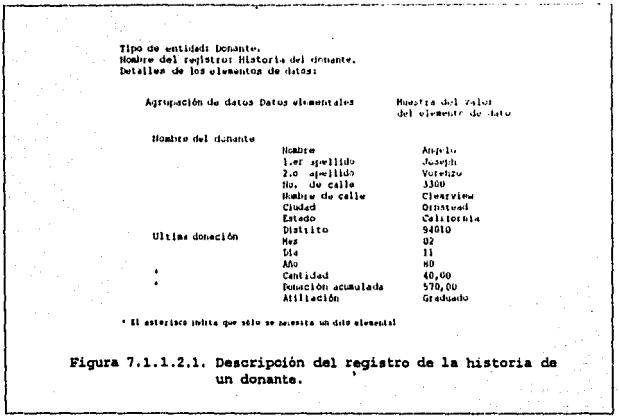
Hay diversos modos de especificar un registro, lo más importante es que se utilice el mismo método en todo el sistema. Todos los sistemas formales de procesamiento de datos tienen patrones que se utilizan para especificar los formatos del registro para registrar los utilizados en estos sistemas.

Cuando son considerados como partes de un registro, los elementos de datos se suelen denominar campos, grupos de caracteres que ocupan posiciones en el campo total de caracteres que configuran un registro.

VII.1.1.2. ESPECIFICACION DE LAS RELACIONES ENTRE LOS ELEMENTOS DE DATOS

Como la información frecuentemente consta de un grupo de elementos (tales como nombre y fecha) que constan de más elementos básicos (nombre, apellidos, mes, día, año), es útil disponer de un medio para describir el contenido de los registros que tienen grupos de datos y datos elementales relacionados.

Como ejemplo consideramos la figura 7.1.1.2.1 en donde se ilustra la descripción del registro de la historia de un donante mantenido por una institución de ayuda.



Tipo de entidad: Donante.
Nombre del registro: Historia del donante.
Detalles de los elementos de datos:

Agrupación de datos	Datos elementales	Muestra del valor del elemento de dato
Nombre del donante		
	Nombre	Angelo
	1.er apellido	Joseph
	2.o apellido	Vorehiz
	No. de calle	3300
	Nombre de calle	Clearview
	Ciudad	Ornstead
	Estado	California
	Distrito	94010
Ultima donación	Mes	02
	Día	11
	Año	80
*	Cantidad	40,00
*	Donación acumulada	570,00
	Afiliación	Graduado

* El asterisco indica que sólo se necesita un dato elemental

Figura 7.1.1.2.1. Descripción del registro de la historia de un donante.

Este método asegura que todo el que tiene derecho a conocer el contenido informativo de los registros de un sistema, puede determinar fácilmente el nombre y el contexto de cada elemento contenido en esos registros.

En la figura 7.1.1.2.2 se puede especificar el formato de un registro.

Adviértase que no se han anotado valores específicos, para el nombre del cliente. No sería práctico hacerlo, por la cantidad de nombres posibles que podrían darse. No siempre sucede que los valores específicos puedan ser incluídos, de antemano en la especificación del formato de un registro.

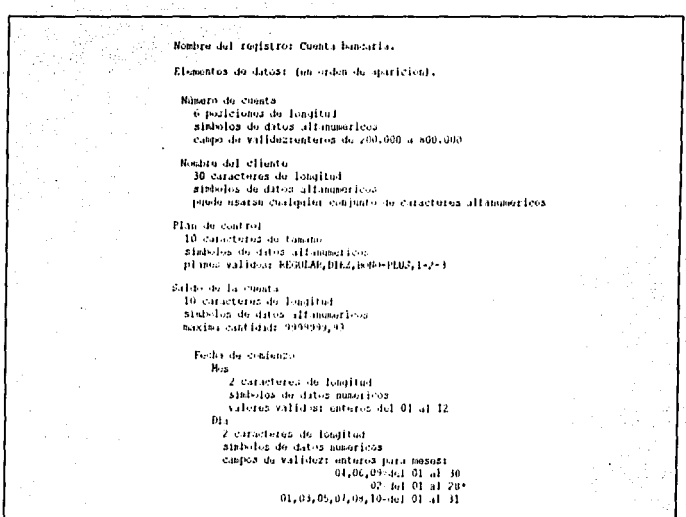


Figura 7.1.1.2.2. Ejemplo de una forma en que se puede especificar el formato de un registro.

Aunque las especificaciones anteriores puedan parecer muy detalladas son sólo una pequeña parte de los que se requieren para el formato de un registro completo. Puesto que escribir esta información en la forma anterior puede llevar una técnica adicional para ilustrar la forma del registro. Esta técnica es una ilustración gráfica de la forma del registro.

En la figura 7.1.1.2.3 se presenta una ilustración gráfica de la forma de un registro.

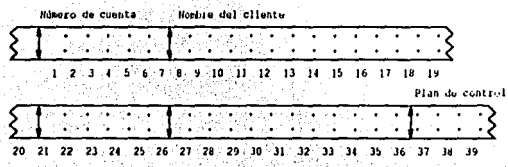


Figura 7.1.1.2.3. Ilustración gráfica de la forma de un registro.

Este ejemplo no pretende ser una presentación completa del formato del registro de una corriente, sino un ejemplo de cómo se especifican los formatos del registro y cómo se presentan en los sistemas de procesamiento de datos.

VII.1.2 ORGANIZACION DE ARCHIVOS

Un archivo está organizado lógicamente como una secuencia de registros. Estos registros se mapean a bloques del disco. Es preciso considerar las formas en que se mapean a bloques del disco. Es preciso considerar las formas en que puedan representarse los modelos lógicos de datos en términos de archivos.

Aunque los bloques son de tamaño fijo, el cual está determinado por las propiedades físicas del disco y por el sistema operativo, el tamaño de los registros varía.

Una forma de enfocar el mapeo de las bases de datos a los archivos es emplear varios archivos y almacenar en un archivo dado exclusivamente registros de una longitud fija. Una alternativa es estructurar los archivos de tal manera que puedan manejarse registros de varias longitudes diferentes. Los archivos de registros de longitud fija son más fáciles de implantar que los registros de longitud variable, por eso aquí trataremos en primer instancia los registros de longitud fija.

VII.1.2.1. REGISTROS DE LONGITUD FIJA

Consideremos como ejemplo un archivo de registros depósito para una base de datos bancaria. Cada registro del archivo se define de la siguiente manera:

```
Type Depósito = Record
          nombre - sucursal = char (20);
          número - cuenta   = integer;
          nombre - cliente  = char (20);
          saldo             = real;
            end
```

Si se supone que cada caracter ocupa un byte, un entero ocupa cuatro bytes y un número real ocupa ocho bytes, el registro depósito tendrá una longitud de 52 bytes. Un enfoque muy sencillo sería utilizar los primeros 52 bytes para el primer registro, los siguientes 52 para el segundo registro, etc. (figura 7.1.2.1.1). Pero para este enfoque tenemos dos problemas :

1.- Es difícil eliminar un registro de esta estructura. El espacio que ocupa el registro que se va a eliminar debe llenar con algún otro registro del archivo o bien debe ser posible marcar los registros eliminados de manera que el sistema los ignore.

2.- A menos que el tamaño del bloque sea un múltiplo de 52 (lo cual es poco probable) algunos registros quedarán olvidados entre dos bloques distintos, es decir, parte del registro quedará almacenada en un bloque y parte en otro. En cierto caso se requeriría tener acceso a dos bloques para leer o grabar el registro.

Cuando se elimina un registro podría moverse el registro que le seguía al lugar que ocupaba anteriormente el registro eliminado, etc, hasta que se hubieran movido hacia adelante todos los registros que seguían al registro eliminado (figura 7.1.2.1.2). Este método requiere mover una gran cantidad de registros.

registro 0	Perryridge	107	Hayes	400
registro 1	Round Hill	305	Turner	350
registro 2	Mianus	215	Smith	700
registro 3	Centro	101	Johnson	500
registro 4	Redwood	222	Lindsay	700
registro 5	Perryridge	201	Williams	900
registro 6	Brighton	217	Green	750
registro 7	Centro	110	Peterson	600
registro 8	Perryridge	218	Lyle	700

Figura 7.1.2.1.1. Archivo que contiene registros depósito.

Podría ser mejor simplemente mover el último registro del archivo al espacio que ocupaba el registro eliminado como se muestra en la figura 7.1.2.1.3.

registro 0	Perrydge	102	Hayes	400
registro 1	Round Hill	305	Turner	350
registro 3	Centro	101	Johnson	500
registro 4	Redwood	222	Lindsay	700
registro 5	Perryridge	201	Williams	900
registro 6	Brighton	217	Green	750
registro 7	Centro	110	Peterson	600
registro 8	Perryridge	218	Lyle	700

Figura 7.1.2.1.2. Archivo de la figura 7.1.2.1.1 después de eliminar el registro 2.

No es recomendable mover registros para ocupar el espacio que deja libre un registro eliminado ya que esto requiere tener acceso a bloques adicionales.

registro 0	Perrydge	102	Hayes	400
registro 1	Round Hill	305	Turner	350
registro 3	Perryridge	218	Lyle	700
registro 4	Centro	101	Johnson	500
registro 5	Redwood	222	Lindsay	700
registro 6	Perryridge	201	Williams	750
registro 7	Brighton	217	Green	750
registro 8	Centro	110	Peterson	700

Figura 7.1.2.1.3. Archivo de la figura 7.1.2.1.1 después de eliminar el registro 2.

Como las inserciones suelen ser más comúnes que las eliminaciones es aceptable dejar abierto el espacio que ocupaba el registro eliminado y esperar a una inserción posterior antes de volver a utilizar el espacio.

No basta simplemente marcar el registro eliminado ya que no sería fácil encontrar el espacio disponible cuando se va a realizar una inserción. Por lo tanto, es preciso introducir una estructura adicional.

Al principio del archivo el encabezado contiene información diversa acerca del archivo. Se almacena la dirección del primer registro cuyo contenido haya sido eliminado, éste primer registro se utiliza para almacenar la dirección del segundo registro disponible, etc. A estas direcciones se le pueden considerar como apuntadores, ya que *apunta* al lugar donde está almacenado un registro. La figura 7.1.2.1.4 muestra al archivo de la figura 7.1.2.1.1 después de que se eliminaron los registros 1, 4 y 6.

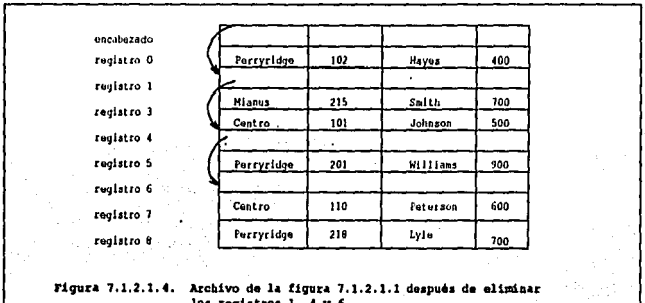


Figura 7.1.2.1.4. Archivo de la figura 7.1.2.1.1 después de eliminar los registros 1, 4 y 6.

Al insertarse un registro nuevo, se utiliza el registro al que apuntaba el encabezado; éste se modifica para que apunte al siguiente registro disponible. Si no hay espacio disponible, se agrega el registro al final del archivo.

La inserción y eliminación en archivos de registros de longitud fija es fácil de implantar porque el espacio que deja disponible un registro eliminado es exactamente el que se requiere para insertar un registro. Si se permiten registros de longitud variable en un

archivo la situación será diferente ya que es posible que un registro insertado no quepa en el espacio que dejo libre un registro eliminado o que llene solamente una parte de ese espacio.

VII.1.2.2. REGISTROS DE LONGITUD VARIABLE

Para implantar los registros de longitud variable de manera eficiente en un sistema de archivos, se utiliza uno o más registros de longitud fija para representar una longitud variable.

Existen dos técnicas para implantar archivos de registros de longitud variable empleando registros de longitud fija, las cuales son:

1) Espacio Reservado: Si los registros de longitud variable tienen una longitud máxima que nunca se va a exceder, pueden utilizarse registros de longitud fija de ese tamaño.

El espacio que no se utilice (para registros más cortos que la longitud máxima) se llena con un símbolo especial nulo o de *fin de registro.*

2) Apuntadores: El registro de longitud variable se representa por una lista de registros de longitud fija, encadenados por medio de apuntadores.

Si se opta por el método de espacio reservado para el ejemplo bancario, es preciso elegir un tamaño de registro máximo. La figura 7.1.2.2.1 muestra como se representa un máximo de tres cuentas por sucursal. Un registro de éste archivo es el tipo *lista-depósito*, con la salvedad de que el arreglo contiene exactamente tres elementos.

Las sucursales que tienen menos de tres cuentas (por ejemplo Round Hill) tiene registros con varios campos. Para representar, esto se utiliza el símbolo ⊥ en la figura

7.1.2.2.1. En la práctica, se emplea un valor determinado que no pueda ser nunca parte de la información real.

0	Perryridge	102	Hayes	400	201	Williams	900	218	Lyle	
1	Round Hill	305	Turner	350	⊥	⊥	⊥	⊥	⊥	⊥
2	Mianus	215	Smith	700	⊥	⊥	⊥	⊥	⊥	⊥
3	Centro	101	Johnson	500		Peterson	600	⊥	⊥	⊥
4	Redwood	222	Lindsay	700	⊥	⊥	⊥	⊥	⊥	⊥
5	Brighton	217	Green	750	⊥	⊥	⊥	⊥	⊥	⊥

Figura 7.1.2.2.1. Archivo de la figura 7.1.2.1.4 que utiliza el método de espacio reservado.

El método de espacio reservado es útil cuando gran parte de los registros son de longitud cercana al máximo, ya que de otra manera puede desperdiciarce una cantidad apreciable de espacio.

En el ejemplo bancario, puede suceder que algunas sucursales tengan muchas más cuentas que otras. Esto obliga a considerar el método de apuntadores, para representar el archivo empleando el método de apuntadores se agrega un campo que corresponda al apuntador, como se hizo en la figura 7.1.2.1.4 la estructura que resulta se muestra en la figura 7.1.2.2.2.

Una desventaja de la estructura de la figura 7.1.2.2.2 es que se desperdicia espacio en todos los registros, menos en el primero de la cadena.

Es preciso que el primer registro incluya el valor de *nombre - sucursal*, pero los subsecuentes no lo necesitan, sin embargo es necesario incluir el campo para *nombre - sucursal* en todos los registros pues de lo contrario no serían de longitud fija. Este espacio desperdiciado es considerable ya que en la práctica es de esperarse que las sucursales tengan un gran número de cuentas.

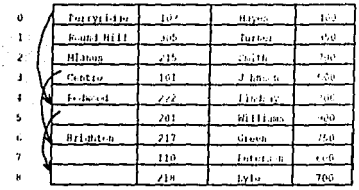


Figura 7.1.2.2.2. Archivo de la figura 7.1.2.1.1 después de eliminar los registros 1, 4 y 6.

Para resolver este problema, se permiten 2 tipos de bloque en el archivo :

1) Bloque ancla. Que contiene el primer registro de una cadena.

2) Bloque de desborde. Que contiene todos los registros que no son el primer registro de una cadena.

Así todos los registros dentro de un bloque tienen la misma longitud, aunque no todos los registros del archivo tienen la misma longitud. La figura 7.1.2.2.3 muestra esta estructura de archivo.

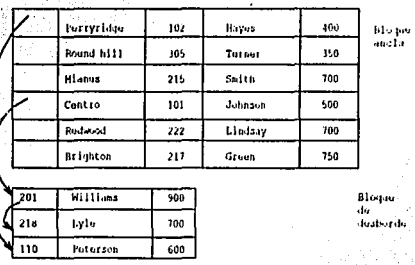


Figura 7.1.2.2.3. Organización de bloque ancla y bloque de desborde.

CAPITULO VIII

APLICACION DE LAS BASES DE DATOS

VIII.1. BASES DE DATOS DISTRIBUIDAS

En un sistema de base de datos distribuida, los datos se almacenan en varias computadoras, las computadoras de un sistema distribuído se comunican entre sí a través de diversos medios de comunicación, como pueden ser cables paralelos de alta velocidad o líneas telefónicas.

Los procesadores de un sistema distribuído pueden variar en cuanto a su tamaño y función. Pueden incluir microcomputadoras pequeñas, estaciones de trabajo, minicomputadoras y sistemas de cómputo grandes de aplicación general. Estos procesadores reciben diferentes nombres: localidades, nodos, etc, dependiendo del contexto en el que se mencionen. Aquí se usará normalmente el término localidad, para hacer hincapié en la distribución física de estos sistemas.

Un sistema distribuído de base de datos consiste en un conjunto de localidades, cada una de las cuales puede participar en la ejecución de transacciones que accesen datos de una o varias localidades. La diferencia principal entre un sistema de bases de datos centralizados y distribuídos es que, en los primeros, los datos residen en una sola localidad, mientras que en los últimos se encuentran en varias localidades.

VIII.1.1. ESTRUCTURA DE LAS BASES DE DATOS DISTRIBUIDAS

Un sistema de base de datos distribuído consiste de un conjunto de localidades, cada una de las cuales mantiene un sistema de base de datos local. Cada localidad puede procesar transacciones locales, es decir, aquellas que sólo procesan información que reside en esa localidad. Además, una localidad puede

participar en la ejecución de transacciones globales, es decir, aquellas que accesan información de varias localidades. La ejecución de transacciones globales requiere comunicación entre las localidades.

Las localidades del sistema pueden conectarse físicamente de diversas formas. Las distintas configuraciones se representan por medio de gráficas cuyos nodos corresponden a las localidades. Una arista del nodo A al nodo B corresponde a una conexión directa entre las dos localidades. En la figura 8.1.1.1 se ilustran algunas de las configuraciones más comunes. Las diferencias principales entre estas configuraciones son:

a) Costo de instalación. El costo de conectar físicamente las localidades del sistema.

b) Costo de comunicaciones. El costo en tiempo y dinero que implica enviar un mensaje de la localidad A a la B.

c) Confiabilidad. La frecuencia con que falla una línea de comunicación a una localidad.

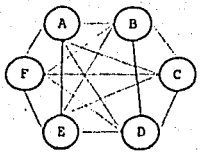
d) Disponibilidad. La posibilidad de accesar la información a pesar de fallas en algunas localidades o líneas de comunicación.

Como se verá, estas diferencias juegan un papel importante en la elección del mecanismo apropiado para manejar la distribución de los datos.

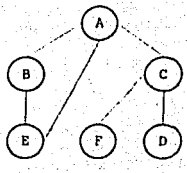
Las localidades de un sistema distribuido de base de datos pueden estar dispersas de manera física ya sea en una área geográficamente extensa o en una área reducida. Una red del primer tipo se denomina *red de larga distancia*, mientras que las últimas se conocen como *redes de área local*.

Puesto que las localidades de las redes de larga distancia están distribuidas en forma física en una área geográficamente extensa, es probable

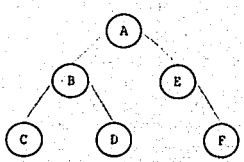
que las líneas de comunicación sean relativamente lentas y menos confiables en comparación con las redes de área local



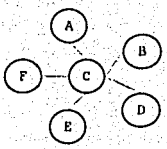
red totalmente conectada



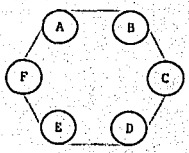
red parcialmente conectada



red con estructura de árbol



red de estrella



red anillo

Figura 8.1.1.1. Topología de las redes.

Las líneas de comunicación de larga distancia normales son las líneas telefónicas, conexiones de microondas y canales de satélite. Por otra parte, como todas las localidades de las redes de área local están próximas entre sí, las líneas de comunicación son de más alta velocidad y menor tasa de errores que sus contrapartes en las redes de larga distancia. Las conexiones más comúnes son cables dobles trenzados, coaxiales de banda base, coaxiales de banda ancha y de fibras ópticas.

Estos ejemplos se ilustran por medio de un sistema bancario que cuenta con cuatro sucursales situadas en cuatro ciudades diferentes.

Cada sucursal tiene su propia computadora con una base de datos que incluye todas las cuentas que tiene esa sucursal. Así pues, cada una de estas instalaciones es una localidad. También existe una localidad única que mantiene información acerca de todas las sucursales del banco.

Una transacción local es una transacción que accesa cuentas sólo en la localidad en que se inició. En cambio, una transacción global accesa cuentas de una localidad que no es en la que se inició, o cuentas de varias localidades distintas.

Por tanto, las condiciones para que se pueda tener una base de datos distribuida son:

1) Cada localidad debe estar consciente de la existencia de las demás.

2) Cada localidad permite ejecutar transacciones tanto locales como globales.

VIII.1.2. CONCESIONES AL DISTRIBUIR LA BASE DE DATOS

Existen varias razones para construir sistemas distribuidos de bases de datos: compartir la información, mejorar la confiabilidad y la disponibilidad, y agilizar el procesamiento de las consultas. Sin embargo, estas ventajas vienen acompañadas de varias desventajas una de las cuales son mayores costos de desarrollo de software, mayor posibilidad de errores y aumento en el costo extra de procesamiento.

VIII.1.3. VENTAJAS DE LA DISTRIBUCION DE LA INFORMACION

La principal ventaja de los sistemas distribuídos de base de datos es la capacidad para compartir y accesar la información de manera confiable y eficiente. Entre las principales ventajas tenemos:

1) Utilización compartida de los datos y distribución del control. Si varias localidades diferentes estan conectadas entre sí, entonces un usuario de una localidad puede accesar datos disponibles en otra localidad. Si no se contara con esta facilidad, un usuario que quisiera transferir fondos de una sucursal a otra, tendría que recurrir a un mecanismo externo para realizar la transacción. Este mecanismo externo sería, de hecho, una base de datos única centralizada.

La ventaja principal de compartir los datos por medio de la distribución de la información es que cada localidad puede controlar hasta cierto punto los datos almacenados localmente. En un sistema centralizado, el administrador de base de datos de la localidad central controla la base de datos. En un sistema distribuído, existe un administrador global de la base de datos que se encarga de

todo el sistema. Parte de estas responsabilidades se delegan al administrador de base de datos de cada localidad. Dependiendo del diseño del sistema distribuído, cada administrador local podrá tener un grado de autonomía diferente que se conoce como *autonomía local*. La posibilidad de contar con autonomía local es en muchos casos una ventaja importante de las bases de datos distribuídas.

2) Confiabilidad y disponibilidad. Si se presenta una falla en una localidad distribuída, es posible que las demás localidades puedan seguir trabajando. En particular, si los datos se repiten en varias localidades, una transacción que requiera de un dato específico puede encontrarse en más de una localidad. Así, la falla de una localidad no implica necesariamente la desactivación del sistema.

El sistema debe detectar cuando falla una localidad y tomar las providencias necesarias para recuperarse de la falla. El sistema no debe seguir utilizando la localidad que falló. Por último, cuando se recupere o repare ésta localidad, debe contarse con mecanismos para reintegrarla al sistema con un mínimo de complicaciones.

Aunque la recuperación de fallas es más compleja en los sistemas distribuídos que en los centralizados, la capacidad que tiene el sistema para seguir trabajando a pesar de la falla de una localidad resulta en una mayor disponibilidad. La disponibilidad es fundamental para los sistemas de base de datos que se utilizan en aplicaciones de tiempo real. Por ejemplo, si una línea aérea no puede tener acceso a la información, es posible que pierda clientes a favor de la competencia.

3) Agilización del procesamiento de consultas. Si una consulta comprende datos de varias localidades, puede ser posible dividir la consulta en varias subconsultas que se ejecutan en paralelo en distintas localidades. Este

cálculo en paralelo permite procesar con rapidéz la consulta del usuario. En los casos en que hay repetición de los datos, el sistema puede pasar la consulta a las localidades con carga más ligera.

VIII.1.4. DESVENTAJAS DE LA DISTRIBUCION DE LOS DATOS

La desventaja principal de los sistemas distribuídos de bases de datos es la mayor complejidad que se requiere para garantizar una condición adecuada entre localidades. El aumento en la complejidad se refleja en:

1) Costo de desarrollo de software. Es más difícil estructurar un sistema de base de datos distribuído y por tanto su costo es mayor.

2) Mayor posibilidad de errores. Puesto que las localidades del sistema distribuído operan en paralelo, es más difícil garantizar que los algoritmos sean correctos.

Existe la posibilidad de errores extremadamente sútiles. El arte para construír algoritmos para sistemas distribuídos sigue siendo un campo de investigación activo e importante.

3) Mayor tiempo extra de procesamiento. El intercambio de mensajes y los cálculos adicionales que se requieren para coordinar las localidades son una forma de tiempo extra que no existe en los sistemas centralizados.

Al escoger el diseño de una base de datos, el diseñador debe equilibrar las ventajas y las desventajas de la distribución de los datos.

VIII.1.5. DISEÑO DE LAS BASES DE DATOS DISTRIBUIDAS

Los principios del diseño de bases de datos que se explicaron en capítulos anteriores también se aplican a las bases de datos distribuídas. Esta sección se dedicará a los factores de diseño que se aplican específicamente a las bases de datos distribuídas.

Considérese una relación que se va a almacenar en la base de datos. Hay varios factores que deben tomarse en cuenta al almacenar esta relación en la base de datos distribuída. Tres de ellas son:

1) Repetición. El sistema mantiene varias copias idénticas de la relación. Cada copia se almacena en una localidad diferente, lo que resulta en repetición de información. La alternativa a la repetición es almacenar una sola copia de la relación.

Si la relación *r* esta repetida, se almacena una copia en dos o mas localidades. En el caso extremo se tiene *repetición total*, en la que se almacena una copia de la relación en cada una de las localidades del sistema.

La repetición tiene varias ventajas y desventajas:

a) Disponibilidad. Si falla una de las localidades que contiene a la relación, puede disponerse de ésta en otra localidad. Así, el sistema puede continuar procesando consultas que impliquen a *r* a pesar de haber fallado una localidad.

b) Mayor paralelismo. En el caso en que la mayor parte de los accesos a la relación *r* resulten sólo en la lectura de la relación, varias localidades podrán procesar consultas que involucren a *r* en paralelo. Mientras más copias de *r* existan, mayor será la probabilidad de que los datos requeridos se encuentren en

la localidad donde se está ejecutando la transacción. Por tanto, la repetición de los datos reduce al mínimo el movimiento de información entre las localidades.

c) Mayor tiempo extra para las actualizaciones. El sistema debe asegurarse de que todas las copias de la relación *r* sean consistentes, pues de otra manera pueden hacerse cálculos erróneos. Esto implica que cada vez que se actualice *r*, la actualización debe propagarse a todas las localidades que contengan copias, lo que resulta en un mayor tiempo extra.

2) Fragmentación. La relación se divide en varios fragmentos. Cada fragmento se almacena en una localidad diferente.

Si la relación r está fragmentada, *r* se dividirá en varios fragmentos r1, r2,., rn. Estos fragmentos contienen información suficiente para reconstruir la relación *r* original. Como se verá esta reconstrucción puede llevarse a cabo ya sea aplicando la operación de unión o un tipo especial de operación de unión sobre los diversos fragmentos. Existen dos esquemas diferentes para fragmentar una relación: fragmentación horizontal y fragmentación vertical. La fragmentación horizontal divide a la relación asignando cada tupla de *r* a uno o más fragmentos. La fragmentación vertical divide a la relación descomponiendo el esquema R de la relación *r* de una manera especial. Estos dos esquemas pueden aplicarse en forma sucesiva a la misma relación, lo que resulta en varios fragmentos diferentes.

3) Repetición y fragmentación. Esta es una combinación de los dos incisos antes mencionados.

La relación se divide en varios fragmentos. El sistema mantiene varias copias idénticas de cada uno de los fragmentos.

La técnica que se acaba de describir para repetir y fragmentar la información puede aplicarse de manera sucesiva a la misma relación. Es decir, un fragmento puede repetirse, las copias pueden fragmentarse, etc.

VIII.2. BASES DE DATOS EN MICROCOMPUTADORAS

La tecnología no ha hecho más que seguir el ritmo de la creciente demanda de información. Su disponibilidad ha creado, de por sí, una creciente demanda para su uso. Información útil que era considerada demasiado costosa o difícil de obtener con las viejas técnicas, está ahora inmediatamente disponible a un coste razonable. El resultado es una creciente demanda de tal información. Los computadores, particularmente, han capacitado a la gente para reunir y almacenar grandes cantidades de datos de un modo tal que los datos pueden ser recuperados de un modo rápido, cómodo y económico. Los datos almacenados pueden ser manipulados y asociados en una ilimitada variedad de modos para el análisis y la información.

Las mejoras en la eficacia de los equipos computadores, evaluados por factores como el coste, la velocidad, la capacidad de memoria, fiabilidad y tamaño, se han alcanzado a un ritmo tan asombroso que se le ha comparado con pasar desde el aeroplano de los hermanos Wright hasta la lanzadera espacial en el plazo de una década.

La característica fundamental de los sistemas de base de datos para microcomputadoras, es su sencillez. La capacidad limitada de las computadoras personales restringe tanto el tamaño de la base de datos como el grado de complejidad del sistema, aunque casi todos los sistemas de base de datos se

diseñan pensando en la facilidad de uso, la importancia de éste factor en el mercado de las computadoras personales es extraordinaria, ya que los usuarios no pueden contar con la ayuda de un administrador de base de datos experimentado. Cada uno de los usuarios de un sistema de base de datos de microcomputadora funge como administrador de base de datos.

A continuación se comparan las características comúnes de los sistemas de base de datos para microcomputadora con los de los sistemas grandes:

- **Modelo de datos.** Dado que los sistemas de base de datos para microcomputadoras son relativamente nuevos, casi todos están basados en el modelo relacional. Algunos de esos sistemas deben considerarse más bien tabulares, ya que aunque utilizan tablas, son demasiado primitivos para llamarse relacionales.
- **Lenguaje de consultas.** Es posible que aún los lenguajes de más alto nivel sean demasiado complejos para el usuario casual de un sistema de base de datos para microcomputadoras. Muchos de los lenguajes se basan en una interfaz de forma, con la cual el usuario puede interactuar con el sistema llenando una forma.
- **Implantación física.** Para los implantadores de un sistema, uno de los factores importantes es el espacio que ocupa el código objeto de un sistema de base de datos para microcomputadora. Si se reduce el espacio requerido, es posible utilizar el sistema en máquinas que cuenten con menos memoria principal. Esto puede tener una influencia determinante sobre el mercado potencial. Por esta razón son pocos los sistemas que utilizan técnicas de manejo de memoria y de indexación complejos. Por lo general se elige un solo tipo de índice, y la optimización de consultas, cuando se lleva a cabo es rudimentaria.

* **Recuperación.** Muchos sistemas no cuentan con subsistemas de recuperación. El usuario tiene la responsabilidad de sacar copias de respaldo de sus datos en forma regular.

* **Concurrencia.** No se requiere control de concurrencia para computadoras personales de un sólo usuario.

La distinción entre los sistemas de base de datos para microcomputadora y los sistemas más grandes se hace menos clara. Los primeros sistemas de base de datos para microcomputadoras no eran mucho más que interfaces para lograr acceso a un solo archivo de registros de longitud fija. Al crecer la capacidad de las computadoras personales, ha crecido también la complejidad de los sistemas de base de datos para microcomputadoras. De hecho, han comenzado a aparecer versiones de algunos de los sistemas de base de datos de gran escala en las computadoras personales de mayor tamaño. Un ejemplo de este tipo de sistemas es el SQL/RT de la computadora personal IBM RT.

VIII.3. BASES DE DATOS ORIENTADA A OBJETOS (ODB)

Las bases de datos distribuidas y el enfoque orientado a objetos son el punto central de las plataformas actuales y a pesar de los diferentes productos que han surgido en el mundo plataformas como CORBA y Open Doc y otras bases de datos orientadas a objetos ninguna es dominante o está marcando la pauta a seguir.

La combinación de bases de datos orientadas a objetos (ODB) y las aplicaciones orientadas a objetos dan como resultado dos categorías discernibles de métodos de clases:

1) Aquellas que especifican la aplicación (métodos de aplicación).

2) Aquellas que relacionan la administración y la manipulación de datos (métodos de base de datos orientadas a objetos.

Por ejemplo:

El cálculo de una fórmula química es una aplicación específica, mientras un objetos teniendo un valor particular es un valor específico de la base de datos.

Es muy importante no confundir estos métodos con el método de ejecución.

Algunos productos actualmente ejecutan sus propios métodos de bases de datos orientadas a objetos como parte de la aplicación y otros ejecutan métodos de aplicación como parte de un procesos servidor orientado a objetos.

VIII.3.1. INSTALACION DE BASES DE DATOS ORIENTADAS A OBJETOS

Una etiqueta es colocada en la posición dentro de la estructura de la instancia de datos en donde el código de aplicación espera encontrarla. Estas etiquetas son identificadores únicos de objetos (OIDs), nombres de clases,de números e identificadoras de cada versión de esquemas. Obviamente debe haber un sistema central para prevenir la redundancia en los OIDs y forzar el uso correcto de las clases.

Dado un número o nombre de clase, el cliente puede hacer referencia al método de clase mediante un mecanismo que carga un programa referido a que nombre ó número.

Por ejemplo, las instancias de bases de datos distribuidas incluyen un header con un identificador de clase que se puedan disparar métodos de

llamados indirectos atravéz de un vector de métodos direccionados. Es así que la instancia de clase es reconocida como una libreria ligada a la base de datos o el código de una libreria.

Si el código está instalado localmente pero las instancias son surtidas por un objeto de la base de datos, la base de datos es pasiva Pero si ambas instancias y el código son proporcionadas por el objeto de la base de datos, entonces la base de datos es activa.

Un programa de aplicación no necesita accesar instancias de clases remotas o código remoto, es suficiente que la aplicación pueda identificar un objeto o una clase de objeto en cualquier localidad ya que la la aplicación puede enviar un mensaje al server remoto solicitando que los métodos de clases sean cargados.

La base de datos orientada a objetos ITASCA carga ambos recursos y métodos en donde el método de ejecución toma lugar. Esto permite un activa distribución del cargo en la computadora entre los nodos servidores.

El sistema provee una compilación automática ligada al código fuente.

Hay cuatro enfoques para administrar en bases de datos distribuidas orientadas a objetos :

1) El código de una libreria local liga la ejecución o al programa servidor ODB.

2) La base de datos orientada a objetos carga información referenciada al identificar el método y la libreria ligada en donde ésta se encuentra. Esta información información pasa aun programa convencional. Ocasionalmente el método es cargado y ejecutado en un nodo remoto de la red.

3) El ODB guarda el código en la computadora intermediaria y este es deliberado al interprete del compilador y a su vez guardado en el servidor de la aplicación.

4) La base de datos orientada a objetos carga un código nativo (binario) en la plataforma de ejecución e implica la carga de un programa extendido, es decir emite segmentos de código a la aplicación.

Una libreria local generalmente un DDL (Dinamic Link Library) , puede ser estáticamente ligada, puede sólo aparecer como local al proceso que la carga , o puede ser un directorio remoto que ha sido instalado localmente.

CAPITULO IX

DISEÑO Y APLICACION REAL DE UN SISTEMA DE BASE DE DATOS

Para el desarrollo de nuestro sistema el cual consiste en la *Programación de Vuelos y Reservaciones de una línea aérea* diseñamos las siguientes tres herramientas:

1) Manejo de menús.

2) Captura de información.

3) Manejo de información.

La herramienta de manejo de menús nos permite elegir el proceso que se desea llevar a cabo dentro del sistema. Para la elección de un proceso determinado se programaron una serie de teclas que nos permiten navegar en los menús y submenús que incluye el sistema de aplicación.

La herramienta de captura de información llamada *algoritmo de captura*, tiene la función, valga la redundancia, de capturar y validar la información. La importancia de esta herramienta radica en que permite verificar la información capturada antes de ser grabada, pues presta la facilidad de navegar a través de la máscara de captura mediante las teclas de cursor arriba y abajo.

Por último, la herramienta de manejo de información llamada *algoritmo indexador* es la parte más importante dentro del sistema, ya que por medio de ésta se tiene acceso a la información almacenada en las diferentes bases de datos.

Consideramos que para búsqueda, eliminación, inserción y recuperación de información los algoritmos de *árbol binario* y *hash* son los más eficientes. A continuación se da una breve explicación de ellos.

El algoritmo *hash* realiza un acceso directo al registro buscado calculando la dirección del mismo por medio de diferentes métodos, el diseñador es responsable del método a utilizar.

A continuación se mencionan algunas técnicas para calcular la dirección de un registro (en todos los casos se deben convertir los caracteres a su valor ASCII o bien, asignarles un cierto número):

a) Truncamiento. Sólo se consideran determinadas posiciones de la llave y el resultado es la posición del registro.

b) Doblamiento. Se divide la llave en varias partes y se suman o multiplican para obtener el índice.

c) Aritmética modular. Se basa en la fórmula:

dirección = módulo(llave/número primo fijo)

El número primo fijo debe aproximarse al rango de llaves que se espera será el máximo a almacenar en la base de datos.

La desventaja que este método presenta es que se pueden dar colisiones, y cuando este sucede se debe emplear un método alterno, lo cual implica que el cálculo de la dirección sea más tardado y más complejo.

Ahora bien, el algoritmo de árbol binario se basa en la comparación de la llave con un nodo determinado del árbol, realizando las siguientes acciones: 1) Continuación del recorrido si el resultado de la comparación es mayor o menor, 2) Inserción de un nuevo nodo u 3) Obtención del apuntador del registro.

Como se puede ver en este algoritmo nunca se presentan colisiones, tal vez la desventaja que se puede presentar es que sea un poco más lento debido al crecimiento del archivo índice generado.

Habiendo analizado lo anterior, basamos el algoritmo indexador en el algoritmo de *árbol binario*, ya que a nuestro criterio es el más viable para la aplicación.

IX.1. HERRAMIENTAS DEL SISTEMA DE BASE DE DATOS

IX.1.1. DESCRIPCION Y FUNCIONAMIENTO DEL ALGORITMO DE CAPTURA

Este es un procedimiento general de captura que tiene la función de verificar cada una de las teclas pulsadas y ejecutar el proceso que se le ha asignado. También realiza validaciones del tipo de campo que se está capturando, y cuando es necesario valida todos y cada uno de los caracteres del campo (por ejemplo en campos de tipo facha y hora); cuando ocurre algún error de tipo o de caracter en la captura, presenta un mensaje indicando que se cometió un error y cuál debe ser el tipo de caracter o el único válido a capturar. Además sólo permite la captura del número de caracteres especificado, haciendo un salto automático al siguiente campo cuando se ha llegado al límite de caracteres a capturar.

Una lista de las teclas programadas se muestra a continuación:

a) BACKSPACE.

b) RETURN.

c) ESCAPE.

d) F2.

e) F3.

f) FLECHA ARRIBA.

g) FLECHA ABAJO.

h) FLECHA IZQUIERDA.

i) FLECHA DERECHA.

j) INSERT.

k) DELETE.

Lo anterior ha sido una descripción general del funcionamiento del algoritmo, a continuación se dará una explicación más amplia de la manera en que éste funciona.

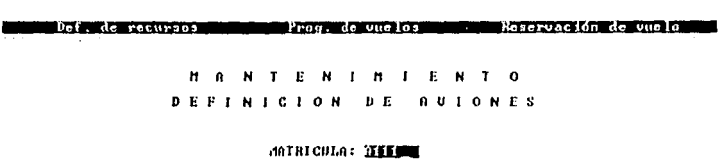
Para que el algoritmo funcione correctamente, se deben definir antes las características de cada uno de los campos que van a componer un registro y mantener las mismas en memoria hasta que no se le indique a la computadora que aborte la captura mediante la tecla ESCAPE, dichas características son las siguientes: 1) El número del campo, 2) Cuantos campos componen el registro, 3) El título del campo, 4) La posición en la coordenada X en donde se imprimirá el título del campo, 5) La posición en la coordenada Y de donde se imprimirá el título del campo, 6) La posición en la coordenada Y en donde se comenzará a capturar el campo, 7) La longitud en caracteres del campo y 8) El tipo del campo (Numérico = n, Alfanumérico = a, De fecha = f y De hora = h).

Ahora bien, mientras no se llegue al límite de la longitud definida para ese campo, el procedimiento verifica que tecla se ha pulsado y entonces ejecuta la función definida para esa tecla. Cuando se llega al límite de la longitud definida para ese campo salta al siguiente, si el campo capturado es el último, salta al primero de los campos definidos para ese registro.

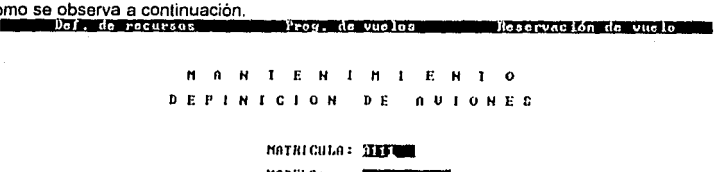
Anteriormente se dió una lista de las teclas programadas en este procedimiento, en seguida se explica detalladamente la función que se realiza cuando se oprime alguna de ellas:

FLECHA IZQUIERDA. Si se trata de un campo numérico o alfanumérico el que se está capturando, provoca que el cursor se posicione un espacio hacia la izquierda de donde se encuentra parado. Pero si se trata de un campo de tipo fecha u hora, se posiciona uno o dos espacios a la izquierda, dependiendo del lugar en el que el cursor se encuentre situado en ese momento.

En la figura siguiente se aprecia cómo después de capturar la MATRICULA, el cursor aparece delante del último caracter introducido.

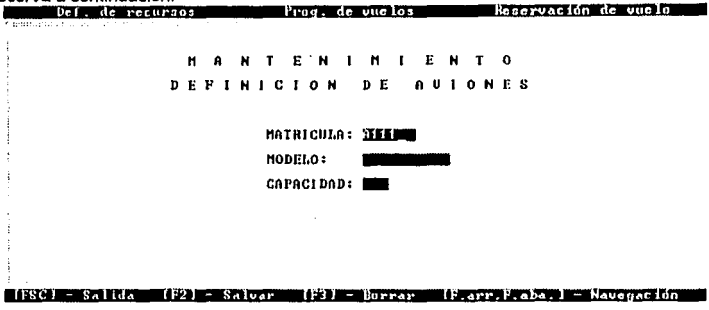


Después de haber pulsado la tecla de flecha izquierda el cursor salta una posición como se observa a continuación.



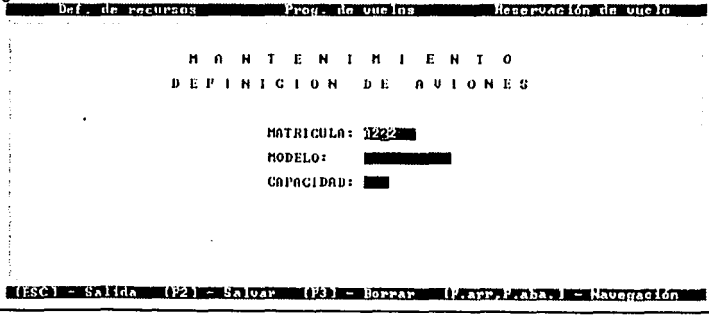
FLECHA DERECHA. Si se trata de un campo numérico o alfanumérico el que se está capturando, provoca que el cursor se posicione un espacio hacia la derecha de donde se encuentra situado. Pero si se trata de un campo de tipo fecha u hora, se posiciona uno o dos espacios a la derecha, dependiendo del lugar en el que el cursor se encuentre situado en ese momento.

De la última figura al presionar flecha derecha el cursor salta una posición como se observa a continuación.

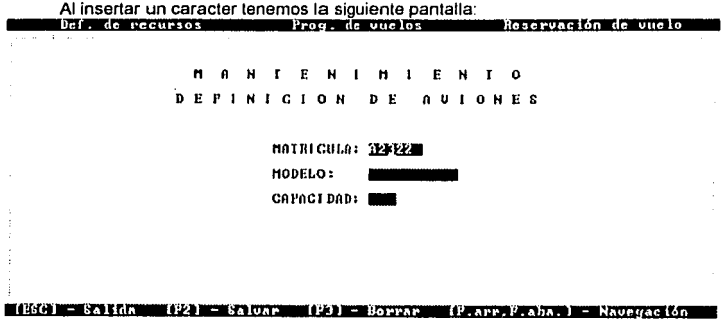


INSERT. Si se presiona esta tecla estando en el modo de sobreescritura, provoca que el procedimiento pase al modo de inserción. Pero si se está en el modo de inserción provocará que se pase al modo de sobreescritura.

En la siguiente figura, el sistema estaba en modo de sobreescritura y al presionar la tecla INSERT se cambió al modo de inserción. Cuando ocurre lo anterior el sistema lo indica agrandando el cursor.



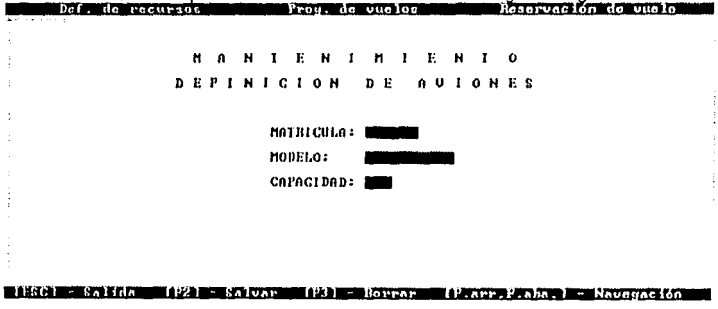
Al insertar un caracter tenemos la siguiente pantalla:



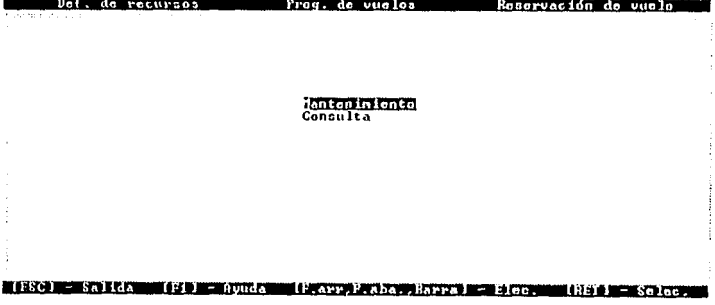
ESCAPE. Aborta en cualquier momento el proceso de captura del registro y restaura la pantalla anterior.

Observemos la siguiente figura:

Al seleccionar la opción de mantenimiento obtenemos la siguiente figura:

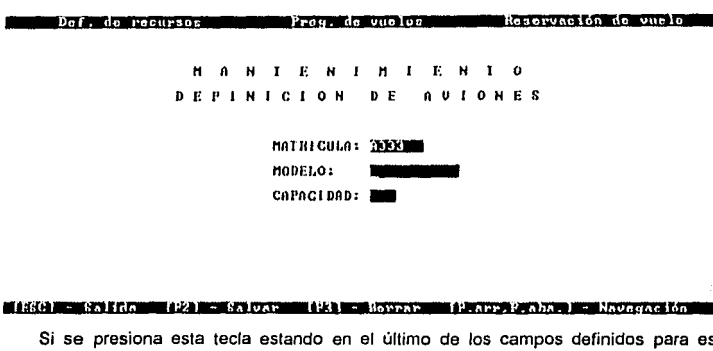


De manera que al presionar la tecla de ESC regresamos a la pantalla anterior:



RETURN. Da terminación a la captura de un campo y el procedimiento se prepara a recibir el siguiente campo.

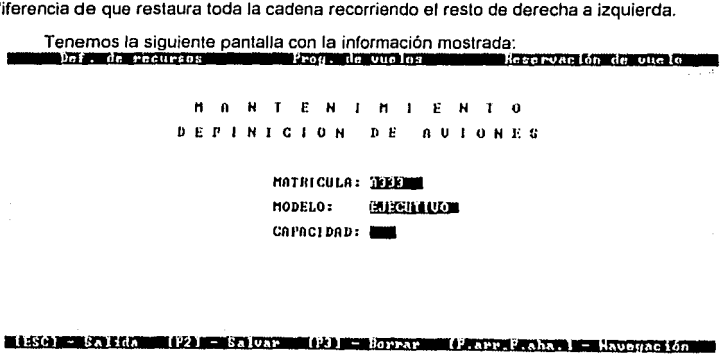
Terminamos de capturar el primer campo y al presionar la tecla de RETURN pasamos al siguiente como vemos a continuación.



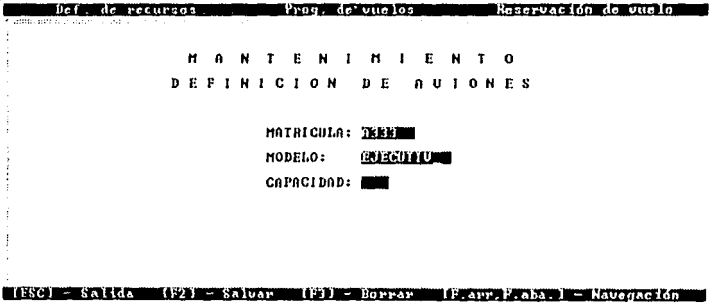
Si se presiona esta tecla estando en el último de los campos definidos para ese registro, el procedimiento se prepara para capturar el primero.

BACKSPACE. Si se presiona esta tecla estando en el final de la cadena borra un caracter a la izquierda del cursor. Pero si se presiona esta tecla estando en cualquier posición intermedia de la cadena, también borra un caracter a la izquierda del cursor, con la diferencia de que restaura toda la cadena recorriendo el resto de derecha a izquierda.

Tenemos la siguiente pantalla con la información mostrada:



Al presionar la tecla de BACKSPACE borramos el último caracter del campo MODELO.

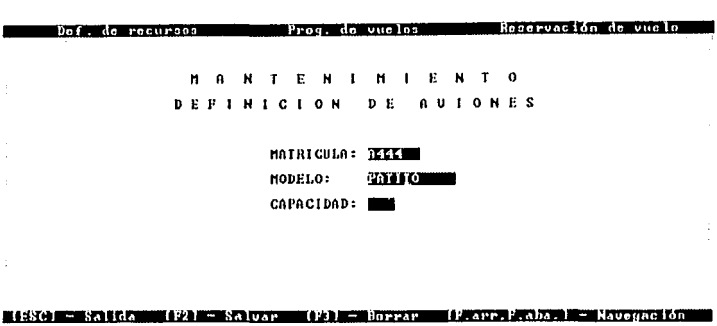


F2. En el momento de presionar esta tecla, se graba a disco la información capturada hasta entonces. Además da terminación al procedimiento de captura y restaura la pantalla anterior.

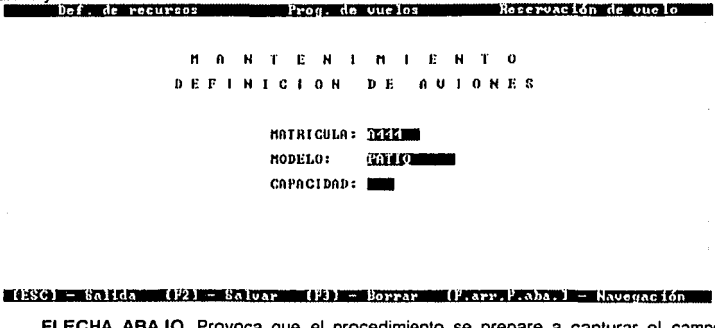
F3. Al presionar esta tecla, se da de baja lógica al registro visualizado en ese momento. Y al igual que la tecla F2 da terminación a la captura y restaura la pantalla anterior.

DELETE. Si se presiona esta tecla estando en cualquier posición intermedia de la cadena, borra el caracter sobre el cursor y restaura toda la cadena recorriendo el resto de los datos hacia la izquierda.

Esto se ilustra en la pantall siguiente:

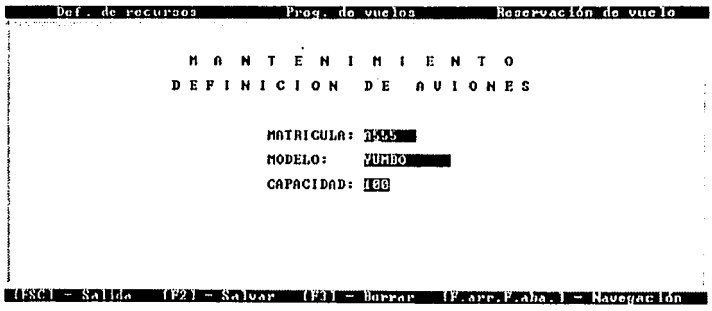


Al presionar la tecla de DELETE borramos el caracter en donde se encuentra el cursor y el resto de la cadena se recorre.

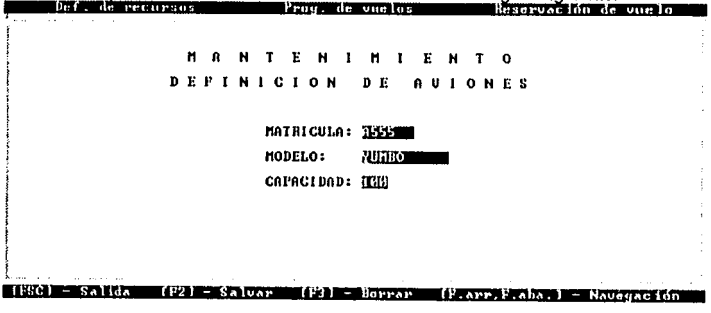


FLECHA ABAJO. Provoca que el procedimiento se prepare a capturar el campo siguiente del que se está capturando cuando se presiona esta tecla. Si el campo que se está capturando es el último, el procedimiento se dispone a capturar el primero.

Nos encontramos en el primer campo de captura:



Una vez presionada la tecla FLECHA ABAJO tenemos la figura siguiente:



FLECHA ARRIBA. Provoca que el procedimiento se prepare a capturar el campo anterior del que se estaba capturando cuando se presiona esta tecla. Si el campo que se estaba capturando era el primero, el procedimiento se dispone a capturar el último.

De la misma manera al presionar flecha arriba regresamos al primer caracter del campo anterior como se puede ver en la siguiente figura:

Si se presiona cualquier otra tecla, y esa tecla no corresponde al tipo de campo o al caracter válido, se presenta un mensaje indicando que se ha cometido un error y cuál es el tipo de caracter o tipo de dato válido. Por otro lado, si la tecla pulsada es válida para el tipo de campo, si se está en el modo de inserción, inserta el caracter en la posición que se captura y si se está en el modo de sobreescritura, sobreescribe el caracter en la posición que se captura.

El diagrama de flujo general del algoritmo de captura se muestra en la figura 9.1.1.1 parte 1 y 2.

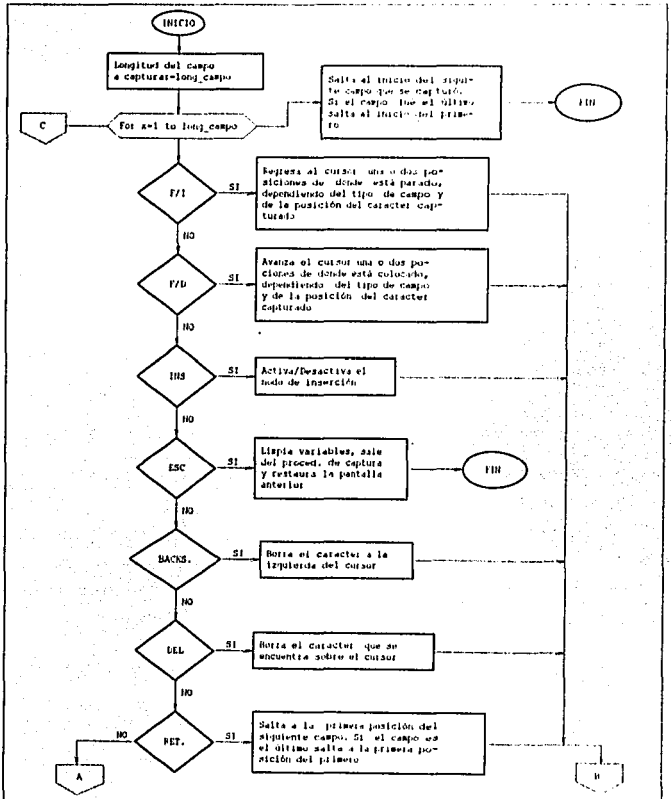


Figura 9.1.1.1. Diagrama de flujo general del algoritmo de captura (parte 1).

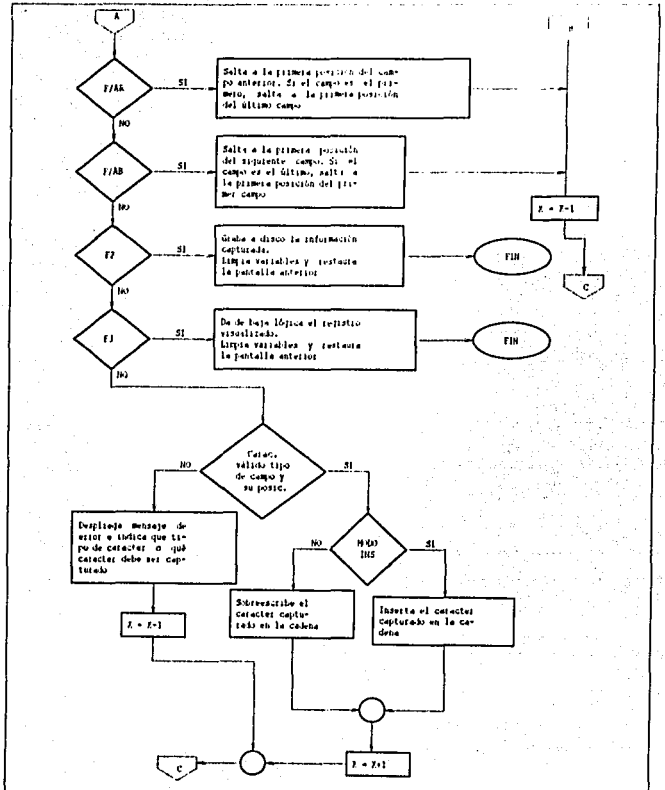


Figura 9.1.1.1. Diagrama de flujo general del algoritmo de captura (parte 2).

En seguida se presenta el programa fuente de la herramienta de captura.

```
/* DECLARACION DE ARCHIVOS CABECERA */

# include "ctype.h"
# include "dos.h"
# include "string.h"
# include "stdlib.h"
# include "graphics.h"


/* DECLARACION DE CONSTANTES PARA LA LECTURA DE TECLAS */

# define BACKSPACE 8
# define RETORNO 13
# define ESC 27
# define F_DOS 60
# define F_TRES 61
# define FLECHA_ARR 72
# define FLECHA_IZQ 75
# define FLECHA_DER 77
# define FLECHA_ABA 80
# define INS 82
# define DEL 83


/* DECLARACION DE TIPOS DE FUNCIONES Y PROCEDIMIENTOS */

void  vete_xy (),
      retardo (),
      tam_cursor (),
      flecha_der (),
      flecha_izq (),
      salto_atras (),
      salto_adelante (),
      limpia_estructura_captura (),
      graba_en (),
      mensaje_error (),
char  captura (),
      borra_izq (),
      sobreescribe (),
      flecha_arriba (),
      flecha_abajo (),
      borra_sobre_cursor ();


/* DECLARACION DE LAS VARIABLES UTILIZADAS */

int   ikera,
      y1,
      car_ant,
      bandera,
      banmodi,
      activar_ins = 0;
char  clase     [50],
      tipo_dato [1],
struct estructura_captura {
            int   contador_campos,
                  pos_x,
                  pos_y,
                  longitud,
            char tipo,
                  campo [50];
```

```
        } estructura_cap [15];
extern int   opc,
                                banf_dos,
                                desact_presenta,
                                ban_cons_modi,
                                ayuda_grabacion,
extern char or_des      [50],
                                capacidad [4];


/* ALMACENA EN LA ESTRUCTURA DECLARADA PARA TAL EFECTO LAS POSICIONES DE */
/* CAPTURA DE CADA UNO DE LOS CAMPOS QUE CONFORMAN UN REGISTRO .         */

crea_captura (num,contador,titulo,x_t,y_t,y,longitud,tipo)
int   num,
      contador,
      x_t,
      y_t,
      y,
      longitud;
char tipo,
     titulo [50];
{
        estructura_cap [num].contador_campos = contador;
        estructura_cap [num].pos_x           = x_t;
        estructura_cap [num].pos_y           = y;
        estructura_cap [num].longitud        = longitud;
        estructura_cap [num].tipo            = tipo;
        vete_xy (x_t,y_t);
        printf ("%s",titulo);
}


/* CAPTURA DE LOS CAMPOS. ADEMAS LLAMA A LOS PROCEDIMIENTOS Y FUNCIONES */
/* PARA VALIDACION DE TIPOS, SALTOS ADELANTE Y ATRAS, INSERCION, CONCA- */
/* TENACION, ETC, ETC.                                                  */

char captura (num)
int num;
{
        int   alf,
              pun,
              esp,
              nume,
              fec,
              hor,
              sig_campo,
              op;
        char p_num [3],

        tipo_dato [1] = '\0';
        y1            = estructura_cap [num].pos_y;
        activar_ins   = 0;
        tam_cursor (6,7);
        for (itera = 1;itera <= estructura_cap [num].longitud;itera++) {
                if (estructura_cap [num].tipo == 'a' || estructura_cap [num].tipo == 'n') {
                        vete_xy (estructura_cap [num].pos_x,y1);
                        tipo_dato [0] = checa_caracter (num);
                        if (tipo_dato [0] == -3) borra_izq (num);
                        else if (tipo_dato [0] == -4) borra_sobre_cursor (num);
                        else if (tipo_dato [0] == -6) {
                                if (activar_ins) tam_cursor (6,7);
                                flecha_izq (num);
                                car_ant = -6;
                        }
                        else if (tipo_dato [0] == -7) {
```

```
                if (activar_ins) tam_cursor (6,7);
                flecha_der (num);
        }
        else if (tipo_dato [0] == -8) {
                activar_ins = activa_desactiva_ins ();
        }
        else {
                if (tipo_dato [0] == -5) return;
                if (estructura_cap [num] tipo == 's') {
                        alf  = isalnum (tipo_dato [0]);
                        pun = ispunct (tipo_dato [0]);
                        esp = isspace (tipo_dato [0]);
                        switch (opc) {
                                case 1:
                                        if (num == 5 && itera == 1) {
                                                if (tipo_dato [0] != 'P') {
                                                        mensaje_error (5);
                                                        alf = 0;
                                                }
                                        }
                                        if ((num == 6 || num == 7) && itera == 1) {
                                                if (tipo_dato [0] != 'C') {
                                                        mensaje_error (6);
                                                        alf = 0;
                                                }
                                        }
                                        if ((num == 8 || num == 9 || num == 10) && itera == 1) {
                                                if (tipo_dato [0] != 'A') {
                                                        mensaje_error (7);
                                                        alf = 0;
                                                }
                                        }
                                break;

                                case 2:
                                        strcpy (p_num,"000");
                                        p_num [2] = tipo_dato [0];
                                        if (num == 4 && itera == 1) {
                                                if ((tipo_dato [0] != 'A') && (tipo_dato [0] != 'B') &&
                                                        (tipo_dato [0] != 'C')) {
                                                        mensaje_error (8);
                                                        alf = 0;
                                                }
                                        }
                                        if (num == 5 && itera == 1) {
                                                if (tipo_dato [0] != estructura_cap [4] campo [0]) {
                                                        mensaje_error (9);
                                                        alf = 0;
                                                }
                                        }
                                        if (num == 5 && itera == 2) {
                                                if (atoi(capacidad) == 90) {
                                                        if (estructura_cap [4] campo [0] == 'A') {
                                                                if (atoi(p_num) < 1 || atoi(p_num) > 2) {
                                                                        mensaje_error (10);
                                                                        alf = 0;
                                                                }
                                                        }
                                                        if (estructura_cap [4] campo [0] == 'B' ||
                                                                estructura_cap [4] campo [0] == 'C') {
                                                                if (atoi(p_num) < 1 || atoi(p_num) > 4) {
                                                                        mensaje_error (11);
                                                                        alf = 0;
                                                                }
                                                        }
                                                }
```

```
                                        if (atoi(capacidad) == 180) {
                                                if (estructura_cap [4] campo [0] == 'A') {
                                                        if (atoi(p_num) < 1 || atoi(p_num) > 4) {
                                                                mensaje_error (11);
                                                                alf = 0;
                                                        }
                                                }
                                                if (estructura_cap [4] campo [0] == 'B' ||
                                                        estructura_cap [4] campo [0] == 'C') {
                                                        if (atoi(p_num) < 1 || atoi(p_num) > 8) {
                                                                mensaje_error (12);
                                                                alf = 0;
                                                        }
                                                }
                                        }
                                }
                                if (num == 5 && itera == 3) {
                                        if (atoi(p_num) < 1) {
                                                mensaje_error (13);
                                                alf = 0;
                                        }
                                }
                        break;
                        case 5:
                                if (num == 0 && itera == 1) {
                                        if ((tipo_dato [0] != 'P') && (tipo_dato [0] != 'C') &&
                                                (tipo_dato [0] != 'A')) {
                                                mensaje_error (15);
                                                alf = 0;
                                        }
                                }
                        break;
                        case 6:
                                if (num == 1 && itera == 1) {
                                        if ((tipo_dato [0] != 'A') && (tipo_dato [0] != 'B') &&
                                                (tipo_dato [0] != 'C')) {
                                                mensaje_error (8);
                                                alf = 0;
                                        }
                                }
                        break;
                        case 7:
                                if (num == 0 && itera == 1) {
                                        if ((tipo_dato [0] != 'A') && (tipo_dato [0] != 'B') &&
                                                (tipo_dato [0] != 'C')) {
                                                mensaje_error (8);
                                                alf = 0;
                                        }
                                }
                        break;
                }
                if ((alf == 0) && (pun == 0) && (esp == 0)) {
                        if (tipo_dato [0] != -3) retardo (num);
                }
                else {
                        op = tipo_a_y_n (num);
                        if (op == 1) itera = estructura_cap [num] longitud;
                }
        }
        else if (estructura_cap [num] tipo == 'n') {
                nume = isdigit (tipo_dato [0]);
                if (nume == 0) {
                        if (tipo_dato [0] != -3) retardo (num);
                }
```

```
                                        else {
                                                op = tipo_a_y_n (num);
                                                if (op == 1) itera = estructura_cap [num] longitud;
                                        }
                                }
                        }
                }
                if (estructura_cap [num] tipo == 'f' || estructura_cap [num] tipo == 'h') {
                        vete_xy (estructura_cap [num] pos_x,y1);
                        tipo_dato [0] = checa_caracter (num);
                        if (tipo_dato [0] == -6) {
                                flecha_izq (num);
                                car_ant = -6;
                        }
                        else if (tipo_dato [0] == -7) {
                                flecha_der (num);
                        }
                        else if (tipo_dato [0] == -8) {
                                retardo (num);
                                if (itera <= 0) itera = 0;
                                else itera-=1;
                        }
                        else {
                                if (tipo_dato [0] == -5) return;
                                if (estructura_cap [num] tipo == 'f') {
                                        fec = isdigit (tipo_dato [0]);
                                        if (fec == 0) {
                                                if (tipo_dato [0] != -3) retardo (num);
                                        }
                                        else {
                                                op = tipo_f_y_h (num);
                                                if (op == 1) itera = estructura_cap [num] longitud;
                                        }
                                }
                                else if (estructura_cap [num] tipo == 'h') {
                                        hor = isdigit (tipo_dato [0]);
                                        if (hor == 0) {
                                                if(tipo_dato [0] != -3) retardo (num);
                                        }
                                        else {
                                                op = tipo_f_y_h (num);
                                                if (op == 1) itera = estructura_cap [num] longitud;
                                        }
                                }
                        }
                }
        }
        if (num == 0) pone_sombras (1);
        activar_ins = 0;
        fam_cursor (6,7);
        banmodi = 0;
        sig_campo = transacciones (num);
        if (sig_campo == 1) captura (num);
        else {
                num+=1;
                if (num > estructura_cap [num] contador_campos) {
                        num = 0;
                        if (desact_presenta == 1) {
                                switch (opc) {
                                        case 1:
                                        case 3:
                                        case 4:
                                        case 5:
                                        case 7:
                                                if (ban_cons_modi == 0) captura (num);
                                                else captura (num+=1);
```

```
                                        break;

                                        case 2:
                                        case 6:
                                                if (ban_cons_modi == 0) captura (num);
                                                else captura (num+=2);
                                        break;
                                }
                        }
                        else captura (num);
                }
                else captura (num);
        }
}

/* COMPLEMENTO PARA LA CAPTURA DE DATOS DE TIPO ALFANUMERICO Y NUMERICO */

tipo_a_y_n (num)
int num;
{
        int opera;

        if (car_ant == -6) {
                sobreescribe (num);
        }
        else if (activar_ins) {
                opera = insertar (num);
                if (opera == 1) return 1;
                else salto_adelante (num);
        }
        else sobreescribe (num);
        return 0;
}

/* COMPLEMENTO PARA LA CAPTURA DE DATOS DE TIPO FECHA Y HORA */

tipo_f_y_h (num)
int num;
{
        int pasa,
            opera;

        pasa = valida (num);
        if (pasa) {
                retardo (num);
                return 0;
        }
        else if (car_ant == -6) sobreescribe (num);
        else if (activar_ins) {
                opera = insertar (num);
                if (opera == 1) return 1;
                else salto_adelante (num);
        }
        else sobreescribe (num);
        return 0;
}

/* CHECA QUE TIPO DE TECLA SE HA PULSADO Y EJECUTA EL BLOQUE DE CODIGO */
/* ASIGNADO                                                            */

checa_caracter (num)
int num;
{
```

```
union inkey {
        char ch [2];
        int i;
} c;
int sig_campo,
    tnum = num;

while (!bioskey (1));
        c.i = bioskey (0);
        if (c.ch [0]) {
                switch (c.ch [0]) {
                        case BACKSPACE:
                                return -3;

                        case RETORNO:
                                if (ban_cons_modi == 0) {
                                        switch (opc) {
                                                case 1:
                                                case 3:
                                                case 4:
                                                case 5:
                                                case 7:
                                                        pone_sombras (1);
                                                break;

                                                case 2:
                                                case 6:
                                                        pone_sombras (2);
                                                break;
                                        }
                                }
                                banmodi = 0;
                                sig_campo = transacciones (tnum);
                                if (sig_campo == 1) captura (tnum);
                                else {
                                        activar_ins = 0;
                                        fam_cursor (6,7);
                                        tnum+=1;
                                        if (tnum > estructura_cap [tnum] contador_campos) {
                                                tnum = 0;
                                                if (desact_presenta == 1) {
                                                        switch (opc) {
                                                                case 1:
                                                                case 3:
                                                                case 4:
                                                                case 5:
                                                                case 7:
                                                                        if (ban_cons_modi == 0) captura (tnum);
                                                                        else captura (tnum+=1);
                                                                break;

                                                                case 2:
                                                                case 6:
                                                                        if (ban_cons_modi == 0) captura (tnum);
                                                                        else captura (tnum+=2);
                                                                break;
                                                        }
                                                }
                                                else captura (tnum);
                                        }
                                        else captura (tnum);
                                }
                        break;

                        case ESC:
                                limpia_estructura_captura ();
```

```
                                        activar_ins = desact_presenta = banmodi = 0,
                                        tam_cursor (6,7);
                                        return -5;
                                }
                        }
                        else {
                                switch (c_ch [1]) {
                                        case F_DOS:
                                                activar_ins = 0,
                                                tam_cursor (6,7);
                                                if (banf_dos == 1 && ayuda_grabacion == 1) {
                                                        banmodi = 1;
                                                        graba_en ();
                                                }
                                                desact_presenta = 0;
                                                limpia_estructura_captura ();
                                                return -5;

                                        case F_TRES:
                                                if (banf_dos == 1) {
                                                        banmodi = desact_presenta = 0,
                                                        baja ();
                                                }
                                                limpia_estructura_captura ();
                                                return -5;

                                        case FLECHA_IZQ:
                                                return -6;

                                        case FLECHA_DER:
                                                return -7;

                                        case INS:
                                                return -8;

                                        case DEL:
                                                return -4;

                                        case FLECHA_ARR:
                                                banmodi = 0;
                                                if (desact_presenta == 1) {
                                                        switch (opc) {
                                                                case 1:
                                                                case 3:
                                                                case 4:
                                                                case 5:
                                                                case 7:
                                                                        if (tnum == 1) {
                                                                                tnum = estructura_cap [1].contador_campos;
                                                                                captura (tnum-=1);
                                                                        }
                                                                        else captura (tnum-=1);
                                                                break;

                                                                case 2:
                                                                        if (ban_cons_modi == 0) {
                                                                                if (tnum == 0) captura (1);
                                                                                if (tnum == 1) captura (0);
                                                                        }
                                                                        else {
                                                                                if      (tnum == 1) captura (5);
                                                                                else if (tnum == 2) captura (5);
                                                                                else
captura (tnum-=1);
                                                                        }
                                                                break;
```

```
                                        case 6:
                                                if (ban_cons_modi == 0) {
                                                        if (tnum == 0) captura (1);
                                                        if (tnum == 1) captura (0);
                                                }
                                                else {
                                                        if (tnum == 1 || tnum == 2) captura (2);
                                                }
                                        break;
                                        }
                                }
                                else {
                                        switch (opc) {
                                                case 1:
                                                case 3:
                                                case 4:
                                                case 5:
                                                case 7:
                                                if (tnum == 0) {
                                                        tnum = estructura_cap [1] contador_campos;
                                                        captura (tnum-=1);
                                                }
                                                else captura (tnum-=1);
                                                break;

                                                case 2:
                                                if (ban_cons_modi == 0) {
                                                        if (tnum == 0) captura (1);
                                                        if (tnum == 1) captura (0);
                                                }
                                                else {
                                                        if (tnum == 0) captura (5);
                                                        if (tnum == 1) captura (0);
                                                        if (tnum == 2) captura (1);
                                                        if (tnum == 3) captura (2);
                                                        if (tnum == 4) captura (3);
                                                        if (tnum == 5) captura (4);
                                                }
                                                break;

                                                case 6:
                                                if (ban_cons_modi == 0) {
                                                        if (tnum == 0) captura (1);
                                                        if (tnum == 1) captura (0);
                                                }
                                                else {
                                                        if (tnum == 0) captura (2);
                                                        if (tnum == 1) captura (0);
                                                        if (tnum == 2) captura (1);
                                                }
                                                break;
                                        }
                                }
                        break;

                        case FLECHA_ABA:
                                banmodi = 0;
                                tnum+=1;
                                if (tnum > estructura_cap [tnum] contador_campos) {
                                        tnum = 0;
                                        if (desact_presenta == 1) {
                                                switch (opc) {
                                                        case 1:
                                                        case 3:
                                                        case 4:
```

```
                                                        case 5:
                                                        case 7:
                                                                if (ban_cons_modi == 0) captura (tnum);
                                                                else captura (tnum+=1);
                                                        break;

                                                        case 2:
                                                        case 6:
                                                                if (ban_cons_modi == 0) captura (tnum);
                                                                else captura (tnum+=2);
                                                        break;
                                                }
                                        }
                                        else captura (tnum);
                                }
                                else captura (tnum);
                        break;
                }
        }
}

/* SE GENERA UN MENSAJE DE ERROR CON RETARDO CUANDO NO SE PRESIONA UNA */
/* TECLA VALIDA PARA EL TIPO DE CAMPO QUE SE ESTA CAPTURANDO          */

void retardo (num)
int num;
{
        switch (opc) {
                case 1:
                        if (num == 0) mensaje_error (14);
                        if (num == 3) {
                                if (itera == 1) mensaje_error (1);
                                if (itera == 2 || itera == 3) mensaje_error (2);
                                if (itera == 4) mensaje_error (3);
                        }
                        if (num == 4) {
                                if (itera == 1) mensaje_error (3);
                                if (itera == 2) mensaje_error (1);
                                if (itera == 3) mensaje_error (4);
                        }
                break;

                case 2:
                        if (num == 0) mensaje_error (14);
                break;

                case 3:
                        if (num == 2) mensaje_error (14);
                break;

                case 6:
                        if (num == 2) mensaje_error (14);
                break;
        }
        vete_xy (estructura_cap [num] pos_x,y1);
        if (itera >= 0) itera-=1;
        else itera = itera;
}

/* SOBREESCRIBE CARACTERES EN LOS CAMPOS CAPTURADOS, ADEMAS CONCATENA */
/* CUANDO ASI SEA NECESARIO                                          */

char sobreescribe (num)
int num;
```

```
{
        estructura_cap [num] campo [itera-1] = tipo_dato [0];
        vete_xy (estructura_cap [num] pos_x,y1);
        printf ("%c",tipo_dato [0]);
        salto_adelante (num);
}


/* PROVOCA QUE EL CURSOR SALTE HACIA ADELANTE CUANDO SE PRESIONA LA FLECHA */
/* DERECHA                                                                 */

void flecha_der (num)
int num;
{
        salto_adelante (num);
        vete_xy (estructura_cap [num] pos_x,y1);
}


/* CUANDO SE PRESIONA BACKSPACE SE BORRA EL CARACTER QUE SE ENCUENTRA A LA */
/* IZQUIERDA DEL CURSOR                                                    */

char borra_izq (num)
int num;
{
        int   itera_temp,
              longitud,
              y1_temp;
        char parte_uno [50],
             parte_dos [50];
        register int concatena;

        if (y1 <= estructura_cap [num] pos_y) {
                itera = 0;
                vete_xy (estructura_cap [num] pos_x,estructura_cap [num] pos_y);
        }
        else {
                longitud = strlen (estructura_cap [num] campo);
                if (longitud <= estructura_cap [num] longitud) {
                        y1_temp = estructura_cap [num] pos_y;
                        for (concatena = 0;concatena <= estructura_cap [num] longitud;concatena++) {
                                vete_xy (estructura_cap [num] pos_x,y1_temp);
                                printf (" ");
                                y1_temp+=1;
                        }
                        itera-=2;
                        itera_temp = itera;
                        strncpy (parte_uno,estructura_cap [num] campo,itera_temp);
                        parte_uno [itera_temp] = '\0';
                        for (concatena = 0; concatena <= longitud-itera;concatena++) {
                                parte_dos [concatena] = estructura_cap [num] campo [itera_temp+1];
                                itera_temp+=1;
                        }
                        strcpy (estructura_cap [num] campo,parte_uno);
                        strcat (estructura_cap [num] campo,parte_dos);
                        switch (estructura_cap [num] tipo) {
                                case 'f':
                                        visualiza_f_y_h ('f',estructura_cap [num] pos_x,estructura_cap [num] pos_y,
                                                                                        estructura_cap [num] campo [0],estructura_cap
[num] campo [1],
                                                                                        estructura_cap [num] campo [2],estructura_cap
[num] campo [3],
                                                                                        estructura_cap [num] campo [4],estructura_cap
[num] campo [5]);
                                        break;
```

```
                                        case 'h':
                                                visualiza_f_y_h ('h',estructura_cap [num] pos_x,estructura_cap [num] pos_y,
                                                                                        estructura_cap [num] campo [0],estructura_cap
[num] campo [1],
                                                                                        estructura_cap [num] campo [2],estructura_cap
[num] campo [3],
                                                                                        ' ');
                                                break;

                                        default:
                                                visualiza_campo (estructura_cap [num] pos_x,estructura_cap [num] pos_y,
                                                                                        estructura_cap [num] campo),
                                                break;
                                }
                        }
                }
                if (y1 > estructura_cap [num] pos_y) {
                        salto_atras (num);
                        vete_xy (estructura_cap [num] pos_x,y1);
                }
}

/* DEPENDIENDO DEL CAMPO EN EL QUE SE ESTE PARADO, VALIDA QUE LOS */
/* CARACTERES PULSADOS SEAN VALIDOS PARA ESE TIPO DE CAMPO        */

valida (num)
int num;
{
        switch (estructura_cap [num] tipo) {
                case 'f':
                        switch (itera) {
                                case 1:
                                        if (tipo_dato [0] >= '4') return 1;
                                return 0;

                                case 2:
                                        if (estructura_cap [num] campo [itera-2] >= '3') {
                                                if (tipo_dato [0] >= '2') return 1;
                                        }
                                return 0;

                                case 3:
                                        if (tipo_dato [0] >= '2') return 1;
                                return 0;

                                case 4:
                                        if (estructura_cap [num] campo [itera-2] >= '1') {
                                                if (tipo_dato [0] >= '3') return 1;
                                        }
                                return 0;

                                case 5:
                                case 6:
                                        return 0;
                        }
                return 0;

                case 'h':
                        switch (itera) {
                                case 1:
                                        if (tipo_dato [0] >= '3') return 1;
                                return 0;

                                case 2:
                                        if (estructura_cap [num] campo [itera-2] >= '2') {
```

```
                                        if (tipo_dato [0] >= '4') return 1;
                                    }
                                return 0;

                                case 3:
                                        if (tipo_dato [0] >= '6') return 1;
                                return 0,

                                case 4:
                                        return 0;
                            }
                    return 0;
            }
}


/* PROVOCA QUE EL CURSOR SALTE A LA IZQUIERDA CUANDO SE PRESIONA LA FLECHA */
/* IZQUIERDA                                                                */

void flecha_izq (num)
int num;
{
        itera-=2;
        if (y1 <= estructura_cap [num] pos_y) {
                itera = 0;
                vete_xy (estructura_cap [num] pos_x,estructura_cap [num] pos_y);
        }
        else {
                salto_atras (num);
                vete_xy (estructura_cap [num] pos_x,y1);
        }
}


/* SALTA UNO O DOS ESPACIOS HACIA ATRAS DE DONDE ESTA PARADO EL CURSOR, ESTO */
/* DEPENDE DEL TIPO DE CAMPO QUE SE ESTE CAPTURANDO                          */

void salto_atras (num)
int num;
{
        if ((estructura_cap [num] tipo == 'a') || (estructura_cap [num]tipd == 'rf')) y1-=1;
        else {
                switch (estructura_cap [num] tipo) {
                        case 'f':
                                switch (itera) {
                                        case 0:
                                                y1-=1;
                                        break;

                                        case 1:
                                                y1-=1;
                                        case 2:
                                                y1-=1;
                                        break;

                                        case 3:
                                                y1-=1;

                                        case 4:
                                                y1-=1;
                                        break;

                                        case 5:
                                                y1-=1;
                                        break;
```

```
                                        case 6:
                                                y1-=1;
                                                break;
                                        }
                                break;

                                case 'h':
                                        switch (itera) {
                                                case 0:
                                                        y1-=1;
                                                break;

                                                case 1:
                                                case 2:
                                                        y1-=1;
                                                break;

                                                case 3:
                                                        y1-=1;
                                                break;

                                                case 4:
                                                        y1-=1;
                                                break;
                                        }
                                break;
                        }
                }
}


/* SALTA UNO O DOS ESPACIOS HACIA ADELANTE DE DONDE ESTA PARADO EL */
/* CURSOR, ESTO DEPENDE DEL TIPO DE CAMPO QUE SE ESTE CAPTURANDO */

void salto_adelante (num)
int num;
{
        if ((estructura_cap [num] tipo== 'e') ||
            (estructura_cap [num] tipo == 'n')) y1+=1;
        if (estructura_cap [num] tipo == 'f') {
                if ((itera == 1) || (itera == 3) || (itera == 5) || (itera == 6)) y1+=1;
                if ((itera == 2) || (itera == 4)) y1+=2;
        }
        if (estructura_cap [num] tipo == 'h') {
                if ((itera == 1) || (itera == 3) || (itera == 4)) y1+=1;
                if (itera == 2) y1+=2;
        }
}



/* CHECA SI LA INSERCION ESTA ACTIVADA O NO. SI LO ESTA LA DESCATIVA, EN CASO */
/* CONTRARIO LA ACTIVA                                                        */

activa_desactiva_ins ()
{
        if (itera <= 0) itera = 0;
        else itera-=1;
        if (activar_ins == 0) {
                tam_cursor (0,7);
                car_ant = 0;
                return 1;
        }
        else {
                tam_cursor (6,7);
```

```
                return 0;
        }
}

/* CUANDO SE ACTIVA LA INSERCION ESTE BLOQUE DE CODIGO HACE LA ACTUALIZACION */
/* DEL CAMPO CAPTURADO                                                       */

insertar (num)
int num;
{
        int   itera_temp;
        char parte_uno [50],
             parte_dos [50],
        register int concatena;

        if (strlen (estructura_cap [num].campo) < estructura_cap [num].longitud) {
                itera_temp = itera;
                strncpy (parte_uno,estructura_cap [num].campo,itera_temp-1);
                parte_uno [itera_temp-1] = '\0';
                strcat (parte_uno,tipo_dato);
                for (concatena = 0; concatena <= (strlen (estructura_cap [num].campo)-itera)+1;concatena++) {
                        parte_dos [concatena] = estructura_cap [num].campo [itera_temp-1];
                        itera_temp+=1;
                }
                strcpy (estructura_cap [num].campo,parte_uno);
                strcat (estructura_cap [num].campo,parte_dos);
                vete_xy (estructura_cap [num].pos_x,estructura_cap [num].pos_y);
                printf ("%s",estructura_cap [num].campo);
        }
        else if (strlen(estructura_cap [num].campo) == estructura_cap [num].longitud) {
                return 1;
        }
        return 0;
}

/* POSICIONA EL CURSOR EN LAS COORDENADAS X E Y DADAS */

void vete_xy (x,y)
int x,
    y;
{
        union REGS r;

        r.h.ah = 2;
        r.h.dl  = y;
        r.h.dh = x;
        r.h.bh = 0;
        int86 (0x10,&r,&r);
}
/* HACE EL CURSOR GRANDE O PEQUEÑO, DEPENDIENDO DE SI LA INSERCION ESTA */
/* ACTIVADA O NO                                                        */

void tam_cursor (comienzo,fin)
char comienzo,
     fin;
{
        union REGS r;

        r.h.ah = 1;
        r.h.ch = comienzo;
        r.h.cl  = fin;
        int86 (0x10,&r,&r);
}
```

```
/* CUANDO SE PRESIONA DELETE SE BORRA EL CARACTER QUE EN ESE MOMENTO SE */
/* ENCUENTRA SOBRE EL CURSOR                                            */

char borra_sobre_cursor (num)
int num;
{
        int   itera_temp,
              longitud,
              y1_temp;
        char parte_uno [50],
                   parte_dos [50];
        register int concatena;

        if (y1 < estructura_cap [num] pos_y) {
                itera = 0;
                vete_xy (estructura_cap [num] pos_x,estructura_cap [num] pos_y);
        }
        else {
                longitud = strlen (estructura_cap [num] campo);
                if (longitud <= estructura_cap [num] longitud) {
                        y1_temp = estructura_cap [num] pos_y;
                        for (concatena = 0;concatena <= estructura_cap [num] longitud;concatena++) {
                                vete_xy (estructura_cap [num] pos_x,y1_temp);
                                printf (" ");
                                y1_temp+=1;
                        }
                        itera-=1;
                        itera_temp = itera;
                        strncpy (parte_uno,estructura_cap [num] campo,itera_temp);
                        parte_uno [itera_temp] = '\0';
                        for (concatena = 0; concatena <= longitud-itera;concatena++) {
                                parte_dos [concatena] = estructura_cap [num] campo [itera_temp+1];
                                itera_temp+=1;
                        }
                        strcpy (estructura_cap [num] campo,parte_uno);
                        strcat (estructura_cap [num] campo,parte_dos);
                        vete_xy (estructura_cap [num] pos_x,estructura_cap [num] pos_y);
                        switch (estructura_cap [num] tipo) {
                                case 'f':
                                        printf ("%c%c",estructura_cap [num] campo [0],estructura_cap [num] campo [1]);
                                        vete_xy (estructura_cap [num] pos_x,estructura_cap [num] pos_y+2);
                                        printf ("/");
                                        printf ("%c%c",estructura_cap [num] campo [2],estructura_cap [num] campo [3]);
                                        printf ("/");
                                        vete_xy (estructura_cap [num] pos_x,estructura_cap [num] pos_y+6);
                                        printf ("%c%c",estructura_cap [num] campo [4],estructura_cap [num] campo [5]);
                                break;
                                case 'h':
                                        printf ("%c%c",estructura_cap [num] campo [0],estructura_cap [num] campo [1]);
                                        vete_xy (estructura_cap [num] pos_x,estructura_cap [num] pos_y+2);
                                        printf (":");
                                        printf ("%c%c",estructura_cap [num] campo [2],estructura_cap [num] campo [3]);
                                break;

                                default:
                                        printf ("%s",estructura_cap [num] campo);
                                break;
                        }
                }
        }
}

/* LIMPIA LAS VARIABLES DE LA ESTRUCTURA DE CAPTURA */
```

```
void limpia_estructura_captura ()
{
        register int limpia;

        for (limpia = 0;limpia <= 15;limpia++) {
                estructura_cap [limpia] contador_campos = 0;
                estructura_cap [limpia] pos_x           = 0;
                estructura_cap [limpia] pos_y           = 0;
                estructura_cap [limpia] longitud        = 0;
                estructura_cap [limpia] tipo            = '\0';
                strncpy (estructura_cap [limpia] campo," ",50);
        }
}


/* REALIZA LAS TRANSACCIONES NECESARIAS PARA LA EXTRACCION DE CAMPOS YA */
/* EXISTENTES EN OTRAS BASES DE DATOS, O BIEN, CHECAR SI ALGUNA LLAVE YA */
/* EXISTE EN LA BASE DE DATOS CON LA QUE SE ESTA TRABAJANDO EN ESE */
/* MOMENTO                                                          */

transacciones (num)
int num;
{
        int  sigue;
        char claves [50];

  strncpy (claves," ",50);
        switch (opc) {
                case 1:
                        if (num == 0) {
                                strcpy (claves,estructura_cap [0] campo);
                                introduce (claves,1);
                        }
                        else if (num == 1) {
                                strcpy (claves,estructura_cap [1] campo);
                                sigue = checa_disponibilidad (claves,num);
                                if (sigue == 1) return 1;
                                sigue = trae_capacidad (claves);
                                if (sigue == 1) return 1;
                        }
                        else if (num == 2) {
                                strcpy (claves,estructura_cap [2] campo);
                                sigue = trae_escalas (claves);
                                if (sigue == 1) return 1;
                        }
                        else if (num==5 || num==6 || num==7 || num==8 || num==9 || num==10) {
                                strcpy (claves,estructura_cap [num] campo);
                                sigue = checa_disponibilidad (claves,num);
                                if (sigue == 1) return 1;
                                trae_nombre_empleado (claves,num);
                        }
                break;

                case 2:
                        if (num == 0) {
                                strcpy (claves,estructura_cap [0] campo);
                                sigue = checa_capacidad (claves);
                                if (sigue == 1) return 1;
                                sigue = trae_fecha_hora_ordes (claves);
                                if (sigue == 1) return 1;
                        }
                        else if (num == 1) {
                                strcpy (claves,estructura_cap [0] campo);
                                strcat (claves,estructura_cap [1] campo);
                                introduce (claves,1);
                        }
```

```
                        else if (num == 4) {
                                cls (12,39,20,76);
                                strncpy (clase," ",50);
                                strncpy (clase,estructura_cap [4] campo,1);
                                strncpy (claves,or_des,6);
                                strcat (claves,clase);
                                sigue = trae_costo (claves);
                                if (sigue == 1) return 1;
                        }
                        else if (num == 5) {
                                sigue = checa_asiento (estructura_cap [0] campo,
                                                                        estructura_cap [5] campo);
                                if (sigue == 1) return 1;
                        }
                break;

                case 3:
                case 4:
                case 5:
                case 7:
                        if (num == 0) {
                                strcpy (claves,estructura_cap [0] campo);
                                introduce (claves,1);
                        }
                break;

                case 6:
                        if (num == 1) {
                                strcpy (claves,estructura_cap [0].campo);
                                strcat (claves,estructura_cap [1] campo);
                                introduce (claves,1);
                        }
                break;
        }
        return 0;
}

/* CUANDO SE PRESIONA "F2" SE HACE LA GRABACION DEL REGISTRO CAPTURADO A LA */
/* B.D. CORRESPONDIENTE                                                     */

void graba_en ()
{
        char claves [50];

        switch (opc) {
                case 1:
                case 3:
                case 4:
                case 5:
                case 7:
                        strcpy (claves,estructura_cap [0] campo);
                        introduce (claves,3);
                break;

                case 2:
                case 6:
                        strcpy (claves,estructura_cap [0] campo);
                        strcat (claves,estructura_cap [1] campo);
                        introduce (claves,3);
                break;
        }
        if (bandera == 1) salva_modific ();
        if (bandera == 0) salva ();
        limpia_estructura_captura ();
}
```

```
/* SERIE DE MENSAJES PARA ERRORRES EN LA CAPTURA */

void mensaje_error (error)
int error;
{
        switch (error) {
                case 1:
                        centrar (21,"\a*** El caracter debe ser num,rico y no mayor a 3 ***");
                break;

                case 2:
                        centrar (21,"\a*** El caracter debe ser num,rico y no mayor a 1 ***");
                break;

                case 3:
                        centrar (21,"\a*** El caracter debe ser num,rico y no mayor a 2 ***");
                break;

                case 4:
                        centrar (21,"\a*** El caracter debe ser num,rico y no mayor a 5 ***");
                break;

                case 5:
                        centrar (21,"\a*** El caracter debe ser 'P' ***");
                break;

                case 6:
                        centrar (21,"\a*** El caracter debe ser 'C' ***");
                break;

                case 7:
                        centrar (21,"\a*** El caracter debe ser 'A' ***");
                break;

                case 8:
                        centrar (21,"\a*** El caracter debe ser 'A', 'B' ó 'C' ***");
                break;

                case 9:
                        centrar (21,"\a*** El caracter debe ser igual al de la clase ***");
                break;

                case 10:
                        centrar (21,"\a*** El numero debe ser > 0 y < 3 ***");
                break;

                case 11:
                        centrar (21,"\a*** El numero debe ser > 0 y < 5 ***");
                break;

                case 12:
                        centrar (21,"\a*** El numero debe ser > 0 y < 9 ***");
                break;

                case 13:
                        centrar (21,"\a*** El numero debe ser > 0 ***");
                break;

                case 14:
                        centrar (21,"\a*** Debe ser un caracter num,rico ***");
                break;

                case 15:
                        centrar (21,"\a*** El caracter debe ser 'P', 'C' ó 'A' ***");
```

```
                break;
                case 16:
                        centrar (21,"\a*** Avion no disponible ***");
                break;
                case 17:
                        centrar (21,"\a*** Empleado no disponible ***");
                break;
                case 18:
                        centrar (21,"\a*** Ya no hay cupo ***");
                break;
                case 19:
                        centrar (21,"\a*** Asiento reservado ***");
                break;
                case 20:
                        centrar (21,"\a*** El registro esta dado de baja ***");
                        desact_presenta = ayuda_grabacion = 0;
                break;
                case 21:
                        centrar (21,"\a*** El avion esta dado de baja. Elija otro ***");
                break;
                case 22:
                        centrar (21,"\a*** El Origen / Destino esta dado de baja  Elija otro ***");
                break;
                case 23:
                        centrar (21,"\a*** El registro de Programacion de vuelos esta dado de baja ***");
                break;
                case 24:
                        centrar (21,"\a*** El registro de Costo por clase de vlo. esta dado de baja ***");
                break;
        }
        delay (1000);
        cls (21,3,21,78);
}
```

IX.1.2. DESCRIPCION Y FUNCIONAMIENTO DEL ALGORITMO INDEXADOR

Para el diseño y programación del algoritmo indexador (que genera archivos índices .ndx de las bases de datos .dbf), el cual se encarga de la inserción, búsqueda, modificación y eliminación de registros en nuestro sistema de base de datos, se consideraron algunos puntos que debemos mencionar.

En primer lugar se utiliza una estructura de datos llamada *árbol binario.*

Esta estructura de datos es muy importante para la inserción y búsqueda de datos, además de la recuperación de información. La representación gráfica de un árbol de búsqueda binaria se puede ver en la figura 9.1.2.1.

Un árbol binario es aquel en el cual cada nodo contiene una llave que satisface las siguientes condiciones:

1.- Todas la llaves (si las hay) en el subárbol izquierdo de la raíz preceden a la llave en la raíz.

2.- La llave en la raíz precede a todas (si las hay) las que se hallen en su subárbol derecho.

3,- Los árboles izquierdo y derecho de la raíz son también árboles de búsqueda.

Todo árbol binario contiene un nodo raíz que al inicio es un nil (nulo) y un subárbol derecho e izquierdo. Estos a su vez son nodos y tienen un subárbol derecho e izquierdo y así sucesivamente.

Para la inserción, búsqueda, modificación y borrado de un registro se pueden seguir los siguientes pasos.

a) Comparamos la llave introducida con la llave raíz del árbol. Si no es la misma entonces pasamos al subárbol derecho o izquierdo según nos convenga y repetimos de nuevo la búsqueda.

b) Si encontramos la llave se realizará el procedimiento requerido (modificación o baja) en caso contrario continuaremos la búsqueda hasta encontrar un subárbol vacío (nulo) y si lo deseamos, hacer una inserción.

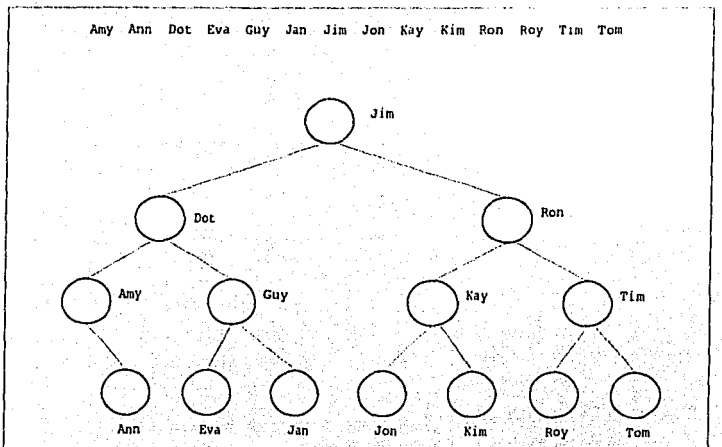


Figura 9.1.2.1. Representación gráfica de un árbol binario.

Hay tres métodos principales para recorrer un árbol binario los cuales son:

1. Preorden. En este recorrido se visita el nodo antes que los subárboles.

2. Enorden. Se visita el árbol entre el nodo izquierdo y el derecho.

3. Postorden. Se visita la raíz después de ambos subárboles.

El algoritmo indexador se basa en el método de preorden para la modificación y baja de los registros dentro de las bases de datos en el sistema.

A continuación se dará una explicación general del funcionamiento de este algoritmo tan importante dentro del sistema.

En primer lugar al introducir un registro con su respectiva o respectivas llaves de acceso, el algoritmo verifica que no se halla creado un árbol binario, si es así crea un nodo raíz con esta o estas llaves además de sus respectivos subárbol izquierdo y subárbol derecho puestos a nulo.

A su vez se crea un archivo de datos (.dbf) y su correspondiente archivo índice (.ndx) el cual contiene el apuntador que nos proporcionará la posición del registro en la base de datos. Cabe mencionar que todos los registros en las bases de datos son de longitud fija.

Una vez creada la raíz se procede a la inserción del resto de la información, la cual será manipulada de la siguiente manera:

1.- Se lleva un contador de registros introducidos el cual se utilizará para la actualización de los apuntadores del subárbol izquierdo (AI) y el subárbol derecho (AD).

2.- Al introducir un registro se compara primeramente con el apuntador del nodo raíz, si la llave introducida es menor que la llave del nodo, se crea un nodo a la izquierda y sus apuntadores derecho e izquierdo se ponen a nulo.

3.- Si la llave introducida es mayor que la llave de la raíz, entonces se crea un nodo a la derecha y sus apuntadores derecho e izquierdo se ponen a nulo.

4.- Al introducir más registros se vuelven a realizar los pasos anteriores para crear el archivo índice.

Como se mencionó anteriormente, el indexador toma del archivo índice previamente generado, los apuntadores izquierdo o derecho, los cuales nos proporcionarán la posición del registro en el archivo de datos. Por lo tanto siempre se tendrá un archivo de datos (.dbf) y un archivo de índices (.ndx).

Para explicar lo antes mencionado tomemos el siguiente ejemplo:

Supongamos que tenemos 10 registros de longitud fija de los cuales sólo tomaremos el campo nombre como llave principal.

Supongamos que tenemos 10 registros de longitud fija de los cuales sólo tomaremos el campo nombre como llave principal.

Las llaves primarias introducidas en el indexador se pueden ver en la tabla de la figura 9.1.2.2.

No. de Reg.	Llave primaria
1	CARLOS
2	JESUS
3	PEDRO
4	JUAN
5	ALBERTO
6	ADOLFO
7	FERNANDO
8	ALMA
9	ROSA
10	TERESA

Figura 9.1.2.2. Llaves primarias introducidas al algoritmo indexador.

Introduciendo estas llaves en el orden antes mencionado se creará el árbol de la figura 9.1.2.3, en donde podemos observar cómo estan constituídos los nodos del árbol binario con sus correspondientes llaves y apuntadores.

Por supuesto que el árbol anterior es una representación gráfica de cómo se genera el archivo índice.

Para una explicación más detallada del funcionamiento del algoritmo indexador y de cómo quedaría constituído el archivo índice analizemos la tabla de la figura 9.1.2.4, que cuenta con los siguientes parámetros.

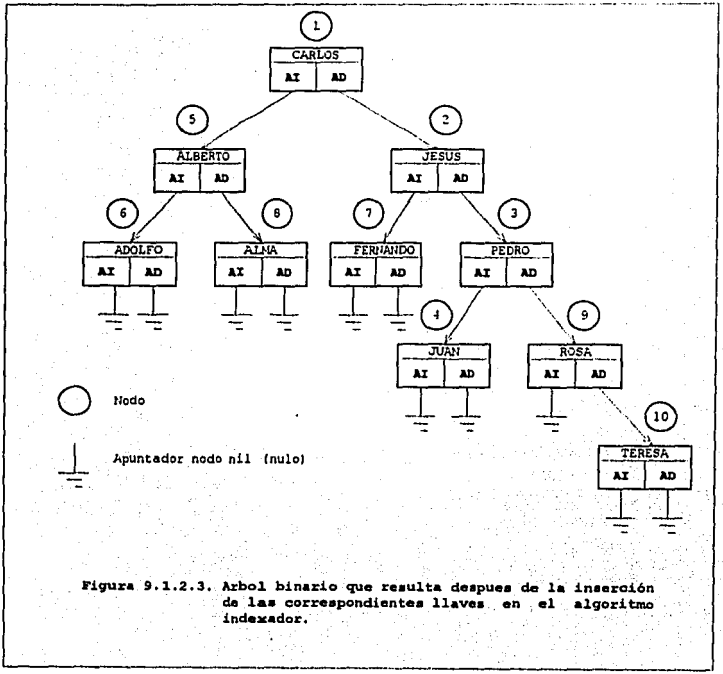


Figura 9.1.2.3. Arbol binario que resulta despues de la inserción de las correspondientes llaves en el algoritmo indexador.

			10	
REG.No	LLAVE	AI	AD	NP
1	CARLOS	0 5	0 2	rait
2	JESUS	0 7	0 3	1
3	PEDRO	0 4	0 9	2
4	JUAN	0	0	3
5	ALBERTO	6	8	1
6	ADOLFO	0	0	5
7	FERNANDO	0	0	2
8	ALMA	0	0	5
9	ROSA	0	10	3
10	TERESA	0	0	9

Contador de registros

X Actualización de Apuntador

AI: Apuntador Izquierdo

AD: Apuntador Derecho

0: Null

NP: Nodo padre

Figura 9.1.2.4. Tabla del archivo índice generado por el indexador.

a) Contador de registros. Se utiliza para llevar un conteo de los registros introducidos, este valor se toma para actualizar los apuntadores izquierdo o derecho.

b) Número de registro. Número asignado para cada registro.

c) Llave. Clave de acceso al archivo.

d) Apuntador izquierdo. Es el apuntador generado a la izquierda del árbol binario.

e) Apuntador derecho. Es el apuntador generado a la derecha del árbol binario.

f) Nodo padre. Nodo padre de la clave de registro.

A continuación se describe cómo se va generando la tabla de la figura 9.1.2.4:

Comenzamos introduciendo la primera llave que es CARLOS, una vez introducidos los parámetros quedan de la siguiente manera:

contador de registros =1

apuntador izquierdo (AI) = 0 (null)

apuntador derecho (AD) = 0 (null)

nodo padre (NP) = raíz

Introduciendo la segunda llave que es JESUS se tiene:

contador de registros = 2

apuntador izquierdo (AI) = 0

apuntador derecho (AD) = 0

nodo padre (NP) = 1

A su vez que se actualiza el apuntador derecho del nodo padre (nodo 1, llave CARLOS) con el valor del contador de registros que en este caso es 2.

Introduciendo la tercera llave que es PEDRO se tiene:

contador de registros = 3

apuntador izquierdo (AI) = 0

apuntador derecho (AD) = 0

nodo padre (NP) = 2

A su vez que se actualiza el apuntador derecho del nodo padre (nodo 2, llave JESUS) con el valor del contador de registros que en este caso es 3.

Introduciendo la cuarta llave que es JUAN se tiene:

contador de registros = 4

apuntador izquierdo (AI) = 0

apuntador derecho (AD) = 0

nodo padre (NP) = 3

A su vez que se actualiza el apuntador izquierdo del nodo padre (nodo 3, llave PEDRO) con el valor del contador de registros que en este caso es 4.

Introduciendo la quinta llave que es ALBERTO se tiene:

contador de registros = 5

apuntador izquierdo (AI) = 0

apuntador derecho (AD) = 0

nodo padre (NP) = 1

A su vez que se actualiza el apuntador izquierdo del nodo padre (nodo 1, llave CARLOS) con el valor del contador de registros que en este caso es 5.

Introduciendo la sexta llave que es ADOLFO se tiene:

contador de registros = 6

apuntador izquierdo (AI) = 0

apuntador derecho (AD) = 0

nodo padre (NP) = 5

A su vez que se actualiza el apuntador izquierdo del nodo padre (nodo 5, llave ALBERTO) con el valor del contador de registros que en este caso es 6.

Introduciendo la septima llave que es FERNANDO se tiene:

contador de registros = 7

apuntador izquierdo (AI) = 0

apuntador derecho (AD) = 0

nodo padre (NP) = 2

A su vez que se actualiza el apuntador izquierdo del nodo padre (nodo 2, llave JESUS) con el valor del contador de registros que en este caso es 7.

Introduciendo la octava llave que es ALMA se tiene:

contador de registros = 8

apuntador izquierdo (AI) = 0

apuntador derecho (AD) = 0

nodo padre (NP) =5

A su vez que se actualiza el apuntador derecho del nodo padre (nodo 5, llave ALBERTO) con el valor del contador de registros que en este caso es 8.

Introduciendo la novena llave que es ROSA se tiene:

contador de registros = 9

apuntador izquierdo (AI) = 0

apuntador derecho (AD) = 0

nodo padre (NP) =3

A su vez que se actualiza el apuntador derecho del nodo padre (nodo 3, llave PEDRO) con el valor del contador de registros que en este caso es 9.

Introduciendo la decima llave que es TERESA se tiene:

contador de registros = 10

apuntador izquierdo (AI) = 0

apuntador derecho (AD) = 0

nodo padre (NP) =9

A su vez que se actualiza el apuntador derecho del nodo padre (nodo 9, llave ROSA) con el valor del contador de registros que en este caso es 10.

Cabe mencionar que las actualizaciones de los apuntadores izquierdo o derecho se hace comparando la llave introducida con la raíz y si la llave es menor, entonces seguimos el recorrido hacia el subárbol izquierdo, en caso de que la llave sea mayor entonces hacemos el recorrido hacia el subárbol derecho.

Una vez introducidas todas las llaves el archivo índice queda generado. Si se introducen más llaves al algoritmo, la tabla se actualizará de acuerdo a los pasos antes mencionados.

A su vez el archivo de datos (.dbf) se puede representar como se muestra en la figura 9.1.2.5.

REG.No	NOMBRE	DIRECCION	TELEFONO
1	CARLOS	COLIMA 162	710-8773
2	JESUS	ROMA 42	584-1919
3	PEDRO	JUAREZ 15	435-1212
4	JUAN	PRADOS 21	710-2323
5	ALBERTO	BAJIO 184	678-1215
6	ADOLFO	CERRADA 3	562-1023
7	FERNANDO	COLIMA 6	777-7462
8	ALMA	MADERO 101	234-1278
9	ROSA	SANTOS 40	945-5666
10	TERESA	TABASCO 45	358-4202

Figura 9.1.2.5. Archivo de datos (.dbf) generado.

El diagrama de flujo general del algoritmo indexador se puede ver en la figura 9.1.2.6.

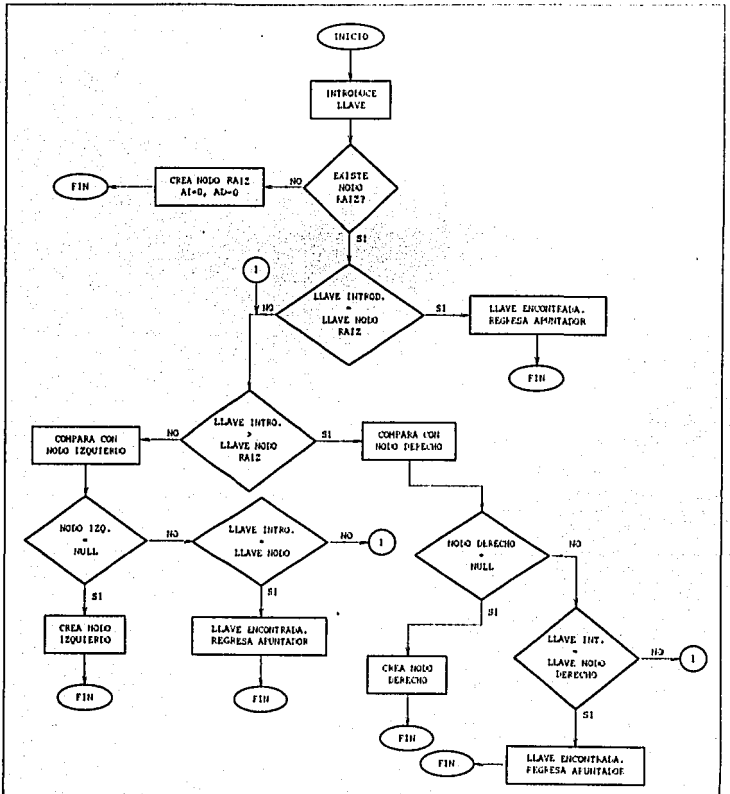


Figura 9.1.2.6. Diagrama general del algoritmo indexador.

En seguida se presenta el programa fuente de esta herramienta:

```
/* DECLARACION DE INCLUDE'S */

#include "stdio.h"
#include "conio.h"
#include "string.h"


/* VARIABLES UTILIZADAS POR ESTE MODULO */

int cont_nod,
    apuntador,
    apuntcero = 0,
extern int   ban_cons_modi;
extern char archivo_indice [20],


/* PROCEDIMIENTO QUE RECIBE LA LLAVE */

introduce (llave,modo)
char llave [40];
int   modo;
{
        char complemento [39] = "                                    ";
        apuntador              = 1;
        cont_nod               = 0;
        strncat (llave,complemento,strlen(complemento)-strlen(llave));

        /* CREA LA PRIMERA ENTRADA << RAIZ >> */
        arbol (llave,modo);
}


/* PROCEDIMIENTO QUE DECIDE SI LA LLAVE YA EXISTE O ES NECESARIO INSERTARLA */
/* EN EL ARBOL                                                              */

int arbol (llave,modo)
char llave [40];
int   modo;
{
        int         ai,
                    ad;
        char buf  [40],
                    cont [48],
                    ail   [4],
                    aid   [4];
        FILE *index;

        strncpy (buf, " ",40);          /* BUFFER LEIDO DEL ARCHIVO */
        strncpy (cont," ",48);/* CONTADOR EN EL ARCHIVO */
        strncpy (ail,  " ",4);          /* APUNTADOR IZQUIERDO EN EL ARCHIVO */
        strncpy (aid,  " ",4);          /* APUNTADOR DERECHO EN EL ARCHIVO */
        if ((index = fopen(archivo_indice,"a+")) == NULL) {
                printf ("\a *****Error al abrir el archivo ******");
                exit (1);
        }

        /* COMPRUEBA SI ESTA VACIO EL ARCHIVO  SI LO */
        /* ESTA CREA LA RAIZ CON LA PRIMERA ENTRADA */
```

```
        if (fread(buf,sizeof(char),strlen(llave),index) == 0) {
                ai = 0;                                  /* APUNTADOR A LA IZQUIERDA */
                ad = 0,                                  /* APUNTADOR A LA DERECHA */
                cont_nod = cont_nod+1;                   /* CONTADOR DE ELEMENTOS DEL ARCHIVO*/
                itoa (cont_nod,cont,10);
                itoa (ai,aii,10);
                itoa (ad,aid,10);
                if (modo==3) {
                        fprintf (index,"%-48s\n",cont);
                        fprintf (index,"%-40s%-4s%-4s\n",llave,aii,aid);
                }
                fclose (index);
                return;
        }
        fclose (index);
        if (modo==3) {
                crea_cont();
        }
        do {
                inserta (llave,modo);
        } while (apuntcero!=0);
}

/* FUNCION QUE INCREMENTA EL CONTADOR EN CADA ENTRADA DE LLAVE */

crea_cont ()
{
        FILE *indice;
        char cont [48];

        strncpy(cont," ",48);
        if (( indice=fopen(archivo_indice,"r+"))==NULL) {
                printf ("\a ***** Error al abrir el archivo ******");
                exit (1);
        }
        fseek (indice,0,0);
        fread (cont,sizeof(char),4,indice);
        cont_nod = atoi(cont);
        cont_nod = cont_nod+1;
        itoa (cont_nod,cont,10);
        fseek (indice,0,0);
        fprintf (indice,"%-48s\n",cont);
        fclose (indice);
}

/* FUNCION QUE DECREMENTA EL CONTADOR CUANDO LAS LLAVES SON IGUALES */

crea_cont2 ()
{
        FILE *indice;
        char cont [50];

        strncpy (cont," ",48);
        if ((indice=fopen(archivo_indice,"r+"))==NULL) {
                printf ("\a ***** Error al abrir el archivo ******");
                exit (1);
        }
        fseek (indice,0,0);
        fread (cont,sizeof(char),4,indice);
        cont_nod = atoi(cont);
        cont_nod = cont_nod+-1;
        itoa (cont_nod,cont,10);
        fseek (indice,0,0);
        fprintf (indice,"%-48s\n",cont);
```

```
        fclose (indice);
}


/* LEE EL ARCHIVO HASTA ENCONTRAR UN NULL EN APUNTADOR IZQUIERDO O DERECHO */

inserta (clve,modo)
char clve [40];
int   modo;
{
        int   ail                       /* APUNTADOR IZQUIERDO */
              aid;                      /* APUNTADOR DERECHO */
        char buffer   [40],
                      llaveint  [40],
                      llaveint  [40],
                      ai        [4]
                      ad        [4];
        FILE *indice;

        strncpy (ai,    "",4);
        strncpy (ad,    "",4);
        strncpy (buffer,"",40);         /* BUFFER EN DONDE SE ALMACENA LA CLAVE LEIDA DEL ARCHIVO */
        if ((indice=fopen(archivo_indice,"r+"))==NULL) {
                printf ("\a ***** Error al abrir el archivo ******");
                exit (1);
        }
        fseek (indice,50*apuntador,0);          /* SE POSICIONA EN LA LLAVE DE CADA REGISTRO DEL ARCHIVO */
        fread (buffer,sizeof(char),(strlen(clve)),indice);      /* LEE LA LLAVE DEL REGISTRO */

        /* AQUI ES DONDE SE HACEN LAS COMPARACIONES DE LA LLAVE INTRODUCIDA */
        /* Y LA LEIDA DEL ARCHIVO */
        if (strcmp(clve,buffer) == 0) {         /* LAS LLAVES SON IGUALES*/
                fclose (indice);
                if (modo == 3) {
                        crea_con2 (),   /* DECREMENTA EL CONTADOR */
                }
                if (ban_cons_modi == 0) consulta (),
                if (ban_cons_modi == 1) consulta_mod ();
        }
        if (strcmp(clve,buffer) < 0) {          /* LA LLAVE INTRODUCIDA ES MENOR QUE LA CLAVE DEL REGISTRO*/
                fclose (indice);
                menor (clve,cont_nod,modo),     /* FUNCION QUE EVALUA EL APUNTADOR IZQUIERDO */
                return;
        }
        if (strcmp(clve,buffer) > 0) {          /* LA LLAVE INTRODUCIDA ES MAYOR QUE LA LLAVE DEL REGISTRO */
                fclose (indice);
                mayor (clve,cont_nod,modo);     /* FUNCION QUE EVALUA EL APUNTADOR DERECHO */
                return;
        }
        fclose (indice);
}


/* FUNCION QUE INSERTA NODOS CUANDO LA LLAVE ES MENOR A LAS QUE YA EXISTEN */

menor (llave,contador,modo)
char llave [40],
int     contador,
        modo;
{
        int   apuntiz  = 0,
              apuntder = 0;
        char api[4],
                      apd  [4],
                      cont [4],
```

```
        FILE *indice;

        strncpy(api, " ",4);            /*APUNTADOR IZQUIERDO */
        strncpy(cont," ",4);            /*CONTADOR */
        strncpy(apd," ",4);             /*APUNTADOR DERECHO */
        if ((indice=fopen(archivo_indice,"r+"))==NULL) {
                printf ("\a ***** Error al abrir el archivo *****");
                exit (1);
        }
        fseek (indice,(50*apuntador)+40,0);     /* POSICIONA EN EL APUNTADOR IZQUIERDO */
        fread (api,sizeof(char),4,indice),      /* LEE EL APUNTADOR IZQUIERDO */
        apuntiz = atoi(api);
        if (apuntiz == 0) {                     /* EVALUA SI EL APUNTADORIZQUIERDO ES NULL Y CREA UN NUEVO NODO */
                itoa (contador,cont,10);
                fseek (indice,(50*apuntador)+40,0);
                if (modo == 3) {
                        fprintf (indice,"%-4s",cont);
                }
                fseek (indice,50*contador,0);
                itoa (apuntiz,api,10),
                itoa (apuntder,apd,10);
                if (modo == 3) {
                        fprintf (indice,"%-40s%-4s%-4s%-4s\n",llave,api,apd);
                }
                fclose (indice);
                apuntcero = apuntiz;
        }
        else {                  /* SI NO ES NULL BUSCA UN SIGUIENTE REGISTRO */
                fclose (indice);
                apuntador = apuntiz;
                inserta (llave,modo);
        }
}

/* FUNCION QUE INSERTA NODOS CUANDO LA LLAVE ES MAYOR A LAS QUE YA EXISTEN */

mayor (llave,contador,modo)
char llave [40];
int     contador,
        modo;
{
        int   apuntiz  = 0,
              apuntder = 0;
        char api  [4],
             apd  [4],
             cont [4];
        FILE *indice;

        strncpy(api, " ",4);
        strncpy(apd, " ",4);
        strncpy(cont," ",4);
        if ((indice=fopen(archivo_indice,"r+"))==NULL) {
                printf ("\a ***** Error al abrir el archivo *****");
                exit (1);
        }
        fseek (indice,(50*apuntador)+44,0);
        fread (apd,sizeof(char),4,indice);
        apuntder = atoi(apd);
        if (apuntder == 0) {
                itoa (contador,cont,10);
                fseek (indice,(50*apuntador)+44,0);
                if (modo == 3) {
                        fprintf (indice,"%-4s",cont);
                }
                fseek (indice,50*contador,0);
```

```
                itoa (apuntiz,api,10);
                itoa (apuntder,apd,10);
                if (modo == 3) {
                        fprintf (indice,"%-40s%-4s%-4s\n",llave,api,apd);
                }
                fclose (indice);
                apuntcero = apuntder;
        }
        else {
                fclose (indice);
                apuntador = apuntder;
                inserta (llave,modo);
        }
}
```

IX.2. ANALISIS

IX.2.1. FUNCIONAMIENTO DEL SISTEMA Y MANEJO DE INFORMACION

Este sistema a pesar de contar con Altas, Bajas, Cambios y Consultas, el menú no visualiza estas opciones dentro de la pantalla de captura del usuario, sino que sólo presenta un menú con las opciones de Mantenimiento y Consultas.

Dentro de la opción de Mantenimiento se realizan los procesos de Altas, Bajas y Cambios; mientras que en la opción de Consultas sólo se lleva a cabo ese proceso, el de Consulta.

La manera en que funciona la opción de Mantenimiento para elegir qué proceso de los mencionados arriba se va a ejecutar se explica a continuación. Pero cabe señalar que esto se lleva a cabo con la interacción de los algoritmos de captura e indexación.

Pues bien, cuando se selecciona la opción de Mantenimiento y se captura una llave que no existe en la base de datos que se está trabajando, automáticamente el sistema estará en el proceso de Altas. Cuando la llave capturada ya existe en la base de datos, el sistema presenta la información correspondiente a esa llave y estará en el proceso de Cambios. O bien el registro se podrá dar de baja.

Por otro lado, cuando se selecciona la opción de Consultas, la llave capturada deberá existir en la base de datos, de lo contrario no se presentará información alguna.

En seguida se da una breve explicación de la manera en que funciona cada uno de los procesos que conforman el sistema.

Todos los campos que son extraídos de otras bases de datos diferentes a las que se capturan en determinado momento sólo se presentan como información al usuario. Estos datos no se graban a la base de datos en cuestión para evitar la repetición de información.

Cabe mencionar también que todas las bajas son solamente lógicas.

1) PROGRAMACION DE VUELOS (Llave = Número de vuelo).

1.1) ALTAS.- Primeramente se solicita al usuario el número de vuelo y a continuación la matrícula del avión que será asignado al vuelo capturado, con ella se accesará a la base de datos de definición de aviones y extraerá el campo de capacidad.

Posteriormente se captura el origen/destino con el cual se accesa a la base de datos de definición de escalas por origen/destino y se extrae el campo de escalas.

A continuación se capturan la fecha y hora de salida del vuelo.

En el proceso de asignación de tripulación se debe considerar lo siguiente: Al capturar la clave de cada empleado el primer caracter debe corresponder a la primera inicial de la jerarquía del mismo, siendo para pilotos la letra P, para copilotos la letra C y para azafatas la letra A. Lo anterior es para identificar qué tipo de empleado se asignará al vuelo y para que no haya errores en la captura, se valida que el primer caracter de cada uno de los campos solicitados solo sea alguno de los mencionados arriba. Cuando se captura la clave del empleado, con ella se accesa la base de datos de definición de tripulación y se extrae su nombre.

Para que se pueda llevar a cabo la asignación de un avión y de su tripulación, el campo de disponibilidad de cada uno de ellos deberá estar en "S" y cuando se salve el registro este campo deberá cambiar a "N".

1.2) BAJAS.- Mediante la llave de acceso se localiza el registro a ser eliminado, cuando esto sucede los campos de disponibilidad de las bases de datos de la definición de aviones y de tripulación cambia a "S".

1.3) CAMBIOS.- Mediante la llave de acceso se localiza al registro deseado y se hacen las modificaciones necesarias. Si se cambian el avión o la tripulación asignada al vuelo los campos de disponibilidad de las bases de datos cambiaran a "S".

1.4) CONSULTAS.- Dependiendo de la llave de acceso se presentará la información solicitada.

2) RESERVACION DE VUELO (Llave = Número de vuelo + Nombre del cliente).

2.1) ALTAS.- En este proceso se captura primero el número de vuelo y con él se accesa a la base de datos de programación de vuelos y se extraen los campos de origen/destino, fecha de salida y hora de salida. Con el campo de origen/destino se accesa a la base de datos de escalas por origen destino y se extrae el campo de escalas.

A continuación se captura la clase a la que se desea hacer la reservación (para que no haya errores de captura se valida que sólo sean los caracteres A, B o C, para clase A, B o C respectivamente). Una vez capturada la clase, con el número de vuelo y la clase se barre la base de datos de reservaciones para encontrar cuáles asientos de esa clase y ese vuelo están disponibles y cuáles han sido ya reservados (se presenta un gráfico de asientos que muestra esta información). Con los campos de origen/destino y clase se accesa la base de datos de itinerario y costo por clase de vuelo y se extrae el costo del pasaje y solo con la clase se accesa la base de datos de servicios por clase de vuelo y se extraen los campos de los servicios que la compañía proporciona para esa clase.

En seguida se captura el número de asiento y aunque ya se esta mostrando el gráfico de asientos para indicar cuáles están ya ocupados, se verifica que el número de asiento capturado no esté ya ocupado.

Por último se capturan el nombre, el teléfono y la dirección del cliente.

2.2) BAJAS.- Con la llave se accesa a la base de datos de reservaciones y el registro se da de baja lógica.

2.3) CAMBIOS.- Mediante la llave de acceso se podrá editar el registro deseado y de este modo hacer los cambios necesarios.

2.4) CONSULTAS.- Con la llave se presenta la información correspondiente.

3) DEFINICION DE AVIONES (Llave = Matricula).

3.1) ALTAS.- En este proceso se capturan las características de cada avión (Matricula, Modelo y Capacidad. La disponibilidad es creada por default y será "S") con que cuenta la aerolínea. Además da la facilidad de capturar las de los aviones que se adquieran posteriormente.

3.2) BAJAS.- Mediante la llave de acceso a la base de datos, se dará de baja el registro que no se desee.

3.3) CAMBIOS.- Mediante la llave de acceso, se harán los cambios necesarios a el registro elegido.

3.4) CONSULTAS.- Mediante la llave de acceso se permitirá la visualización de la información del registro elegido.

4) DEFINICION DE TRIPULACION (Llave = Clave de empleado).

Se requiere que se haga una validación del primer caracter de la llave, mediante ese caracter se identificará de qué tipo de empleado se trata (Piloto, Copiloto o Azafata). El primer caracter sólo podrá ser P, C o A, respectivamente para Piloto, Copiloto o Azafata.

4.1) ALTAS.- Se capturan clave y nombre del empleado. La disponibilidad es creada por default y será "S".

4.2) BAJAS.- Con la llave de acceso se localiza el registro que se da de baja.

4.3) CAMBIOS.- Con la llave de acceso se permitirá la localización del registro que se desea modificar.

4.4) CONSULTAS.- Mediante la llave de acceso se visualiza la información que contenga el registro.

5) DEFINICION DE ESCALAS POR ORIGEN/DESTINO (Llave = Origen/Destino).

5.1) ALTAS.- Se capturan origen/destino y escalas.

5.2) BAJAS.- Con la llave de acceso se localiza el registro no deseado y se da de baja.

5.3) CAMBIOS.- Mediante la llave se accesa al registro y se realizan los cambios en el registro que se quiera modificar.

5.4) CONSULTAS.- Mediante la llave de acceso se consulta la información que contiene el registro.

6) DEFINICION DE ITINERARIOS Y COSTO POR CLASE DE VUELOS (Llave = Origen/Destino + Clase).

Se valida que la clase sólo sea A, B o C.

6.1) ALTAS.- Se capturan origen/destino, clase y costo.

6.2) BAJAS.- Con la llave se localiza el registro que será borrado de la base de datos.

6.3) CAMBIOS.- Mediante la llave de acceso se realizan los cambios al registro que se desee.

6.4) CONSULTAS.- Mediante la llave de acceso se localiza el registro que se desea consultar.

7) DEFINICION DE SERVICIOS POR CLASE DE VUELOS (Llave = Clase de vuelo).

7.1) ALTAS.- En este módulo se capturan los servicios que se proporcionan de acuerdo a la clase del vuelo. Para indicar de qué clase de vuelo se trata se hará con "A", "B" y "C", los cuales indican premier, primera y segunda clase respectivamente.

7.2) BAJAS.- Mediante la llave de acceso se da de baja el registro no deseado.

7.3) CAMBIOS.- Mediante la llave de acceso se posicionará el apuntador en el registro correspondiente a la llave, teniendo la posibilidad de modificar los campos que sean necesarios.

7.4) CONSULTAS.- Proporcionando la llave de acceso se visualizaran los servicios que se proporcionan para cada clase de vuelo.

8) GRAFICO DE ASIENTOS.

Dependiendo de la clase elegida, se presenta un gráfico con la localización de los asientos y cuáles han sido ya reservados.

9) ACTUALIZACIONES AUTOMATICAS.

Se refiere a hacer las actualizaciones necesarias en los campos que requieran modificarse, esto sin necesidad de correr un proceso por separado.

IX.3. DISEÑO

Para el diseño del sistema nos basamos en el **Modelo relacional**, porque después de haber estudiado los diferentes modelos, consideramos que es de fácil implementación, existe poca repetición de información y proporciona un medio útil para el manejo de las bases de datos.

IX.3.1. DIAGRAMAS DE FLUJO DE DATOS

A continuación se presentan los diagramas de flujo de datos correspondientes a los procesos implicados en el sistema de *Programación de vuelos y Reservaciones.*

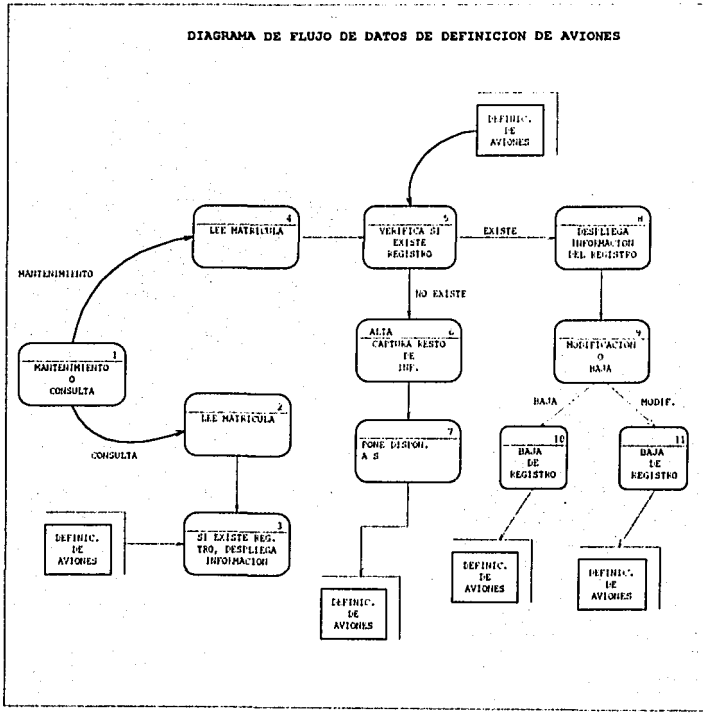


DIAGRAMA DE FLUJO DE DATOS DE DEFINICION DE ESCALAS POR ORIGEN / DESTINO

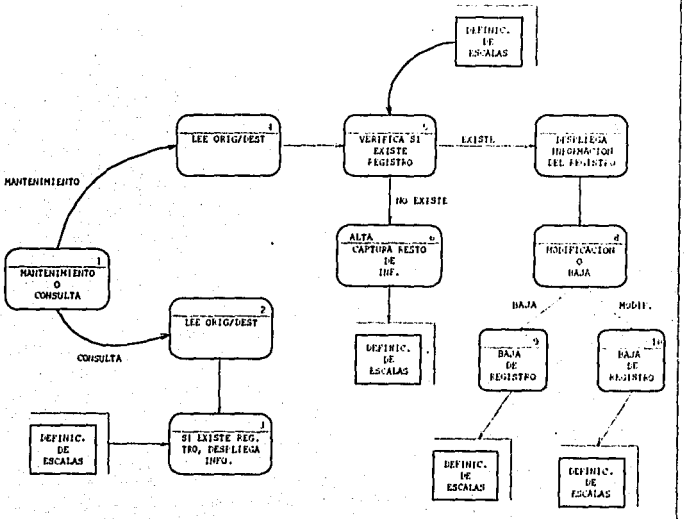


DIAGRAMA DE FLUJO DE FLUJO DE DATOS DE DEFINICION DE TRIPULACION

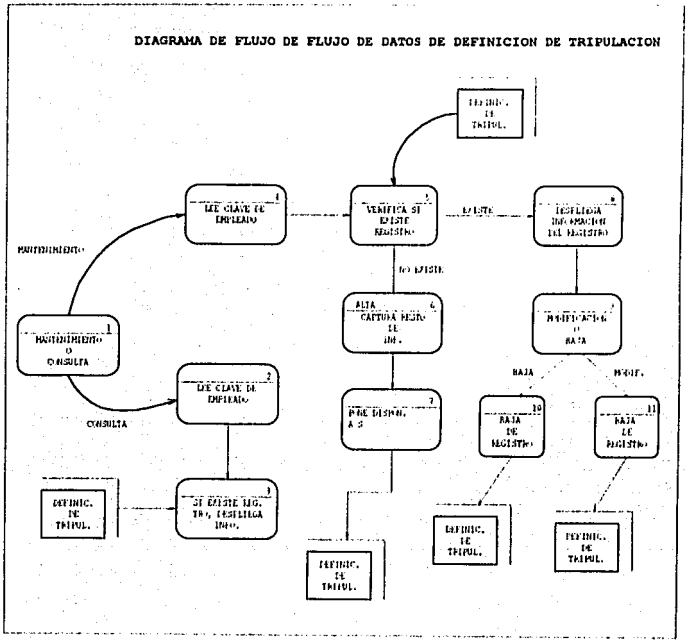
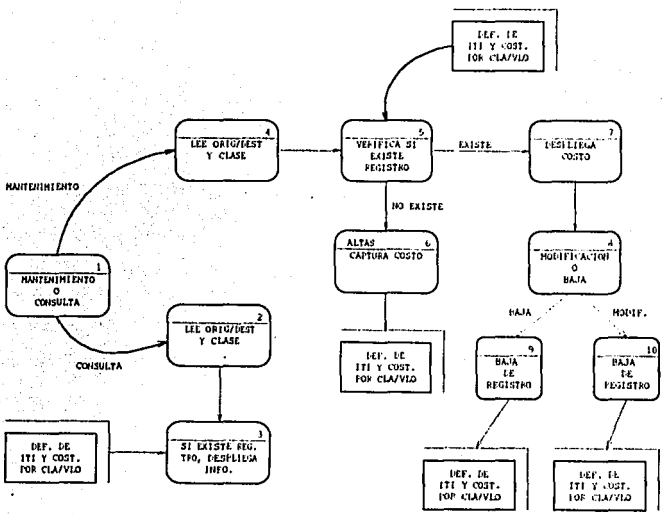
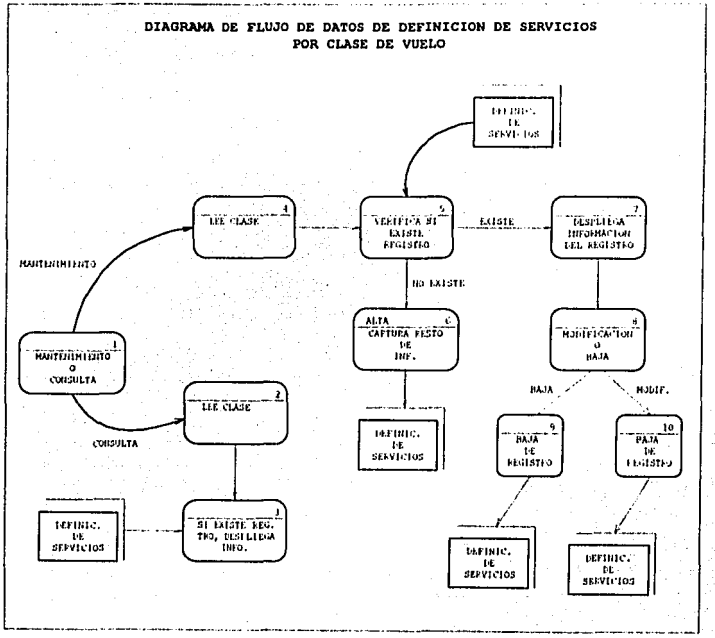


DIAGRAMA DE FLUJO DE DATOS DE DEFINICION DE ITINERARIO Y COSTO POR CLASE DE VUELO





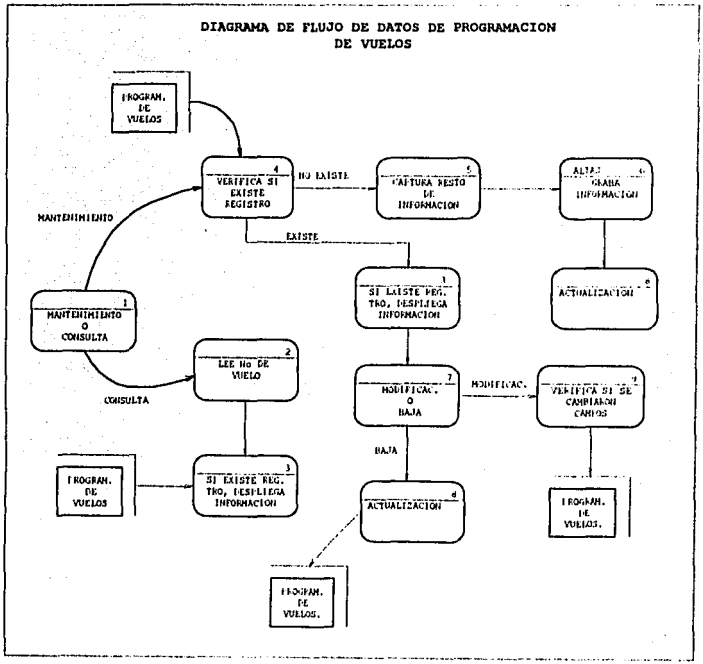


DIAGRAMA DE FLUJO DE DATOS DE PROGRAMACION DE VUELOS (CONTINUACION)

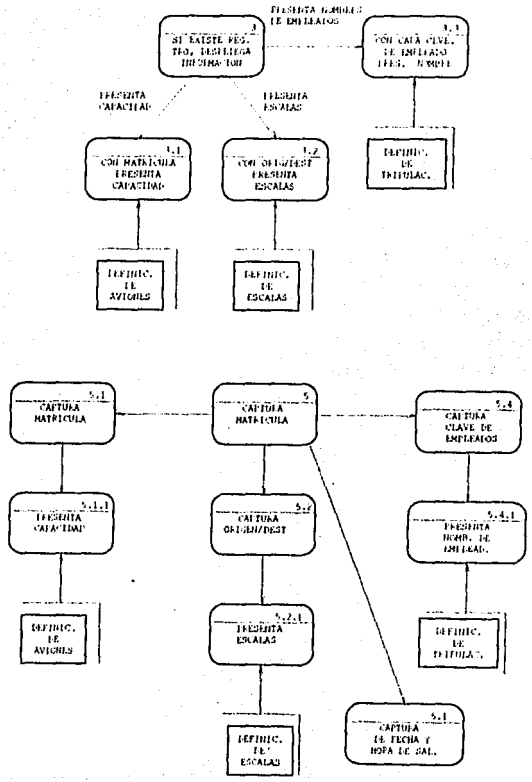
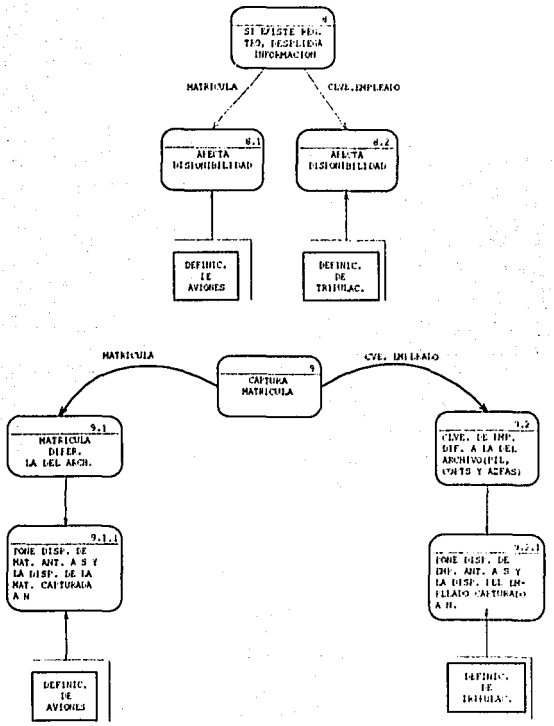


DIAGRAMA DE FLUJO DE DATOS DE PROGRAMACION
DE VUELOS (CONTINUACION)



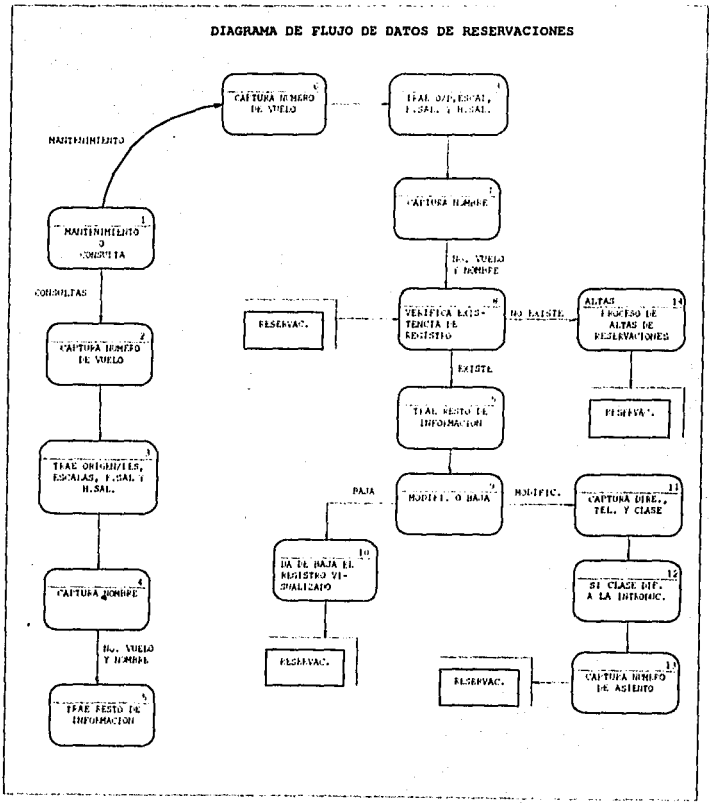


DIAGRAMA DE FLUJO DE DATOS DE RESERVACIONES
(CONTINUACION)

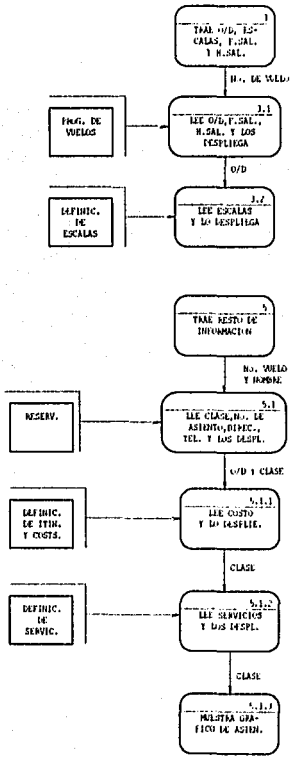
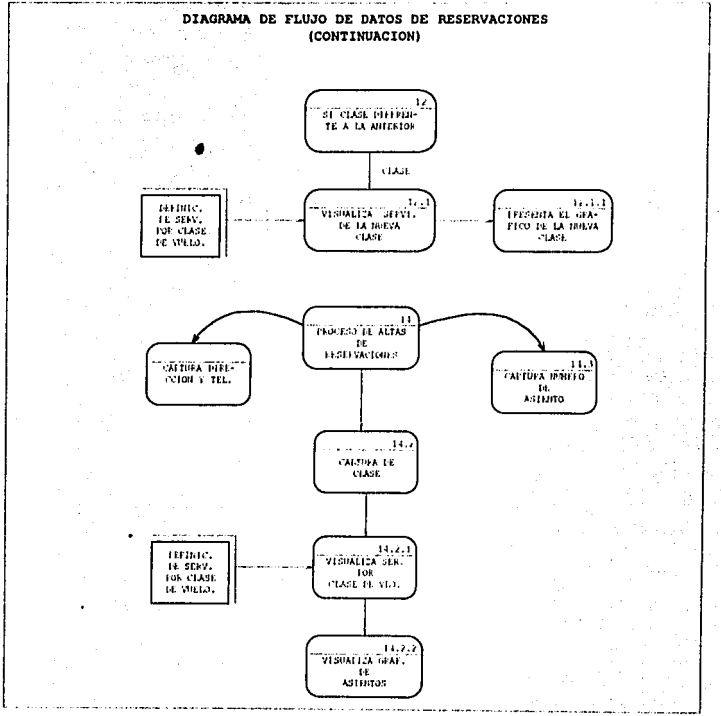
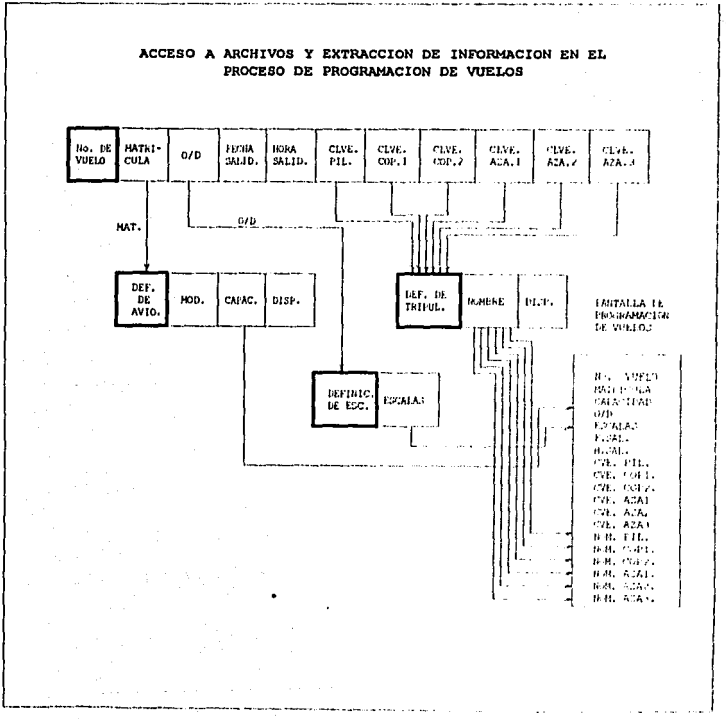


DIAGRAMA DE FLUJO DE DATOS DE RESERVACIONES (CONTINUACION)

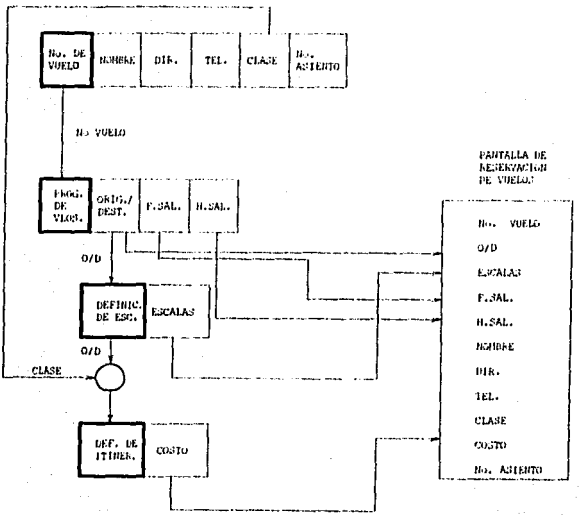


IX.3.2. DIAGRAMAS DE ACCESO A TABLAS

A continuación se presentan los diagramas de acceso y extración de información de los procesos de *Programación de vuelos y Reservaciones.*



ACCESO A ARCHIVOS Y EXTRACCION DE INFORMACION EN EL PROCESO DE RESERVACION DE VUELOS



IX.3.3. DICCIONARIO DE DATOS

PROGRAMACION DE VUELOS			
NOMBRE DEL CAMPO	TIPO	LONGITUD	DESCRIPCION
No DE VUELO	NUM	5	NUMERO DEL VUELO
MATRICULA	CHAR	6	MATRICULA DEL AVION
ORIGEN / DESTINO	CHAR	6	ORIGEN Y DESTINO DEL VUELO
FECHA SALIDA	FECHA	6	FECHA DE SALIDA DEL VUELO
HORA SALIDA	HORA	4	HORA DE SALIDA DEL VUELO
CLAVE PILOTO	CHAR	4	CLAVE DEL PILOTO
CLAVE COPILOTO 1	CHAR	4	CLAVE DEL COPILOTO 1
CLAVE COPILOTO 2	CHAR	4	CLAVE DEL COPILOTO 2
CLAVE AZAFATA 1	CHAR	4	CLAVE DE LA AZAFATA 1
CLAVE AZAFATA 2	CHAR	4	CLAVE DE LA AZAFATA 2
CLAVE AZAFATA 3	CHAR	4	CLAVE DE LA AZAFATA 3
RESERVACION DE VUELOS			
NOMBRE DEL CAMPO	TIPO	LONGITUD	DESCRIPCION
No DE VUELO	NUM	5	NUMERO DEL VUELO
CLASE	CHAR	1	CLASE DEL VUELO
No DE ASIENTO	CHAR	3	NUMERO DE ASIENTO
NOMBRE	CHAR	40	NOMBRE DEL CLIENTE
TELEFONO	CHAR	10	TELEFONO DEL CLIENTE

DIRECCION	CHAR	40	DIRECCION DEL CLIENTE
DEFINICION DE AVIONES			
NOMBRE DEL CAMPO	TIPO	LONGITUD	DESCRIPCION
MATRICULA	CHAR	6	MATRICULA DEL AVION
MODELO	CHAR	10	MODELO DEL AVION
CAPACIDAD	NUM	3	CAPACIDAD DEL AVION
DISPONIBILIDAD	CHAR	1	DICPONIBILIDAD DEL AVION
DEFINICION DE ESCALAS POR ORIGEN / DESTINO			
NOMBRE DEL CAMPO	TIPO	LONGITUD	DESCRIPCION
ORIGEN / DESTINO	CHAR	6	ORIGEN Y DESTINO DEL VUELO
ESCALAS	CHAR	40	ESCALAS QUE HACE EL VUELO
DEFINICION DE TRIPULACION			
NOMBRE DEL CAMPO	TIPO	LONGITUD	DESCRIPCION
CLAVE EMPLEADO	CHAR	4	CLAVE DEL EMPLEADO
NOMBRE EMPLEADO	CHAR	40	NOMBRE DEL EMPLEADO
DISPONIBILIDAD	CHAR	1	DISPONIBILIDAD DEL EMPLEADO
DEFINICION DE ITNERARIO Y COSTO POR CLASE DE VUELO			
NOMBRE DEL CAMPO	TIPO	LONGITUD	DESCRIPCION
ORIGEN / DESTINO	CHAR	6	ORIGEN Y DESTINO DEL VUELO

CLASE	CHAR	1	CLASE DEL VUELO
COSTO	NUM	4	COSTO DEL VUELO
DEFINICION DE SERVICIOS POR CLASE DE VUELO			
NOMBRE DEL CAMPO	TIPO	LONGITUD	DESCRIPCION
CLASE	CHAR	1	CLASE DEL VUELO
SERVICIO 1	CHAR	35	SERVICIO 1 PRESTADO
DEFINICION DE SERVICIOS POR CLASE DE VUELO			
NOMBRE DEL CAMPO	TIPO	LONGITUD	DESCRIPCION
SERVICIO 2	CHAR	35	SERVICIO 2 PRESTADO
SERVICIO 3	CHAR	35	SERVICIO 3 PRESTADO
SERVICIO 4	CHAR	35	SERVICIO 4 PRESTADO
SERVICIO 5	CHAR	35	SERVICIO 5 PRESTADO

CONCLUSIONES

Al finalizar el presente trabajo de tesis, se obtuvieron las siguientes conclusiones:

1) Se logró establecer la arquitectura de un sistema de base de datos.

2) Después de un análisis de los diferentes métodos de acceso, consideramos que el algoritmo de árbol binario es el más apropiado por su bajo grado de complejidad, la no existencia de colisiones y su rapidéz (menor que la del algoritmo hash) de acceso.

3) Consideramos que la herramienta desarrollada para la captura, facilita la introducción de información a los campos de una base de datos, ya que cuenta con validaciones de tipo y navegación entre campos. Por lo que el usuario tiene la facilidad de verificar que la información capturada sea correcta antes de grabarla a disco.

4) Consideramos que el trabajo en su conjunto puede proporcionar una herramienta útil tanto a los profesores como a los alumnos de la asignatura de ***Bases de Datos***. Ya que se abordaron la mayoría de los temas de más importancia en la asignatura antes mencionada.

BIBLIOGRAFIA

Dr. Dobb's Journal
Noviembre 1994.

GEORGE, M. Scott, Principios de información.
Mc Graw Hill,1990.

HENRY F. Korth, Fundamentos de base de datos.
Mc Graw Hill, 1990.

R. Frost, Bases de datos y sistemas expertos.
Díaz de Santos S.A., 1989.

ROBERT L. Kruse, Estructura de datos y diseño de programas.
Prentice Hall, 1988.

ROBERT J. Verzello, Procesamiento de datos.
Mc Graw Hill, 1986.

RONALD A. Radice, Software engineering an industrial approach.
Prentice Hall, 1989.

SHAKUNTALA Atre, Técnicas de bases de datos.

Trillas, 1990.

I.T. Hawryszkiewycz, Database analysis and design.

Macmillan Publishing Company, 1991.