UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

COLEGIO DE CIENCIAS Y HUMANIDADES
UNIDAD ACADEMICA DE LOS CICLOS
PROFESIONAL Y DE POSGRADO

Un Sistema de Administración de Bases de Datos en ANSI C

Tesis que para obtener el grado de Maestro en Ciencias de la Computación

P R E S E N T A :

Jorge E. Rodríguez Buenfil

Ciudad Universitaria, D.F. 1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Con un agradecimiento muy especial
para las siguientes personas:

Dr. David Rosenblueth Laguette
Por su invaluable dirección y asesoría
en la elaboración de este trabajo.

M. en C. Gloria Quintanilla Osorio
Por su ayuda y aliento constante.

Dra. Hanna Oktaba
Por su motivación y apoyo
para terminar la tesis.

Dra. Ofelia Cervantes Villagrán
Por su experiencia y dedicación
por mejorar este trabajo.

M. en C. Mónica Ardisson
Por sus valiosas sugerencias
y comentarios.

A Eugenia

Por su comprensión, amor y apoyo.

A Audrey

Por la esperanza que ella representa.

A mis padres

Sin cuyo respaldo y amor
nunca hubiera podido realizar mis estudios.

A mis maestros, compañeros y amigos

Por su guía, ayuda y enseñanzas.

Contenido

Introducción

En los últimos quince años se ha podido observar la creciente utilización de computadoras personales para la administración de microempresas, así como de empresas pequeñas que tradicionalmente habían llevado a cabo sus tareas administrativas de forma manual, con la lentitud y los frecuentes errores y omisiones que son característicos de la administración manual de las empresas.

En la actualidad, en la mayoría de las ciudades del país se manifiesta la necesidad de apoyar a las micro y pequeñas empresas a desarrollar sistemas de cómputo que les apoyen en su administración. Es así que el desarrollo de sistemas administrativos y contables adquiere relevancia práctica debido a que hay una necesidad social de los mismos para ayudar a las empresas que los requieren a mantenerse competitivas en un entorno cada vez más demandante debido al ingreso de competidores internacionales sumamente organizados y agresivos comercialmente gracias al Tratado de Libre Comercio (TLC).

SABAD, un sistema de administración de bases de datos en ANSI C, surge precisamente como respuesta a la necesidad de contar con una herramienta para desarrollar aplicaciones

Resumen

SABAD es un sistema de administración de bases de datos capaz de funcionar como herramienta de desarrollo de aplicaciones, especialmente aquellas administrativas basadas en la manipulación de datos tales como inventarios, nómina, contabilidad, etc.

SABAD está desarrollado en el lenguaje de programación C, según está definido por el estándar ANSI, y es capaz de funcionar en prácticamente cualquier plataforma personal de cómputo, debido a la facilidad de transportabilidad que proporciona ANSI C, al existir compiladores para dicho lenguaje en prácticamente cualquier tipo de computadora.

administrativas para el ambiente de computadoras personales bajo el sistema operativo MS-DOS[1].

Aunque existen herramientas comerciales para el desarrollo de tales aplicaciones, éstas no siempre son satisfactorias debido principalmente a que:

1) Se debe pagar constantemente a las empresas extranjeras fabricantes del software[2] por sus actualizaciones y, en algunos casos, regalías por cada aplicación que se venda por parte del desarrollador, lo cual para México implica una fuga de divisas.

2) La poca flexibilidad de modificación de las herramientas debido a que muchos detalles de su implantación se ocultan y se manejan como secretos comerciales, lo cual genera una dependencia tecnológica al estar utilizando sistemas de manejo de bases de datos sobre cuyo desempeño se tiene poco o ningún control.

3) Generalmente el código resultante del desarrollo de las aplicaciones es muy ineficiente en cuanto al tiempo de ejecución y el espacio que ocupa en disco y en memoria principal.

---

1 MS-DOS son las siglas de Microsoft Disk Operating System, el sistema operativo más común en las computadoras personales compatibles con el estándar de computadora personal de I.B.M. (International Business Machines).

2 Software, a falta de una palabra más castiza, se emplea en esta tesis para describir la secuencia de instrucciones y los datos de los programas de cómputo.

SABAD soluciona todos los problemas anteriores al ser el resultado de un desarrollo nacional, cuyo código fuente está disponible para el que lo solicite sin regalías ni costo de ningún tipo; también es fácilmente modificable y extensible por cualquier programador en lenguaje C que lo desee debido a su estructura modular. Por otro lado, las aplicaciones desarrolladas con *SABAD* producen un código ejecutable especialmente veloz y compacto en cuanto a su tamaño en bytes.

Esta tesis está estructurada en torno de cuatro capítulos.

En el **primer capítulo** se presenta el marco teórico en el cual se fundamenta el diseño de *SABAD*, el modelo relacional de bases de datos así como los conceptos más importantes que son necesarios para comprender la exposición de los capítulos siguientes.

El **segundo capítulo** presenta el diseño de *SABAD*, destacando las aportaciones que se hicieron al estado del arte en cuanto a sistemas manejadores de bases de datos basados en computadoras personales.

En el **capítulo tres** se muestran los detalles de la implantación del diseño anterior en código C estándar, con énfasis en las diferentes funciones y módulos que se tuvo que desarrollar para lograr la funcionalidad del diseño de *SABAD* y llevar a la práctica los conceptos de diseño original que constituyen las aportaciones de este trabajo en el ámbito técnico del manejo de aplicaciones de bases de datos.

El **capítulo cuatro** se dedica a exponer los aspectos funcionales de *SABAD* y las facilidades que ofrece a los desarrolladores de aplicaciones. También se presenta un ejemplo de una aplicación desarrollada con el beneficio de toda la funcionalidad de *SABAD*.

Por último se presenta una evaluación del trabajo realizado y se sugieren algunas extensiones posibles.

1

Fundamentos de Bases de Datos y Modelo Relacional

RESUMEN DEL CAPITULO

En un principio los datos se guardaban en archivos que no tenían ninguna posibilidad de interactuar entre sí; no se hablaba de *bases de datos* sino de *sistemas de archivos*.

Antes del advenimiento de las bases de datos, en la mayoría de los sistemas de computación resultaba muy difícil modificar el modo de utilizar los datos. Cada programador veía los datos a su modo y quería siempre modificarlos a medida que variaban sus necesidades. Sin embargo, cualquier modificación era capaz (potencialmente) de desatar una verdadera reacción en cadena de cambios en los programas existentes y resultar así demasiado caros éstos.

El desarrollo de la teoría del manejo de bases de datos responde a una necesidad de romper el cuello de botella que representaba la cada vez mayor necesidad de manipulación de información, por un lado, y la falta de comunicación entre los archivos que la almacenaban, por el otro. La redundancia innecesaria de la información genera problemas que con frecuencia conducen a fallas en la integridad de la misma y esfuerzos duplicados de captura y actualización de los datos.

En este capítulo se presentan las bases teóricas sobre las cuales está fundamentado el diseño de *SABAD*.

Se introduce el vocabulario básico que será utilizado en el resto de la tesis.

Se hace una exposición breve de las características del modelo relacional de bases de datos que es el que caracteriza a *SABAD*.

1.1. Conceptos básicos.

Se presentan los conceptos fundamentales sobre los que se apoya este trabajo de tesis.

Para poder hablar con precisión acerca del manejo de bases de datos es recomendable establecer qué se entiende por:

- **dato,** (del latín *data*, plural de **datum**) significa simplemente "hechos", entidades independientes sin evaluar[1].

- **información,** es un conjunto ordenado de datos los cuales pueden recuperarse de acuerdo con la necesidad del usuario[2].

- **campo,** también llamado *ítem de datos*, es el grupo de datos más pequeño[3] que posee un nombre para ser identificado.

- **registro,** conjunto de campos con relación entre sí[4]. Aunque el sistema operativo ve un archivo simplemente como una colección de bytes con nombre, los programadores cuando pensamos en ar-

---

1 Tsai, SISTEMAS DE BASES DE DATOS, p3.
2 íbid, ídem.
3 Martin, J., *Organización de las Bases de Datos*, pág. 11.
4 Tsai, op. cit.

chivos lo hacemos como si estuvieran compuestos de uno o más registros. Cada registro describe a una persona u objeto en particular, puede estar compuesto de diversos campos de varios tipos, y tiene un número entero de bytes como longitud.[5]

- **archivo**, colección de registros del mismo tipo[6].

- **buffer**, o depósito de transferencia es un lugar para guardar algún bloque de datos que va a ser transferido entre un almacenamiento secundario y la memoria del programa[7].

En la época en que no se contaba con verdaderas bases de datos, sino con sistemas de archivos, los métodos de procesamiento se tenían que quedar congelados por la estructura de los datos que utilizaban[8],ello debido a la falta de generalidad en los métodos de acceso y la carencia de un modelo teórico que sustente la estructura y el manejo de la información.

Para evitar que cada programador intente modificar la estructura y los métodos de manejo de los datos a su antojo, como si fueran algo propio y privado de ellos y no recursos

---

5 Duncan, Ray en el artículo **File management in C and assembly language**, publicado en PC MAGAZINE, Feb. 28, 1989, p. 299.

6 Tsai, op. cit.

7 Tsai, op. cit., pág. 9

8 Martin, *Organización de las Bases de Datos*, p.5

valiosos de la empresa donde prestan sus servicios, se recomienda mantener una independencia física y lógica[9] de los datos con respecto a los sistemas manejadores de los mismos.

Atendiendo a las definiciones que nos dan diversos autores acerca de lo que son las bases de datos tenemos lo que dice Martin:

> **La base de datos puede definirse como una colección de datos interrelacionados almacenados en conjunto sin redundancias perjudiciales o innecesarias.**
>
> **Su finalidad es la de servir a una aplicación o más, de la mejor manera posible.**
>
> **Los datos se almacenan de modo que resulten independientes de los programas que los usan.**
>
> **Se emplean métodos bien determinados para incluir datos nuevos y para modificar o extraer los datos almacenados.**[10]

Date indica lo siguiente:

---

9 El nivel lógico se referirá a la forma en que un programador ve las cosas, aunque no necesariamente signifique que en realidad los datos estén grabados de esa forma, lo cual corresponde al nivel físico.

10 Martin, *op. cit.*, pág. 19.

> **¿Qué es exactamente un sistema de bases de datos? En esencia, no es más que un sistema de mantenimiento de registros basado en computadoras, es decir, un sistema cuyo propósito general es registrar y mantener información. Tal información puede estar relacionada con cualquier cosa que sea significativa para la organización donde el sistema opera --en otras palabras, cualquier dato necesario para los procesos de toma de decisiones inherentes a la administración de esa organización.**[11]

Por su parte, Tsai indica que:

> **Un sistema de bases de datos es un sistema computarizado de información para el manejo de datos por medio de paquetes de software llamados sistemas de manejo de bases de datos (DBMS). Los tres componentes principales de un sistema de bases de datos son el hardware, el software DBMS y los datos por manejar.**
>
> **Una base de datos es una colección de archivos interrelacionados creados con un DBMS. El contenido de una base se obtiene combinando**

11 Date, *Introducción a los Sistemas de Bases de Datos*, p. 5

datos de todas las diferentes fuentes en una organización, de tal manera que los datos estén disponibles para todos los usuarios, y los datos redundantes puedan eliminarse, o al menos minimizarse.[12]".

En general se puede concebir un sistema de bases de datos como compuesta por cuatro elementos, tal como se aprecia en la figura 1.1: Programas de aplicación, el sistema de administración de bases de datos (DBMS), las bases de datos, el hardware y los usuarios.

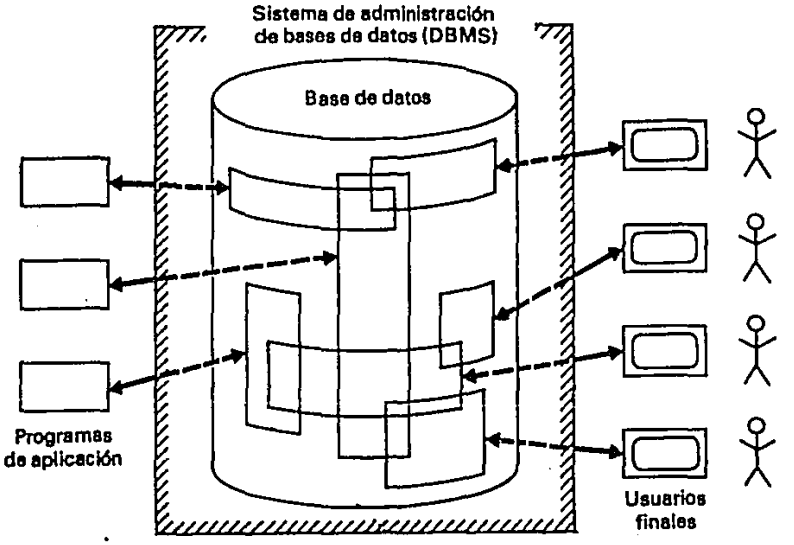

Fig. 1.1 *Un sistema de bases de datos*

12 Tsai, op. cit., p. 5

Los *programas de aplicación* son aquellos programas que realizan los cómputos específicos que el usuario necesita obtener y que se derivan a partir de la información que ha ido almacenando tales como totales de ventas, saldos por pagar, volumen de operaciones en el mes, etc.

El *sistema de administración de bases de datos* (DBMS) es un conjunto de programas básicos que manejan todas las solicitudes de acceso a la base de datos formuladas por los usuarios. De esta forma, se puede concebir al DBMS como un intermediario entre el usuario y el hardware, de manera que el usuario no tenga que ocuparse de detalles rutinarios y muy detallados como el manejo de los índices o la posición de los registros físicos dentro del archivo en disco, etc.

Las *bases de datos* son tal como han sido explicadas anteriormente.

El *hardware* se compone de los volúmenes de almacenamiento secundario -discos, unidades de cinta- donde reside la base de datos, junto con los dispositivos necesarios para su manejo, tales como tarjetas controladoras, etc.

Por su parte, los *usuarios* se pueden dividir en tres grandes categorías:

- **Programador de aplicaciones**, es el encargado de escribir los programas de aplicación que utilicen las bases de datos, por lo general en un lenguaje de alto nivel, como en el caso del trabajo que se presenta aquí en C.

- **Usuario final**, accesa la base de datos desde una terminal. Un usuario final puede emplear un lenguaje de consulta proporcionado como parte integral del sistema o recurrir a un programa de aplicación escrito por un programador de aplicaciones, el cual formule por él las solicitudes de acceso a las bases de datos (como es el caso en mi sistema DBMS).

- **Administrador de bases de datos**, el cual debe decidir con exactitud qué información se mantendrá en la base de datos, la manera en que habrán de representarse los mismos, la vinculación con los demás usuarios, entre otras importantes tareas de control de la información.

Más específicamente, en cuanto a lo que es un sistema administrador de bases de datos tenemos:

> **"Un DBMS es una colección de numerosas rutinas de software interrelacionadas, cada una de las cuales es responsable de alguna tarea específica[13]."**

En cuanto a cuáles deben ser las tareas de un DBMS se apunta:

> **i) crear y organizar la base de datos**

13 Tsai, op. cit., p. 6

ii) establecer y mantener las trayectorias de acceso a la base de datos, de tal manera que los datos en cualquier parte de la base se puedan accesar rápidamente.

iii) manejar los datos de acuerdo con las peticiones de los usuarios

iv) mantener la integridad y seguridad de los datos

v) registrar el uso de las bases de datos[14]"

Dos características muy importantes de las bases de datos es que sean *integradas* y *compartidas*[15].

Por *integrada* se entiende que la base de datos puede considerarse como la unificación de varios archivos de datos independientes, donde se elimina total o parcialmente cualquier redundancia entre los mismos.

Por *compartida* se entiende que partes individuales de la base de datos pueden ser utilizadas por varios usuarios distintos, en el sentido de que cada uno de ellos puede tener acceso a la misma parte parte de la base de datos. La palabra compartida, a menudo se amplía para abarcar no sólo al compartimiento antes descrito, sino también al compartimiento *concurrente*, es decir, que diversos usuarios accesen la base de datos al mismo tiempo.

---

14 Tsai, op. cit., p. 6
15 Date, op. cit., pág. 6

Uno de los objetivos de un sistema de administración de bases de datos es evitar a los usuarios la tediosa programación que tendrían que hacer para organizar el almacenamiento de los datos, o accesarlos una vez almacenados.

Por su parte, Date[16] nos indica otras ventajas de utilizar bases de datos: reducir la redundancia, evitar la inconsistencia, compartir los datos, hacer cumplir las normas establecidas, aplicar restricciones de seguridad, conservar la integridad, y, sobre todo, proporcionar la independencia de los datos respecto de los programas que los manejan. También se tiene la ventaja de poder ejercer un control centralizado de los datos de operación de una empresa, los cuales si los concebimos como un valioso activo de la misma, deben ser controlados cuidadosamente.

1.2. Maneras en las que es posible representar los datos en un archivo de disco.

En los archivos de disco los elementos de datos (campos) deben ser almacenados de manera que su recuperación y posibles modificaciones se vean favorecidas por la estructura de almacenamiento que se haya decidido utilizar.

En cuanto a la representación de los campos de información dentro de registros, se utilizan básicamente cuatro métodos[17]:

- **Posicional**, los datos de un registro se almacenan en campos consecutivos de longitud fija en un orden predeterminado. Esta disposición es sencilla de reali-

16 Date, op. cit., pp. 12-14
17 Tsai, op. cit., pág. 23

zar y conveniente para recurperar la información, pero tiene la desventaja de que desperdicia espacio si la información guardada no utiliza (lo que comúnmente sucede) todo el espacio definido, quedando éste vacío.

- **Relacional,** esta técnica elimina los espacios desperdiciados mediante el uso de un delimitador que indica el final de un campo. El delimitador debe ser un carácter especial que no se usará en ningún otro sitio en la base de datos.

- **Indexada,** se utilizan aquí señaladores para indicar el principio de cada campo en un registro almacenado. El señalador puede ser una *dirección absoluta* o *relativa*.

- **Etiquetada,** cuando un archivo tiene muchos valores por omisión, se puede usar la técnica etiquetada para almacenar únicamente los valores diferentes a los de por omisión.

El balance en el diseño de la representación de los campos se encuentra entre la cantidad de espacio de almacenamiento y la velocidad de recuperación, por lo que la elección para el diseño deberá hacerse considerando estos dos parámetros. Ninguna de las disposiciones anteriores es superior a las otras si se consideran ambos parámetros. Por ejemplo, la técnica posicional es la más sencilla, pero no es buena en cuanto a la economía del espacio de almacenamiento. El método relacional reduce ese problema, pero con ello disminuye la velocidad de recuperación, ya que la información debe buscarse pasando por todos

los datos del registro y contando cierto número de delimitadores para encontrar finalmente la ubicación del campo deseado, a costa de la necesidad de mantener arreglos de señaladores[18]

Es pertinente señalar que la elección de la representación de los campos podría afectar directamente la dependencia de las aplicaciones con respecto de los datos. Esto podría significar que la manera como los datos se organizan en almacenamiento secundario y la manera como se accesan dependan de los requerimientos de la aplicación y, además, que el conocimiento de la organización de los datos y de la técnica de acceso forme parte de la lógica de la aplicación; lo cual complicaría enormemente la posibilidad de realizar cambios en cualquiera de las dos características mencionadas debido a que, casi con seguridad, implicarían el desarrollo de aplicaciones nuevas y el abandono de las antiguas por quedar inmediatamente obsoletas, lo cual por muchas razones no es aceptable.

Por dicha razón es conveniente adoptar un modelo de representación y manejo de los datos que sirva como paradigma en el cual nos podamos mover sin caer en inconsistencias ni problemas como los mencionados en el párrafo anterior. A continuación se explicará uno de los modelos más conocidos y utilizados en la actualidad.

1.3. Indices

El propósito de un índice es proporcionar una liga de acceso para el archivo que está indicando, es decir, una forma de localizar los registros en el archivo indicado[19].

---

18 Tsai, op. cit., pág. 24
19 Date, *op, cit.*, p. 44

Un archivo particular puede tener asociadas muchas ligas de acceso. Una liga que siempre está disponible es el ordenamiento secuencial simple -siempre es posible realizar una búsqueda exhaustiva a través del archivo, registro por registro, de acuerdo a la secuencia básica del método de acceso que se esté utilizando.

Es posible distinguir dos tipos básicos de índices generalmente llamados *índices densos* e *índices no densos*.

El término *denso* se refiere al hecho de que el índice contenga una entrada para cada ocurrencia de registro almacenado en el archivo indicado; por su parte, los índices no densos (como es el caso de aquellos basados en la estructura de *árboles B*), pueden contener varias ocurrencias de registro almacenado (junto con el campo indicado) para cada entrada del índice.

Los índices no densos tienen la característica de que, a pesar de que una de las principales razones de existencia de los índices es la de evitar la necesidad de realizar recorridos secuenciales en un archivo cada vez que se desea localizar la información de un registro, necesitan hacer un recorrido secuencial en el índice para localizar cuál de las ocurrencias nos da la clave de la información que se está buscando. Sin embargo, esta situación puede ocasionar un problema grave de eficiencia cuando el índice crece.

1.4. Bases de Datos Relacionales

Las bases de datos relacionales (bases de datos con una estructura relacional) son las preferidas actualmente[20] en el mercado de los manejadores de bases de datos.

Buena parte del atractivo de dicha estructura de organización de los datos proviene del hecho de que se puede establecer una correspondencia directa con la teoría matemática de conjuntos.

Un concepto fundamental en la teoría de este tipo de bases de datos es el de **relación**, la cual se puede definir formalmente de la siguiente manera:

Sean D_1, D_2, ..., D_n conjuntos (no necesariamente distintos). Se dice que **R** es una *relación* sobre estos n conjuntos si **R** es un conjunto de n tuplas ordenadas $<d_1, d_2, ..., d_n>$ tales que d_1 pertenece a D_1, d_2 pertenece a D_2, ..., d_n pertenece a D_n. Los conjuntos D_1, D_2, ..., D_n son los *dominios* de **R**. El valor n es el *grado* de **R**.

Un *dominio* se puede entender como un depósito de valores del cual se obtienen los *atributos* o valores que pueden tomar las tuplas en una relación.

---

20 esta posición de preferencia ha tenido lugar desde fines de los años 70 hasta el momento de realizar esta tesis -1993. Aunque parece inevitable que tal preferencia la pierdan las bases de datos de estructura relacional en un futuro muy cercano a favor de las *bases de datos deductivas*, que utilizan técnicas de sistemas expertos, o las *bases de datos orientadas a objetos*, aunque ésto último lo encuentro, en lo personal, menos probable.

Una característica crucial de la estructura de datos relacional es que las asociaciones entre tuplas se representan únicamente por valores de datos en columnas sacadas de un dominio común[21].

Es conveniente representar una relación en forma de tabla, donde cada renglón de la tabla represente una *n* tupla (o sencillamente una *tupla*) de la relación. El número de tuplas de una relación se llama *cardinalidad* de la relación.

Cada tabla de esta representación se parecerá a un archivo secuencial convencional, donde los renglones correspondan a los registros del archivo y las columnas a los campos de los registros. A pesar de las similitudes evidentes como la anterior entre archivos secuenciales y las tablas es importante no perder de vista que las tablas son el producto de una teoría matemática sólida que permite un tratamiento mucho más riguroso, en particular, una tabla en este contexto responde al concepto matemático de relación expuesto anteriormente.

Otros términos importantes en la teoría de las bases de datos relacionales son los de *dominio* y *columna* (atributo) que se obtiene del dominio.

Un *atributo* representa el uso de un dominio dentro de una relación.

Otra importante idea del campo de las bases de datos con estructura relacional es la de la *normalización* de los datos.

Es un requisito indispensable que todas las relaciones de una base de datos relacional satisfagan la siguiente condición: en cada intersección de un renglón y una columna de cada tabla

21 Date, op. cit., p. 73

deberá haber exactamente un valor, nunca un conjunto de valores. Se permite, sin embargo, la posibilidad de valores *nulos*, es decir, valores especiales que representan algo desconocido o inaplicable.

De una relación que satisface la condición anterior se dice que está *normalizada*[22], más concretamente se dice que dicha relación está en la *primera forma normal.*

Los objetivos del enfoque relacional son[23]:

- Proporcionar un alto grado de independencia de los datos[24]
- proporcionar una vista comunitaria de los datos de sencillez espartana, de modo que una amplia diversidad de usuarios en una empresa (desde el más inexperto hasta el más versado en computadoras) pueda interactuar con una vista común (sin prohibir vistas yuxtapuestas de los usuarios para fines especializados).
- simplificar el trabajo potencialmente abrumador del administrador de bases de datos

---

22 Date,AN INTRODUCTION TO DATABASE SYSTEMS , pp. 95-96

23 Date, *op. cit.* , pp. 541-542.

24 Es posible definir la independencia de los datos como la inmunidad de las aplicaiones a los cambios de la estructura de almacenamiento y de la estrategia de acceso- lo que implica que las aplicaciones no dependen de ninguna estructura de almacenamiento o estrategia de acceso en especial-.

- introducir un fundamento teórico (aunque modesto) en la administración de bases de datos (campo que desafortunadamente carece de pautas y principios sólidos).

- unir los campos de la recuperación de hechos y de administración de archivos como preparación para la adición, en un instante posterior, de servicios de inferencia en el mundo comercial

- elevar la programación de aplicaciones de bases de datos a un nivel nuevo -un nivel donde los conjuntos (y más específicamente las relaciones) se traten como operandos en lugar de procesarse elemento por elemento.

Analizando lo anterior, creo que sigue siendo válida la observación de Date respecto de que "*nadie afirmaría que estos objetivos ya se han alcanzado; mucho trabajo queda por hacerse*".[25]

Puede decirse que, en términos tradicionales, una **relación** se asemeja a un *archivo*, una **tupla** a un *registro* (a la ocurrencia, no al tipo) y un **atributo** a un *campo* (al tipo, no a la ocurrencia).

Las relaciones pueden considerarse archivos altamente disciplinados -esta disciplina se traduce en una simplificación considerable de las estructuras de datos que el usuario debe

25 Date, op. cit., pág. 542.

manejar, y, por tanto, en una simplificación correspondiente de los operadores necesarios para manipularlas.[26]

1.5. Algebra relacional

Muchos problemas se expresan con mayor naturalidad, no en términos de registros individuales, sino en términos de conjuntos. De ahí que se haya buscado desarrollar lenguajes de manipulación de datos con la posibilidad de manipular conjuntos completos como si fueran objetos singulares, en lugar de estar restringidos a un solo registro cada vez.

Como el modelo de bases de datos con estructura relacional está sólidamente apoyado en la teoría matemática de conjuntos, es natural trasladar la herramienta de manipulación de conjuntos, el álgebra de conjuntos, al campo de las bases de datos, obteniéndose el álgebra relacional.

El álgebra relacional define las siguientes operaciones fundamentales:

- **Unión**, la unión de dos relaciones **A** y **B** produce una nueva relación que contiene todas las tuplas que están en **A** y todas las tuplas que están en **B**; desde luego sin repetición, debido a que los conjuntos no admiten elementos repetidos.

- **Intersección**, es el conjunto de todas las tuplas que pertenecen simultáneamente a cada una de las relaciones que se están intersectando.

---

26 Date, op. cit., p. 102

- **Diferencia.** La diferencia de una relación **A** con respecto de una relación **B**, produce el conjunto de todas las tuplas que están en **A**, pero que no están en **B**.

- **Producto cartesiano.** Sean $D_1, D_2, ..., D_n$ conjuntos (no por fuerza distintos). El producto cartesiano de estos n conjuntos, denotado por $D_1 \times D_2 \times ... \times D_n$, es el conjunto de todas las n tuplas ordenadas posibles $<d_1, d_2, ..., d_n>$ tales que d_1 pertenece a D_1, d_2 pertenece a D_2, ..., d_n pertenece a D_n.

- **Producto cartesiano extendido.** El producto cartesiano extendido de dos relaciones **A** y **B**, (**A veces B**) es el conjunto de todas las tuplas t tales que t es la concatenación[27] de una tupla a que pertenezca a **A** y una tupla b que pertenezca a **B**. Esta operación es mucho más común en el campo del manejo de bases de datos que la del producto cartesiano simple, debido a que esta operación proporciona información útil acerca de las tablas que se operan y no únicamente un conjunto de posibles asociaciones potenciales entre las tuplas, como sería el caso del producto cartesiano simple.

---

27 La concatenación de dos tuplas a y b consiste en copiar los valores de los atributos de la tupla a seguidos de los valores de los atributos de la tupla b para dar lugar así a una nueva tupla t con los valores de los atributos tanto de a como de b.

Todas estas operaciones, excepto el producto cartesiano, están definidas para relaciones compatibles con la **unión**, lo cual significa que aceptan como operandos relaciones representadas en forma de tablas con el mismo *grado* (manifestado en las tablas con el mismo número de columnas) y los *atributos* o valores de las tuplas deben tener el mismo *dominio* subyacente.

Además de las operaciones antes mencionadas, es común definir otras operaciones útiles en el manejo de las bases de datos relacionales, a las cuales se les denomina *operaciones relacionales especiales*, y son:

- **Selección**, este operador algebraico (no debe ser confundido con el operador SELECT de SQL[28]) produce un subconjunto *horizontal* de una relación específica -es decir, el subconjunto de las tuplas de la relación dada para el cual se cumple un predicado específico-. El predicado se expresa como una combinación booleana de términos, donde cada término es una comparación simple que se puede establecer como verdadera o falsa para una tupla dada inspeccionando esa tupla por separado.

- **Proyección**, este operador produce un subconjunto *vertical* de una relación dada -es decir, el subconjunto obtenido al seleccionar los atributos especificados, en un orden especificado de izquierda a derecha y eliminando luego las tuplas

28 SQL es un lenguaje estructurado de consulta que ha venido ganando aceptación como un estándar *de facto* en la industria.

duplicadas en los atributos seleccionados. Como se está asignando significado al orden de los atributos dentro de una relación, la proyección proporciona una manera de permutar los atributos de una relación específica.

- **Reunión** (*Join*), cuando dos relaciones o tablas contienen al menos una columna idéntica, este hecho se puede utilizar para juntar las tuplas que contienen el mismo valor en dicho atributo común; cuando la reunión de las tuplas se basa en la igualdad del atributo común se habla de la *equirreunión*, y, aunque también es posible definir otro tipo de reuniones basadas en la desigualdad (diferente de, mayor o menor que), la equirreunión es la más usual. Desde luego, siempre es posible eliminar una de las dos columnas repetidas por medio de la operación **proyección**; se llama *reunión natural* a una equirreunión con una de las dos columnas idénticas eliminada.

- **División**, este operador divide una relación dividendo **A** de grado $m+n$ entre una relación divisor **B** de grado n, y produce una relación resultado de grado m. Se puede considerar los primeros m atributos de **A** como un sólo atributo compuesto **X**, y los últimos n como otro, **Y**; entonces **A** puede considerarse como un conjunto de pares de valores $<x,y>$. Asimismo, **B** puede considerarse como un conjunto de valores simples $<y>$.

> Entonces, el resultado de dividir **A** entre **B** es el conjunto de los valores *x* tales que el par <*x*,*y*> aparece en **A** para *todos* los valores *y* que aparecen en **B**.

El álgebra relacional es importante para la manipulación de los datos, aunque sea menos amigable que los lenguajes modernos de consulta como SQL y QBE[29]. La razón básica de su importancia estriba en que proporciona un patrón con respecto al cual se pueden medir otros lenguajes. Puesto que para mostrar que algún lenguaje L es equivalente al álgebra relacional basta con demostrar que en L se puede generar cualquier expresión que se puede generar en el álgebra, con frecuencia se acepta la equivalencia indicando que el lenguaje L incluye equivalentes de los operadores algebraicos. Tal es el caso del Cálculo Relacional, el cual es el resultado de utilizar la herramienta matemática del cálculo, aplicándola al manejo y conceptualización de las bases de datos relacionales. Aunque la herramienta es muy buena y sirve perfectamente para la conceptualización antes mencionada, se puede establecer la equivalencia con el álgebra relacional, la cual es más comprensible y manejable para un buen número de personas, y por lo tanto, es la herramienta más común para estos propósitos.

1.6. Llaves

Es común que dentro de una relación exista un atributo cuyos valores sean únicos dentro de la relación y se puedan usar para identificar las tuplas de esa relación. A este atributo se le denomina *llave primaria*.

---

29 Query-by-Example (Lenguaje de consultas al cual se le proporciona un ejemplo de lo que se espera obtener).

No toda relación contiene una llave primaria de un sólo atributo; sin embargo, cada relación *tiene* alguna combinación de atributos que, tomados en conjunto, tienen la propiedad de la identificación única. La existencia de esa combinación está garantizada por el hecho de que una relación es un conjunto[30].

Se puede encontrar en ocasiones una relación donde hay más de una combinación de atributos que tienen la propiedad de identificación única, entonces se habla de *llaves candidatas*. En tales casos, se puede escoger cualquiera de las llaves candidatas para ser la *llave primaria*, y las demás se conocerán también con el nombre de *llaves alternas*.

Las llaves deben cumplir con ciertas reglas para que su utilización sea correcta, a estas reglas se les denomina *reglas de integridad*[31]:

- **Integridad de la entidad.**- Ningún componente de un valor de una llave primaria debe ser nulo.

- **Integridad de referencia.**- Ninguna relación deberá hacer referencia a tuplas inexistentes de otra relación a la cual está asociada por medio de una llave primaria común. Para expresar lo anterior en téminos formales se introduce el concepto de *dominio primario*: Un dominio específico puede designarse como *primario* si y sólo si existe alguna llave

30 puesto que los conjuntos no tienen elementos duplicados, cada tupla de una relación específica es única con respecto a ésa relación, de tal manera que, cuando menos, la combinación de todos los atributos tiene la propiedad de identificación única.

31 Date, *op. cit.*, pág. 99.

primaria de un sólo atributo definida sobre ese dominio. Si D es un domio primario y R_1 una relación con un atributo A que se define sobre D, entonces, en cualquier instante dado, cada valor de A en R_1 debe ser nulo o igual a V, donde V sea el valor de la llave primaria de alguna tupla de alguna relación R_2 (R_1 y R_2 no tienen que ser relaciones distintas) con llave primaria definida sobre D.

1.7. Tipos de sistemas de manejo de las bases de datos relacionales

Con las bases teóricas anteriores se puede definir el *modelo de bases de datos relacionales* de manera formal como aquel que consta de dos componentes principales:

- la estructura de datos relacionales,
- el álgebra relacional.

Un sistema de bases de datos se puede llamar *totalmente relacional*[32] si soporta:

- bases de datos relacionales (incluidos los conceptos de dominio, y llave y las dos reglas de integridad); y

32 E.F. Codd, Extending the Database Relational Model to Capture More Meaning, ACM Transactions on Database Systems 4, núm. 4, dic. 1979, citado por Date en la obra citada, pág. 237.

- un lenguaje que sea al menos tan potente como el álgebra relacional[33].

Un sistema que soporta bases de datos relacionales, pero que tiene un lenguaje menos potente que el álgebra, se puede llamar *semirrelacional*[34].

Date hace un pequeño análisis al respecto del estado del arte en el campo de las bases relacionales a 1981, mismo que sigue siendo válido después de más de diez años y por ello lo resumiré aquí, dice Date: "*cabe señalar que tal vez no existen hoy día sistemas que sean totalmente relacionales, ni siquiera semirrelacionales, de acuerdo con estas definiciones. Las definiciones relacionales han cambiado continuamente en los últimos diez años; las críticas en el sentido de que el sistema relacional es un blanco móvil no son infundadas, pero sería más exacto decir que los conceptos han evolucionado*"[35]

Para el caso específico de *SABAD* se puede afirmar que cae dentro de la categoría de sistemas semirrelacionales, de acuerdo al modelo de base de datos relacionales descrito en esta sección, debido a que todavía no cuenta con un lenguaje tan potente como el álgebra relacional. Puede llegar a tener un lenguaje así expandiendo el sistema básico que se encuentra instalado en la actualidad. No obstante, su funcionalidad es bastante amplia como puede verse en el siguiente capítulo donde se explica el diseño de *SABAD* tomando como base los conceptos teóricos aquí expuestos.

---

33 Date, *op. cit.*, pp. 236-237.
34 E.F. Codd, *Ibid, Idem.*
35 Date, *op, cit.* , pág. 237

NOTAS

2

Diseño de SABAD

RESUMEN DEL CAPITULO

Se expone el diseño de *SABAD* como manejador de datos y herramienta para el desarrollo de aplicaciones, en particular de tipo administrativo.

Se presenta la estructura modular del sistema y las relaciones que existen entre los módulos.

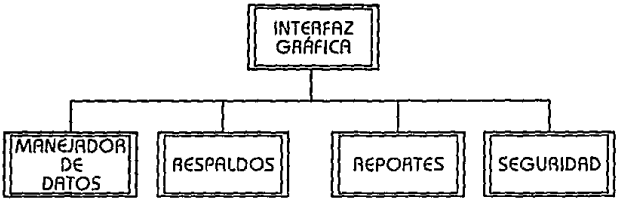

fig. 2.1

2. 1. Estructura General de SABAD.

La gran funcionalidad y flexibilidad de *SABAD* provienen del cuidadoso diseño modular en el que está basada su implantación. Se tuvo sumo cuidado en atender a la mayoría de las normas de construcción de sistemas que indica la Ingeniería de Software y se buscó, hasta donde fue posible, el encapsulamiento de los módulos para evitar dependencias que compliquen el

Un diagrama funcional de la estructura de *SABAD* se encuentra en la figura 2.1. Cada rectángulo representa una función[1] básica del sistema, generalmente encapsulada en un módulo compilable por separado, y cada línea entre dos rectángulos indica la comunicación entre las funciones respectivas.

Con la intención de hacer más fácil la lectura del diagrama se han detallado aquellos módulos que tienen un doble rectángulo en diagramas aparte; esto permite mayor claridad en los dibujos y la separación de los detalles que de otro modo se volverían oscuros por la pequeñez necesaria de los rectángulos para que pudieran aparecer todos en una sóla figura.

El diseño de *SABAD* gira en torno a cinco operaciones que son: interfaz gráfica, manejo de datos, manejo de respaldos, manejo de reportes y seguridad de acceso.

2. 2. Interfaz.

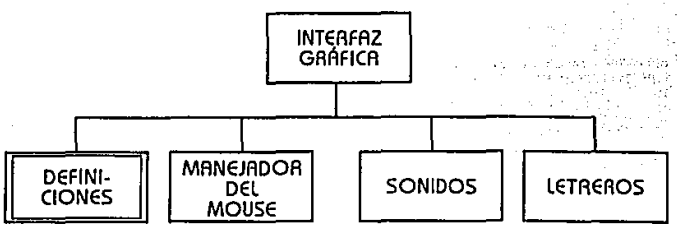

fig. 2.2

La interfaz gráfica deberá contener las funciones necesarias para la creación y manejo de ventanas, la creación y operación de los menús del usuario, así como de funciones de apoyo y estética de la pantalla como son la presentación de una ventana con la fecha y hora del sistema, etc. Adicionalmente, se deberán tener en este módulo las funciones para manejar los efectos de

1 Por función básica no me refiero al concepto de función en el lenguaje C, sino a las capacidades operativas del sistema en términos generales.

sombra de las ventanas y el manejo de la textura del fondo de la pantalla, para que no se vea tan vacío.

Para ejecutar una opción remarcada con el fondo de color especial el usuario lo único que deberá hacer es pulsar la tecla ENTER. Otra alternativa para hacer que se ejecute una opción será utilizar su tecla rápida, para lo cual sólo será necesario oprimir la tecla definida[2] y, en ése caso, la opción se ejecutará sin necesidad de la tecla ENTER, de ahí su nombre de *método rápido*.

Finalmente, también será posible seleccionar opciones pulsando el botón izquierdo del ratón sobre la línea de la opción deseada.

Para salir de un menú sin seleccionar ninguna opción, simplemente se pulsará la tecla ESC, o si se cuenta con un ratón se podrá oprimir el botón izquierdo sobre la esquina superior izquierda del menú marcada adecuadamente para tal efecto.

Aún cuando se cuente con un ratón, deberá ser posible seleccionar opciones mediante el uso de las teclas.

Este módulo para cumplir con sus objetivos se deberá apoyar en otros cuatro módulos que son: definiciones, manejador del mouse, manejador de sonidos y módulo de letreros, como puede verse en la figura 2.2.

2. 2. 1. Definiciones.

En el módulo de definiciones se tiene cuatro submódulos, como puede apreciarse en la fig. 2.3.

---

2 yo recomiendo presentar las opciones en pantalla utilizando sólo minúsculas para que la letra en mayúsculas sea la clave rápida de la opción aunque también podría hacerse ésto utilizando diferentes colores.

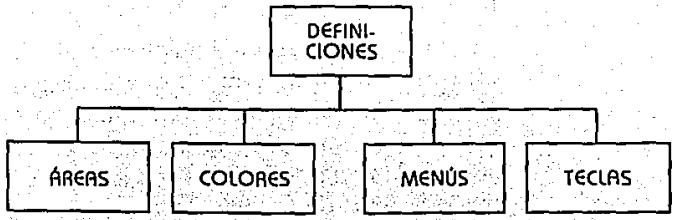

fig. 2.3

2. 2. 1. 1. Areas

El submódulo de áreas contendrá todas las definiciones de las ventanas que serán necesarias para la interfase, tales como las ventanas de menús y submenús, la de títulos, mensajes al usuario, etc.

2. 2. 1. 2. Colores.

En cuanto al submódulo de colores, en él se deberá contener la información relativa a los colores de la interfase, de forma tal que si se requiere una aplicación en blanco y negro, cambiando las definiciones de este módulo se obtenga el efecto deseado sin mayor problema.

2. 2. 1. 3. Menús.

Las definiciones de los menús se refieren a la declaración de las opciones que serán presentadas al usuario con el objeto de que se les pueda tomar con el mouse o las teclas claves señaladas y de esa manera devuelvan un código numérico que corresponda a la opción deseada.

2. 2. 1. 4. Teclas.

El módulo de definición de teclas es aquel que debe declarar los códigos correspondientes a las teclas especiales que serán reconocidas por la aplicación (tales como AV. DE PÁG., teclas de función, etc.) así como los nombres simbólicos que deberán ser asociados con dichas definiciones.

2. 2. 2. Manejador del mouse.

El módulo del manejador del mouse que aparece en la fig. 2.2 tendrá las definiciones de las funciones que controlan el mouse tales como leer el estado de los botones, leer la posición del mouse, encender y apagar la flecha del cursor del mouse, etc.

2. 2. 3. Sonidos.

Continuando con la misma figura se encuentra el módulo de sonidos, el cual, como su nombre lo indica tendrá como objetivo el controlar la emisión de sonidos y tonos musicales que den retroalimentación auditiva al usuario cuando seleccione opciones, cuando cometa equivocaciones, cuando se requiera llamar su atención al solicitar confirmación, etc.

2. 2. 4. Letreros.

Finalmente, el último módulo de la fig. 2.2 lo constituye el de letreros, el cual deberá agrupar los diferentes textos que se utilizarán como títulos de ventanas o como indicadores de qué tipo de información se desea que proporcione el usuario, como en el caso de las pantallas de captura.

2. 3. Manejador de datos.

El manejo de los datos es la espina dorsal de *SABAD* ya que comprende las actividades fundamentales del mismo. Aquí se administrará la información de los archivos en disco y se plantea la utilización en todo momento de un método indirecto de manipulación de los registros por medio de un índice con el objeto de proporcionar el acceso más rápido posible a los datos en el orden lógico que requiera el usuario sin necesidad de reordenar físicamente la información de base.

Como se puede apreciar en el diagrama de la figura 2.4 se ha diseñado esta parte en base a cuatro módulos.

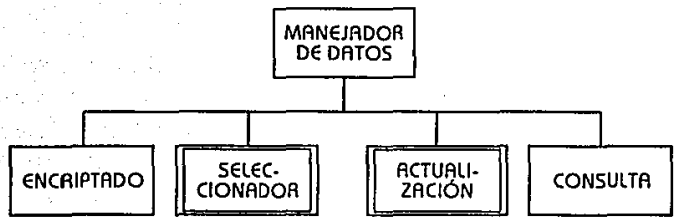

fig. 2.4

2. 3. 1. Encriptado de datos.

El módulo de encriptado debe cumplir con la función de proteger la información que se escribe a disco de tal forma que ésta no pueda ser leída por usuarios no autorizados que quisieran acceder a los datos sin la utilización del sistema SABAD. Para lograr sus objetivos este módulo deberá contar con una función de cifrado de datos la cual, en forma transparente para el usuario, codifique la información al momento de ser escrita a disco y la decodifique al pasarla de disco a memoria principal. Esta función de encriptamiento deberá ser lo suficientemente fuerte como para que no sea posible detectar el código por parte de un aficionado a la computación y que para alguna persona con una formación computacional más fuerte le implique una inversión en tiempo que vuelva muy cara esta alternativa. A la vez será necesario que la función de cifrado no sea tan compleja que imponga una sobrecarga excesiva al tiempo de ejecución de las aplicaciones para no afectar el desempeño del sistema.

2. 3. 2. Seleccionador.

Este módulo tendrá la función de ubicar el archivo de datos que se desee accesar así como el índice particular a emplear para aportar el ordenamiento lógico de la información que se especifique. El módulo deberá ser lo suficientemente inteligente como para detectar que si el archivo a seleccionar no existe proceda a crearlo inmediatamente para dar fluidez a la operación de la aplicación.

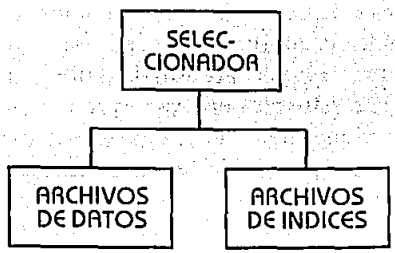

fig. 2.5

2. 3. 2. 1. Búsquedas y ordenamientos.

Una de las operaciones más necesarias de las bases de datos es la de realizar búsquedas de información rápidas de tipo aleatorio y secuencial en un cierto orden que el usuario desee. Para cumplir con este requerimiento se utilizará un sistema de indexamiento sencillo pero que satisfaga las necesidades de manera adecuada. Dicho sistema, el cual es un desarrollo original hasta donde se sabe y constituye una aportación de esta tesis, funcionará de la siguiente manera:

Se partirá de un archivo de registros idénticos, en cuanto al tipo de su estructura, grabado en disco secuencialmente en el orden en que se fue dando de alta los registros que lo componen y se utilizará un arreglo de estructuras tipo *arr_ram* (donde cada una contendrá un campo denominado clave el cual consistirá en un arreglo de caracteres para almacenar la llave y un campo entero denominado *pos_fis* donde se guardará la posición física (el desplazamiento en registros a partir del principio del archivo de disco) del registro que contenga la llave mencionada. Teniendo el archivo antes mencionado se leerá secuencialmente dicho archivo, registro por registro, ignorando aquellos registros que estén marcados como no-activos (bajas lógicas)[3] y por cada registro se copiará secuencialmente en el arreglo de estructuras

3 para el caso en que se deseara tener la opción de visualizar únicamente los registros dados de baja lógica se invertiría esta condición, es decir, se ignoraría todos los registros marcados como activos.

arr_ram el campo llave en el campo *clave* y en el campo *pos_fis* el contenido de una variable que cuente cuántos registros se hubieran leído (lo cual equivaldrá al desplazamiento en registros a partir del principio del archivo de datos), al terminar de leer el archivo que se estuviera indexando se procedería a aplicar una función de ordenamiento al arreglo de estructuras de RAM (por ejemplo la función de biblioteca *quicksort()*).

Una vez ordenado el arreglo de estructuras *arr_ram* en memoria, se procederá a escribirlo a disco para su utilización posterior.

De esta manera cada archivo de datos podrá tener diferentes índices (basados en llaves distintas, o en combinaciones de campos, o en prefijos de campos) cada uno escrito de manera independiente en el disco, listo para ser usado cuando se le requiera. Otra ventaja de utilizar este sistema es que es muy fácil guardar en memoria RAM todo el índice para acelerar las consultas, esto debido a que únicamente se guarda la información esencial de cada registro; en el caso de que se tenga archivos de datos tan grandes que su arreglo de índices no tenga cabida en memoria, entonces, sin ningún problema, se podría sustituir el módulo de índices actual por uno de índices no densos para mantener en memoria RAM la mayor cantidad de estructuras de tipo *arr_ram* que sea posible (una buena alternativa sería el uso de *árboles B*).

En *SABAD* todos los accesos al archivo de datos se realizan mediante la utilización de un índice; de tal manera que el índice maestro se crea desde que se graba el primer registro y se abre desde que se toma la opción de trabajar con dicho archivo. El índice se actualizará con cada cambio a la información del campo llave, cada vez que se borre o se agregue un registro más.

2. 3. 3. Actualización.

Este es la parte más importante del módulo manejador de datos. Esta parte se subdividirá en altas, bajas y cambios, como lo indica la figura 2.6.

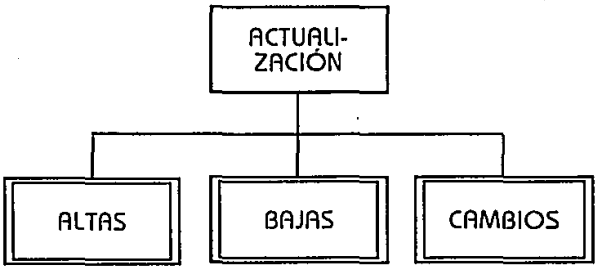

fig. 2.6

2. 3. 3. 1. Altas

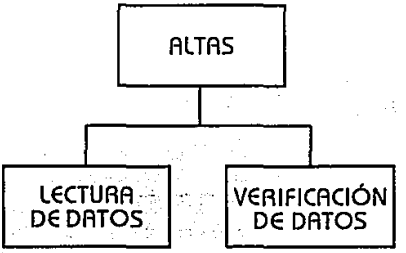

fig. 2.7

Esta parte será la encargada de capturar nuevos registros de información. Primeramente se deberá verificar la existencia del archivo de datos al cual le deseamos agregar un nuevo registro, ya que de no existir éste, se deberá proceder a crearlo de manera automática al igual que un índice por defecto sobre un campo que sea considerado llave del archivo. La información que se reciba deberá ser validada con el objeto de filtrar los más errores posibles que pudiera cometer el usuario a la hora de captura.

2. 3. 3. 2. Bajas.

Como se ve en la figura 2.8. Aquí se agruparán las funciones de baja de registros tanto a nivel lógico como físico. Se recomienda que antes de proceder a cualquier baja física prime-

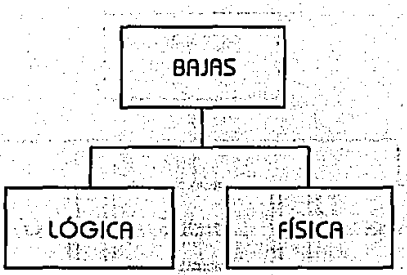

fig. 2.8

ro se efectúe una baja lógica, no sólo porque es más fácil y rápido sino porque de esa manera se le proporciona al usuario la oportunidad de cambiar de opinión y volver a dar de alta el registro que estaba marcado como baja lógica sin necesidad de volver a teclearlo de nuevo.

2. 3. 3. 3. Cambios.

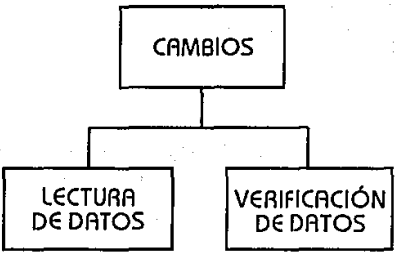

fig. 2.9

Esta es la parte donde se controlará todo cambio a información contenida en registros de un cierto archivo de datos. Se deberá proceder como con el módulo de altas, en cuanto a que será necesario validar la información de entrada que proporcione el usuario así como actualizar el archivo de índices si es que se realizó algún cambio en algún campo que funciona como llave de un índice.

2. 3. 4. Consulta.

Esta sección manejará las funciones de presentación de datos en pantalla y los, posibles, cruces de información que haya que realizar para poder presentar las vistas de los datos que requieran los usuarios. En este caso los archivos de datos se tratan como de sólo lectura.

2. 4. Respaldos

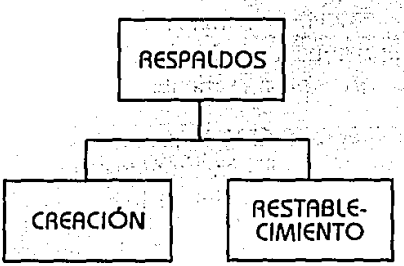

fig. 2.10

Este módulo deberá cumplir con la misión de proteger la información almacenada por el sistema, tal como archivos de datos, formatos de reportes definibles por el usuario y archivos de configuración. La forma de proteger dicha información es copiarla en discos flexibles con el objeto de que, en caso de una falla del disco duro u otra situación que involucre un daño a la información, ésta no se pierda debido a que es preciso considerarla como un recurso valioso.

Como se puede observar en el diagrama de la figura 2.10, el módulo deberá ser capaz de volver a leer los datos que fueron copiados a disco si así lo determinare el usuario.

2. 5. Reportes.

De acuerdo con el diagrama de la figura 2.1 de la página 33, el manejo de reportes también es fundamental, ya que con frecuencia se toma a los documentos impresos que genera el sistema como el producto tangible del mismo y no se le atribuirá

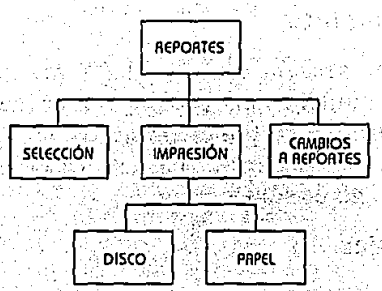

fig. 2:11

ninguna importancia a todo el trabajo de base si los reportes no son satisfactorios para el cliente.

En el diagrama 2.11 se presenta el diseño de este módulo destacando: la selección de reportes, la impresión de los mismos y los cambios que pueda desear hacer el usuario.

La selección de los reportes se refiere simplemente a proporcionar la posibilidad al usuario de elegir cuál de todos los reportes disponibles desea imprimir; facilitándole esa selección por medio de un menú.

La impresión de los reportes, a su vez tiene dos vertientes: la impresión a disco y la impresión en papel por razones que se detallan en el capítulo que habla de las funcionalidades de *SABAD*.

Otro componente importante del módulo de reportes es el que permite la modificación de los mismos por parte del usuario, de tal forma que el sistema sea flexible y se adapte a las necesidades, posiblemente cambiantes, del usuario sin que sea necesario tener que recurrir en cada ocasión al programador de aplicaciones para modificar sus formatos de reporte. Para este efecto se diseñó un pequeño lenguaje de reportes capaz de ser compilado por *SABAD* para dar como resultado un reporte encriptado que puede ser ejecutado por el sistema. Los detalles de este lenguaje se explican en la sección 3. 4. de la página 58 donde se trata la implantación de este diseño, el cual forma parte de las aportaciones originales de esta tesis.

2. 6. Seguridad.

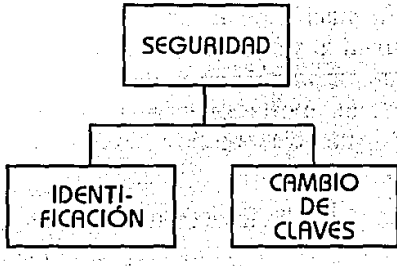

fig. 2.12

El módulo de seguridad es el que se encarga de verificar las contraseñas de acceso al sistema así como de proporcionar las herramientas necesarias para el cambio de contraseñas a discreción del usuario; estas funciones se encuentran esquematizadas en la figura 2.12.

Con el propósito de disminuir la posibilidad de daños a las bases de datos debido a fallas en el suministro de energía eléctrica, se adoptará la estrategia de abrir los archivos de datos únicamente para trasladar información del archivo en disco a memoria y viceversa, un registro a la vez, inmediatamente hecho lo cual se cerrará el archivo y se seguirá trabajando con la información en memoria. El esquema anterior deberá garantizar que la integridad de la información no será violada por accesos a los datos que no acabaron o por fallas en informar al sistema operativo de los cambios efectuados en un archivo antes de sobrevenir una situación anormal que hiciera abortar la operación que se estaba realizando.

2. 7. Compresión de datos

Es de sobra conocido el hecho de que, por lo general, las bases de datos de tipo administrativo, por contener un elevado volumen de texto y espacios en blanco resultan innecesariamente voluminosas, de tal forma que un mecanismo de compresión de los datos *al vuelo* resultaría muy conveniente para evitar el

desperdicio de espacio de almacenamiento. A este respecto se puede realizar un método de compresión de datos de manera similar al del método de encriptamiento, es decir, almacenar toda la información de manera comprimida, y descomprimirla (un registro a la vez) al trasladarla del archivo a memoria, y proceder a comprimirla antes de escribirla en el archivo. Esta también es una aportación original de esta tesis al campo de la operación de manejadores de bases de datos para computadoras personales.

Se plantea utilizar un método de compresión que se aproxime a la compresión óptima, según la plantea la teoría de la información, mediante la utilización de códigos de Huffmann.[4]

Se propone utilizar tablas de códigos de Huffmann resultado de un análisis estadístico de la información que sea habitual manejar por parte de la aplicación.

Para este efecto será conveniente realizar un programa de utilería que revise los datos almacenados por una aplicación, elabore la tabla de códigos de Huffmann adecuada y tenga la capacidad de recodificar los datos de *SABAD* mediante la nueva tabla, grabando en el archivo de configuración correspondiente la nueva tabla en lugar de la tabla anterior.

En el siguiente capítulo se verá en detalle cómo se logró realizar este diseño para llegar a tener a *SABAD* funcionando.

---

4 Los códigos de Huffmann consisten en la asignación de códigos de longitud variable a los símbolos de un alfabeto (a diferencia del código ASCII, por ejemplo, el cual le asigna siempre un byte a cada símbolo) donde los códigos más cortos se asocian con los símbolos más frecuentemente utilizados en el lenguaje y los códigos más largos a los símbolos que aparecen con menor frecuencia, de forma tal que, en promedio, la longitud de los mensajes sea prácticamente la mínima posible en términos de bits.

3

Implantación de SABAD

RESUMEN DEL CAPITULO

A pesar de ser un lenguaje muy eficiente y útil, las características del lenguaje ANSI C permiten muy pocas actividades de manejo de datos. Básicamente el lenguaje sólo posee la capacidad de escribir o leer un carácter a o desde un dispositivo, que pudiera ser un archivo, pero nada más. La biblioteca estándar *stdio.h* proporciona algunas funciones de manejo de archivos, sumamente primitivas, y por tal razón en la implantación de *SABAD* se tuvo que empezar por crear un conjunto de funciones de infraestructura que apoyen el trabajo a realizar por el manejador de bases de datos.

En este capítulo se describen los detalles de implantación que permitieron llevar a la práctica el diseño del capítulo anterior. Se comentan los principales problemas, se señalan las deficiencias y se muestran las soluciones que se adoptaron para superar los problemas encontrados.

Se indican detalladamente cuáles son las aportaciones particulares de *SABAD* al estado del arte en el ambiente de manejadores de bases de datos relacionales para computadoras personales que funcionan bajo el sistema operativo MS-DOS.

Además de las herramientas básicas que proporciona la biblioteca estándar de funciones del lenguaje ANSI C se tuvo que diseñar un conjunto de herramientas genéricas para *SABAD*.

Las herramientas se pueden agrupar por la función que desempeñan para hacer más fácil su presentación de la misma:

3. 1. Interfaz con el usuario

Para lograr una interfaz agradable y dinámica se empezó por diseñar un conjunto de funciones que manejen la memoria de video directamente (para lograr mayor velocidad en el cambio de pantallas) que proporcionen los elementos suficientes para captar la atención del usuario y hacerle más fácil seleccionar las opciones que desea utilizar. Se agruparon las funciones que hacen referencia directa al hardware de la computadora personal (las cuales no son portátiles a otras plataformas de cómputo) en el módulo **videostd.c**, el cual tiene el código para detectar el modo de video en que se encuentra el adaptador de video **CGA**[1] o **MDA**[2], localizar la dirección inicial de la memoria de video, y cambiar el modo de video a cualquier otro que tenga la capacidad el adaptador de video instalado. Adicionalmente, también se creó código para limpiar la pantalla, cambiar la apariencia del cursor de texto y desaparecerlo ,así como otras funciones comunes de manejo de la pantalla.

---

1 CGA, abreviatura en inglés de *Color Graphics Adapter*, el el adaptador de video más común en las computadoras personales, el cual es capaz de desplegar gráficas y colores.

2 MDA, abreviatura en inglés de *Monochrome Display Adapter*, el el adaptador de video que en un principio utilizó la computadora personal de IBM, el cual no podía presentar gráficas ni colores. Aunque muy limitado en sus capacidades, se le toma en cuenta porque la dirección inicial de la memoria de video no coincide con la del adaptador CGA, y para evitar que las aplicaciones fueran a tener problemas en ciertas computadoras, se verifica cuál de éstos adaptadores está presente. Si la computadora personal tiene algún otro adaptador de video no hay problema, debido a que prácticamente todos son compatibles con CGA o MDA.

Por consideraciones de portatibilidad a otras plataformas de cómputo se decidió utilizar una interfaz semigráfica (utilizando los caracteres gráficos del ASCII extendido) en lugar de una interfaz de texto (que en la actualidad resulta muy poco atractiva para los usuarios) o una interfaz gráfica (GUI) que es mucho más complicada de realizar y es muy dependiente del hardware particular de la máquina en la que se desee utilizar el sistema.

En esta interfaz semigráfica se utiliza un carácter de relleno para el fondo de la pantalla para darle textura a la misma y evitar la sensación de vacío que provocan las pantallas con fondo negro y letras brillantes. También se utiliza un efecto de sombreado alrededor de las ventanas para lograr el efecto visual de perspectiva y aparentar que las ventanas van apareciendo unas sobre otras y que la fuente de luz (simulando provenir de fuera de la pantalla y no, como en realidad sucede, proviene del monitor mismo) provoca una sombra sobre los elementos que quedaron abajo.

Para organizar mejor la información y aprovechar lo más posible el área de la pantalla del monitor se definen zonas de la pantalla como ventanas independientes entre sí, que se pueden distribuir aisladamente o en forma de cascada, unas sobre las otras. Cada ventana guarda su información de control en una estructura adecuada para ello; esto hace posible que el usuario cambie su posición a otro lugar de la pantalla sin ningún problema.

También se desarrolló un conjunto de funciones que controlan la bocina interna de la PC[3] para proporcionar una confirmación auditiva de la selección de una opción por parte del usuario o de la salida de la aplicación (caso en el cual se toca una melodía de despedida). Estas funciones de sonidos permiten determinar el tono y la duración de los mismos[4], los cuales también se pueden expresar por medio de notas musicales. Las funciones antes mencionadas y algunos efectos de sonido interesantes se agrupan en el módulo **sonido.c**.

3. 1. 1. Menús de opciones

Las estructuras que controlan las ventanas de menús de opciones[5] se implementan mediante un arreglo donde cada una guarda la siguiente información:

- **pantalla,** un apuntador sin tipo definido (void) para almacenar la dirección incial en memoria RAM donde se guarda la información correspondiente a la parte de la pantalla cubierta por la ventana en cuestión, de esa manera es posible que se pueda volver a presentar el contenido intacto de la parte de la pantalla que estaba debajo de la ventana al salir de la misma. Esto produce el efecto de que las ventanas aparezcan y desaparezcan como si flotaran

---

3 PC, abreviatura en inglés de *Personal Computer*, computadora personal.

4 Para las funciones de efectos de sonido que requieren controlar el chip de sonido de manera muy veloz se define una constante RETARDO que sirve para controlar una cuenta de un ciclo de espera, esta constante está definida en el archivo sonido.h y se debe utilizar la definición de computadora correcta (p.ej. AT486A25 o AT386A33) para ajustar esta variable.

5 los menús se identifican por el índice que utilizan dentro del arreglo de estructuras.

encima de la pantalla, dando la impresión de profundidad en el campo visual.

- **ini_lin,** valor de la linea inicial de la pantalla en la cual se empezará a desplegar la ventana.

- **fin_lin,** guarda la linea final de la pantalla que ocupará la ventana, incluyendo la sombra.

- **ini_col,** columna inicial de la pantalla en la cual empieza la ventana.

- **fin_col,** columna final en pantalla que ocupará la pantalla, incluyendo la sombra.

- **contador,** almacena el número de opciones que contiene.

- **menú,** es un apuntador a la dirección inicial en memoria donde se guardan los apuntadores a cada una de las cadenas de caracteres que forman las opciones a desplegar en los menús.

- **teclas,** esta variable es un apuntador a una cadena de caracteres formada con las letras que identificarán a cada una de las opciones para utilizar su método rápido de acceso (este método es explicado en detalle más adelante).

- **activo,** se utiliza para señalar si el menú está visible (activo) u oculto.contador, almacena el número de opciones que contiene.

Todos los menús se presentan con un borde que consiste en una doble linea alrededor de los mismos.

La función de agregar un registro consiste de varias subrutinas:

- *Machote_registro()*, presenta el formato de lectura de datos.

- *Inic_registro()*, limpia cualquier información que contenga el registro de memoria ram, donde se leen todos los datos.

- *Pregunta_registro()*, esta función invoca la función *machote_nuevo()*, la cual debe limpiar las partes de la pantalla que sirven para presentar el contenido de los campos de cada registro. Una vez que la ventana con los campos del registro está lista para recibir nueva información, se procede a leer de uno en uno todos los campos que componen el registro en cuestión, utilizando para ello las herramientas para verificar y validar el ingreso de la información que se describieron en el capítulo 2, secciónfig. 3.12 Se tiene la opción al final de la lectura de confirmar si los datos son correctos; en caso de no serlo, se procede a confirmar si el usuario desea volver a escribir la información o si desea cancelar su petición de agregar el registro. Si el usuario confirma su deseo de agregar el registro, entonces se invoca a la siguiente función.

- *escribe_registro()*, la cual abre el archivo de disco que contiene la base de datos para agregar información al final del archivo, escribe el nuevo registro y cierra la base de datos. Hecho lo anterior se procede a invocar a la siguiente rutina.

- *prep_indice()*, la cual crea los índices en disco y en memoria con los cuales se va a manejar la base. Con esto el sistema queda listo para seguir operando con la nueva información recién agregada y únicamente se habrá abierto la base de datos el tiempo indispensable para actualizar su información, manteniendo el criterio de seguridad.

3. 2. Manejador de Datos.

Como se explicó en el capítulo anterior esta es otra de las actividades fundamentales de SABAD, de la cual proviene su funcionalidad.

3. 2. 1. Encriptado

Debido a que con mucha frecuencia se considera a los datos de una empresa como confidenciales y valiosos, se adoptó la estrategia de codificar toda la información de los archivos de datos con una estrategia de criptografía lo suficientemente inteligente como para presentar un reto difícil a un hipotético observador no autorizado, y lo suficientemente sencilla como para que no represente un carga excesiva al manejo de la información que haga lenta la operación de los datos. De esta manera, antes de escribir cualquier información a disco se encripta, y después de leer cualquier registro en memoria, se desencripta con las funciones con ese mismo nombre.

Aunque no se pueda garantizar en un 100% la inviolabilidad del sistema de cifrado, se tiene la seguridad de que para un 90% (estimación personal) de las personas que podrían tener acceso a los datos de una empresa comercial (en diskette, o mediante una transmisión por módem) el método de rompimiento del código utilizado estará más allá de sus medios, y para el 10% restante, aunque cuenten con la capacidad técnica de

hacerlo, tal vez sería más fácil obtener la misma información por otros medios distintos de *robarse* la información sin que nadie lo sepa.

3. 2. 2. Creación y Mantenimiento de índices.

Cuando se solicita trabajar con un determinado archivo de datos, primero se verifica que éste exista, de lo contrario se procede de manera automática a crearlo (esta forma de operación es conveniente después que la información correspondiente a un determinado periodo ha sido respaldada y se desea iniciar otro periodo con datos nuevos), aunque esta forma de operación está completamente bajo el control del desarrollador de aplicaciones para tomar otra acción que desee. Una vez verificada la existencia del archivo de datos necesario[6] se procede a verificar la existencia del archivo de índice maestro[7] y, de no estar éste disponible, se procede a crearlo de manera automática, debido a que es indispensable para utilizar el archivo de datos.

La solución más común al problema del manejo de índices que se menciona en el capítulo 1, página19, es la de construir un índice para el índice, lo cual nos lleva a una estructura de índices de niveles múltiples que no es tan sencilla de programar ni tan fácil de entender para la mayoría de los programadores de aplicaciones, todo lo cual se constituyó en uno de los criterios por los cuales para la versión inicial de *SABAD* que aquí se presenta no se adoptó como estructura de indexamiento.

---

6 se verifica abriéndolo y cerrándolo para comprobar si el sistema operativo reportó algún error.

7 el cual se determina por el programador de aplicaciones

Una de las aportaciones de esta tesis consiste en el siguiente planteamiento original, al menos no se pudo localizar en ninguno de los textos de la bibliografía, y si bien no es la solución óptima y definitiva al problema de los índices de bases de datos, sí representa una alterntiva que ha demostrado su viabilidad y eficiencia y es el resultado de la formación recibida en la Maestría en Ciencias de la Computación que sustenta esta tesis. El planteamiento concreto es que: un índice puede tomar la forma de un archivo de registros donde cada uno se componga de un valor del campo llave junto con un apuntador.[8] El apuntador consiste en la posición lógica, según el ordenamiento del índice del registro en cuestión, misma posición que multiplicada por el tamaño del registro nos proporciona el desplazamiento relativo a partir del principio del archivo que hay que realizar para localizar el registro indicado. El valor que se guarde en el campo llave pudiera ser el de una llave parcial, puede ser una parte del campo que se desea que dicte el ordenamiento para aquellos casos en los que dicho campo sea demasiado grande para ser práctico utilizar todos los elementos del mismo.

Utilizando el método anteriormente descrito se implantó en *SABAD* el sistema de índices, el cual permite accesar la información de acuerdo a varios ordenamientos lógicos, sin necesidad de mover físicamente los registros del archivo que contiene los datos, sino únicamente el archivo que contiene el índice en cuestión. Si bien no es la solución más sofisticada al problema, es una solución sencilla, práctica y fácil de programar.

---

8 el término *apuntador* en este momento no se utiliza en el sentido tradicional de los lenguajes de programación (variable capaz de almacenar una dirección de memoria, sino en el sentido de un dato que nos proporciona un medio para localizar (*apunta a*) otro dato o conjunto de datos.

Debido a que el acceso a la información de disco se hace por medio de un buffer en memoria RAM es de suma importanciamantener actualizados los índices de acceso, ya que ellos guian la recuperación y modificación de la información. A este efecto se utiliza la estrategia de volver a crear los índices si se detecta su ausencia en el directorio de trabajo. De esta manera desde el momento en que se crea la base de datos se crea también el índice que la manejará. Cada vez que se agrega un nuevo registro, se realiza una modificación a un registro o se le da de baja se vuelve a construir el índice. De esta forma, se puede tener la seguridad de siempre estar trabajando con un índice actualizado.

Para la creación de los índices se utilizan dos funciones:

- *prep_arr_ram()*, la cual prepara en memoria ram el índice tomado como un arreglo de estructuras donde cada una contiene dos campos: clave y posición física (expresada como el desplazamiento relativo en número de registros a partir del inicio del archivo. En esta función se toma la precaución de cortar las primeras letras del campo llave si este resultara más grande que la capacidad del campo clave del registro de memoria ram.

- *prep_indice()*, procede a ordenar el arreglo de memoria ram creado por la función *prep_arr_ram()*, y después lo copia a disco al archivo correspondiente, de forma tal que no sea necesario volver a crear el índice la siguiente vez que se vaya a utilizar la base de datos, sino que tan sólo sea necesario copiar a memoria el ram el contenido de dicho índice.

3. 2. 3. Actualización y consulta

Esta funciones pudieron ser implantadas exactamente como se propuso en el diseño.

3. 3. Respaldos

Entre las opciones básicas que se presentan en el menú principal de las aplicaciones se tiene la de respaldar la información; esta actividad se realiza mendiante una llamada al sistema operativo para que copie a un diskette los catálogos de datos (extensión **.cat**) y los reportes del sistema (extensión **.crp**). Se recomienda respaldar los formatos de reporte también debido a que el usuario cuenta con la posibilidad de rediseñarlos a su gusto, y, por lo tanto, cabe la posibilidad de que éstos hayan cambiado desde la última vez que se hizo un respaldo.

3. 4. Reportes

La parte que maneja la creación de reportes es una de las más sofisticadas del sistema junto con la interfaz con el usuario debido a que con mucha frecuencia los programas de tipo administrativo y comercial se juzgan por la calidad y facilidad de uso de la interfaz y los reportes impresos.

Para lograr un adecuado funcionamiento del sistema con respecto a las necesidades de impresión de reportes del usuario resultó conveniente realizar un pequeño intérprete de formatos de reportes, y esta es otra de las aportaciones originales de esta tesis, el cual permite elaborar en la pantalla, mediante un procesador de textos, una muestra del reporte que se desea obtener, ubicando en las posiciones deseadas unas marcas que indiquen al intérprete dónde se desea sustituir por variables y la extensión de la sustitución (cuántos caracteres y cuántas lineas). Las fun-

ciones que manejan la impresión de reportes se encuentan agrupadas en el módulo **prn_std.c**. También se desarrolló un módulo llamado **numeros.c** el cual controla las conversiones de cantidades numéricas a su forma en palabras (útil en la impresión de cheques y facturas); si las cantidades fueran monetarias se agregará la parte fraccionaria y las palabras *nuevos pesos*.

Los formatos de reporte son archivos ASCII, encriptados por razones de privacidad, en los cuales hay dos tipos de objetos: textos y marcas de variables.

Los reportes son interpretados linea por linea, aunque su análisis léxico va tomando caracter por caracter la entrada. Los textos se transcriben al archivo de salida tal y como están en el archivo del reporte. Las marcas de variable se reconocen por la presencia del carácter especial « (ASCII 174). Cada variable es seguida de un código de identificación genérico de un byte de longitud que permitirá establecer a qué tipo de sustición se está haciendo mención (cadenas, fechas, números, cantidades monetarias, etc.); para aplicaciones sencillas, con menos de 60 variables en total, este primer código podría ser suficiente para determinar qué es lo que se deberá sustituir en la posición de la marca de variable. A este primer código le puede seguir otro más para determinar con mayor precisión qué información ha de ser sustituida (p.ej., si se desea una fecha en formato internacional corto, en formato *dd/MMM/aa*, con una abreviatura de tres letras para el mes, o si la desea expresar en palabras, etc).

Por lo general, se interpreta la longitud de la marca de sustitución de variable (delimitada por el carácter especial » (ASCII 175) como la longitud de la información a ser insertada en el reporte; esta característica permite al usuario diseñar sus propios reportes de manera sencilla, ya que únicamente debe insertar junto con sus textos fijos la marca de inicio de sustitución de variables, el o los códigos de identificación de la variable a ser sustituida, y con espacios en blanco el lugar que físicamente va

a utilizar la información, terminando con una marca especial que indique el fin de la sustitución. Así todos los demás elementos del reporte se mantienen fijos en sus lugares y la distribución de los mismos se puede realizar visualmente desde el momento de creación del reporte.

Cuando la información correspondiente a una variable tenga una longitud menor a la longitud a ocupar en el reporte, se agregan espacios en blanco al final de la sustición para llenar el espacio designado.

Por otra parte, cuando la información disponible para sustituir dentro de un reporte excede la longitud asignada dentro del reporte, se tienen dos opciones (mediante la selección de las funciones de impresión de cadenas *imp_cad()* e *imp_nombre()*) y estas son: truncar la cadena hasta donde haya llegado la longitud permitida en el reporte, o, escribir la parte que quepa en palabras completas dentro del reporte, rellenando espacios en blanco desde la última palabra que cupo hasta el final, marcando hasta donde se sustituyó para poder proseguir a partir de ahí en otra linea. Con esta última alternativa, se hace posible la impresión de reportes con columnas, donde la información de cada columna se pueda extender por varias lineas del reporte sin que se mezcle la información de una columna con la siguiente.

3. 5. Seguridad.

La seguridad del sistema se controla mediante niveles de contraseñas de acceso.

Las contraseñas se almacenan en un archivo de disco encriptado que contiene un arreglo de cadenas de caracteres donde la dimensión del arreglo indica el número de niveles de contraseñas a reconocer por el sistema. La ventaja de proceder

de esta manera es que brinda la posibilidad de que el usuario cambie su contraseña de acceso en el momento que lo desee.

La manera en que *SABAD* reconoce a los usuarios autorizados a utilizar el sistema es la siguiente:

- se solicita al usuario que escriba su clave de entrada (la cual no aparece en pantalla sino como una serie de asteriscos por razones de seguridad).

- se compara la clave recibida con las claves almacenadas en el arreglo de contraseñas leído del archivo de configuración del disco.

- de ser encontrada la clave, se activa una variable que funcionará como bandera para indicar los privilegios a los que tendrá derecho el usuario durante la sesión. Estos privilegios pueden permitir el acceso a ciertos archivos de datos y a otros no, o accesar ciertos archivos sólo para lectura, o modificar o no la estructura de los reportes, etc.

- Si la clave no fuera reconocida se emite un sonido desagradable y se presenta una pantalla de aviso de que la clave no fue válida; además se espera a que se pulse una tecla antes de salir al sistema operativo (esto con la intención de minimizar el acceso al sistema mediante un mecanismo de generación automática de claves al hacerle demasiado lento el tiempo que se dispone entre uno y otro intentos de entrar al sistema.)

La forma en que realiza *SABAD* los cambios de clave de entrada consiste en:

- validar que el usuario conozca la contraseña que se utilizó para entrar al sistema (se busca prevenir la posibilidad de que el usuario original esté ausente de su terminal y otra persona haga uso de la opción de cambio de contraseña).

- leer la nueva contraseña deseada.

- confirmar la contraseña nueva (es importante debido a que en la pantalla no se ven los caracteres que uno teclea con las contraseñas sino únicamente asteriscos y cabe la posibilidad de errores de mecanografía).

- encriptar la nueva contraseña, y

- sobreescribir en el archivo de disco la nueva clave en lugar de la antigua.

3. 6. Compresión de datos.

No se encuentra implantada en la primera versión de SABAD que aquí se presenta.

3. 7. Pruebas y depuración de errores

Desde el inicio de la aplicación más grande realizada con *SABAD* (alrededor de 37,000 lineas de código), se escribió una bitácora que ha servido para llevar el control de los errores más difíciles que se han presentado al desarrollar las aplicaciones. A continuación se explicarán estos errores con el objetivo de que el desarrollador de aplicaciones tenga un mejor conocimiento de la herramienta y, si decide modificarla, no repita las mismas equivocaciones.

El error más frecuente tiene que ver con el manejo de apuntadores, tanto de áreas de memoria RAM como de áreas de acceso al disco.

En el primer caso, se producían errores al no tomar en cuenta los límites de memoria reservada para arreglos (especialmente de los índices) y utilizando apuntadores se intentaba direccionar localidades más allá del fin del arreglo; esta condición resultaba en defectos impredecibles y esporádicos, debido a que su comportamiento dependía circumstancialmente del contenido de la memoria en el preciso instante en que sucedía la condición de direccionamiento incorrecto. Con las características antes mencionadas, ésta era una condición de error difícil de detectar (ya que se podía probar el sistema varias veces sin que falle, y fallaba hasta después de tener cierta cantidad de información para manejar) y su origen se pudo ir ubicando en aumentos a las capacidades de almacenamiento de los arreglos de índice, o al aumento de nuevas ventanas de interfaz sin el consiguiente incremento en las variables globales que controlan el número máximo de ventanas a aceptar por el sistema (posteriormente se empezó a verificar en la creación de las estructuras de ventanas de menú que no se rebasen los límites máximos declarados.

Otro problema difícil de solucionar relacionado con apuntadores de memoria RAM consistió en el paso de parámetros de tipo apuntador a funciones en las cuales se realizaba un ciclo de inspección de datos utilizando una variable local de tipo apuntador, la cual se inicializaba a la dirección recibida por el parámentro antes mencionado; el problema consistía en suponer que el contenido de la dirección a la que apunta el parámetro recibido de tipo apuntador se mantenía constante cuando, en realidad, los accesos a memoria utilizando la variable local tenían la capacidad de modificar los contenidos de la memoria, haciendo que las comparaciones de control del ciclo no produjeran el resultado esperado. En otras palabras, el hecho de manejar dos

formas distintas de accesar las localidades de memoria inducía a la confusión, ya que se perdía de vista el hecho de que a pesar de ser dos variables diferentes, estaban trabajando con el mismo objeto de datos.

Por lo que toca a los errores en el manejo de los apuntadores que hacen referencia al disco, éstos se originaban en el hecho de intentar leer o escribir a archivos de datos que previamente habían sido cerrados. Dadas las características de diseño de este sistema, las bases de datos se abren el tiempo mínimo para trabajar con ellas y luego se procede a cerrarlas, era muy común que ciertas rutinas de acceso a disco intenten hacer su trabajo sin verificar previamente que el archivo esté disponible para lectura o escritura, según el caso, lo que ocasionaba problemas, típicamente, al regresar al sistema operativo se encontraba con mucha frecuencia el error R6001 acerca de intentar asignar un valor a un apuntador nulo o no inicializado, el cual con mucha frecuencia lo que quiere decir es que se ha intentado utilizar un variable de tipo apuntador a archivo (*FILE* *) antes de que la misma haya sido inicializada. Como respuesta a este problema, se revisó exhaustivamente todas las funciones que hacen accesos al disco para que no asuman que alguna otra función les preparó el archivo para entrada/salida previamente, sino que ellas mismas se encarguen de abrir el archivo antes de consultarlo y cerrarlo inmediatamente después.

Un problema similar se suscitó con los archivos de índice, ya que como todo el sistema depende para el acceso de información de la utilización de los índices, si éstos fallan, todo el sistema falla. Por tal motivo se procedió a abrir y cerrar los archivos de índice en cada función que requiera de su lectura para instalarlos en la memoria RAM.

En caso de que no encuentre el archivo de índice buscado, se procede de forma transparente para el usuario a crearlo; de esta forma se busca garantizar la integridad de los índices en la

mayor medida posible haciendo un balance entre la velocidad de ejecución y la confiabilidad de los datos. También se procede a crear de nuevo los índices cada vez que se cambia la información de la clave de cualquier registro, se agrega o se elimina un registro.

Una situación de archivos que ocasionaron problemas surjió con el inevitable cambio de versiones de los mismos debido a que se les haya agregado nuevos campos de información o se haya modificado la capacidad de almacenamiento de los mismos. El problema, aunque sencillo, fue frecuente, por lo cual se tuvo que desarrollar programas adicionales de conversión del tipo de registros de los diferentes archivos, mismos que se fueron actualizando conforme se fueron modificando los archivos.

Otro problema que se tuvo que solucionar se refiere al manejo de los números de tipo flotante. Este problema se suscitó cuando SABAD tuvo que manejar grandes proyectos de inversión (en una empresa paraestatal) en los cuales con posterioridad a la entrega del sistema, con la influencia de la inflación y la aplicación de nuevos proyectos de inversión no previstos en el momento del diseño, las cantidades monetarias que debían ser almacenadas excedían la capacidad máxima de almacenamiento de los tipos de datos enteros -antes de la reforma monetaria de 1992 cuando se eliminaron tres ceros al peso. Entonces decidí aprovechar la mayor capacidad de almacenamiento que proporcionan los tipos de datos de punto flotante para manejar cifras monetarias de más de diez mil millones de pesos, con centavos, inclusive.

Concretamente los problemas que se me presentaron al manejar los tipos de datos de punto flotante se pueden dividir en dos: la interfaz con el usuario y el manejo interno.

En cuanto a la interfaz con el usuario, se tenía el problema de que se diseñó la interfaz de tal forma que el usuario al estar metiendo datos utilice la tecla ENTER para indicar que no desea cambiar la información que estaba previamente almacenada en un determinado campo. Sin embargo, si se proseguía con esta práctica en el caso de los números de punto flotante, se eliminaba la posiblidad de que el usuario, en realidad, estuviera tratando de dar de alta una cantidad sin enteros pero con parte fraccionaria; la respuesta a este problema consistió en requerir dos veces la tecla ENTER para no modificar el varlor anterior que pudiera tener el campo en cuestión (una vez para la parte entera, otra para la parte fraccionaria).

En cuanto al manejo interno de los números flotantes, el problema fue la falta de precisión en las centésimas y milésimas de las cantidades monetarias. Por la naturaleza misma de la implantación de este tipo de números en el compilador que yo utilizo (MS C v.6) las cantidades se manejan mediante aproximaciones a los valores originales, y en ocasiones, recién ingresada una cantidad, sin intermediar ninguna operación, la variable flotante que recibía la cantidad no la podía almacenar bien (ej. 1,234,567.80 se almacenaba como 1,234,567.799999) y, en otras ocasiones, al realizar los cálculos de sumas, resta, multiplicación y división los resultados perdían hasta un centavo. Este problema, de difícil solución debido a que se ocasiona por la naturaleza del compilador y la precisión numérica finita de la computadora, lo resolví aplicando una función de redondeo a los resultados de las operaciones matemáticas sobre cantidades monetarias. Sin embargo, considero que sería más adecuado definir un nuevo tipo de datos en el sistema tal que manejara las cantidades monetarias como un registro de dos campos (parte entera y parte fraccionaria) de manera independiente para que siempre realizara operaciones con enteros (tanto en pesos como en centavos) y mediante la aplicación de módulo 100 a los centavos se manejara el desbordamiento al peso siguiente.

Un problema que constantemente estuvo presente en el desarrollo del sistema fue el de las variables globales, mismas que, al ser compartidas por muchos módulos diferentes, con frecuencia arrastraban información inesperada para el módulo siguiente que intentaba utilizarla ocasionando diversos errores difíciles de localizar. En consecuencia, se tuvo que hacer una revisión exhaustiva para elimiar lo más posible el uso de variables globales y sustituir su uso por el paso de parámetros entre módulos, de forma tal que exista un mayor control en la información que circula entre las partes del sistema.

En el capítulo siguiente se mostrarán más concretamente cuáles son las facilidades que *SABAD* ofrece al desarrollador de aplicaciones de bases de datos y cómo mejor hacer uso de ellas. También se comenta un ejemplo real de la utilización de *SABAD*.

NOTAS

4

Funcionalidades de SABAD para el desarrollo de aplicaciones

RESUMEN DEL CAPITULO

Se ilustran los aspectos funcionales de *SABAD* destacando las facilidades que brinda a los desarrolladores de aplicaciones, especialmente las administrativas de manejo de bases de datos.

Se destaca la metodología a seguir para realizar una aplicación con *SABAD*.

La gran utilidad de SABAD se ilustra con un ejemplo concreto de manejo de datos de empleados.

El proceso de realizar aplicaciones en *SABAD* se compone de cuatro partes[1]: la realización de la interfaz con el usuario, las rutinas de almacenamiento y recuperación de la información, la emisión de reportes impresos y el procesamiento de la informa-

1 No se está hablando aquí del *ciclo de vida del software* (análisis de requerimientos, diseño, codificación, realización de manuales, pruebas, entrega, capacitación y mantenimiento), tal como lo maneja la Ingeniería de Software clásica, sino únicamente de la parte de codificación que se refiere al proceso de escribir el código fuente de las aplicaciones, el trabajo de utilizar un lenguaje de programación para expresar el diseño del programa que se desea que ejecute la computadora para lograr los objetivos de la aplicación.

ción para los fines particulares de cada aplicación. De estas cuatro partes, *SABAD* proporciona rutinas de biblioteca y en mecanismo general para solucionar las tres primeras, la cuarta parte -el procesamiento particular de los datos- queda bajo la responsabilidad del programador de aplicaciones debido a que cada aplicación es diferente precisamente en cuanto al tratamiento que le de a sus datos de entrada para producir sus datos de salida. Para lograr realizar esta etapa, el programador de aplicaciones debe utilizar el lenguaje C o ensamblador, según sea de importante la velocidad del código a obtener o la facilidad de creación de la aplicación[2].

A continuación se detalla la forma en la cual el programador de aplicaciones puede aprovechar los servicios que proporciona *SABAD*.

4. 1. Interfaz con el usuario

El programador de aplicaciones puede hacer uso de la interfaz semigráfica del sistema, en la cual se tiene la posibilidad de manejar múltiples ventanas para organizar la información, y puede definir y utilizar los menús de opciones por niveles que hacen sumamente fácil para el usuario final elegir qué es lo que desea realizar de todo lo que le ofrece la aplicación. Se cuenta con funciones que solicitan confirmación para proceder a realizar actividades críticas tales como imprimir, eliminar registros, etc. Por otro lado, se cuenta con las funciones suficientes para

---

2 En realidad, con la aparición de los compiladores optimizadores tan eficientes como los de Microsoft y otros que generan un código objeto muy compacto, es difícil justificar la utilización del lenguaje ensamblador, excepto en aquellas partes que se ejecutan una y otra vez, o código a ser utilizado en tiempo real junto con otros dispositivos periféricos muy veloces, o para animación gráfica que requiere el máximo de velocidad.

solicitar y reconocer claves de acceso, incluso con niveles diferentes de seguridad de forma que la contraseña de entrada sirva para identificar qué menús y opciones tendrá disponible el usuario. La información de entrada y visualización de datos se organiza por vistas y se agrupa convenientemente en ventanas fácilmente reconocibles por los usuarios.

Es conveniente señalar que para llevar un mejor orden en el control de las diferentes ventanas, se debe concentrar los datos de sus definiciones de posición y contenido en dos archivos de encabezamiento: **area_std.h** y **def_std.h.** En el primero de estos archivos se declara el nombre de las áreas a ser utilizadas jundo con sus coordenadas de las esquinas superior izquierda e inferior derecha utilizando instrucciones *define* del preprocesador de C.

Todos los menús se presentan con un borde que consiste en una doble linea alrededor de los mismosdiante la eliminación de la llamada a la función *dib_borde(num,AMARILL_AZL)* (aunque esta característica es controlable por el programador de aplicaciones mediante la eliminación de la llamada a la función *dib_borde(num,AMARILL_AZL)* que se encuentra en la función *menú_encadenado()*) y se presenta la lista de opciones a elegir, una por cada linea, resaltando con un color especial de fondo la primera opción inicialmente. El usuario puede entonces seleccionar otra opción utilizando las flechas de cursor hacia arriba y abajo, operación con la cual el color especial del fondo se traslada a la opción siguiente (hacia arriba o abajo); si se está en la primera opción y se pulsa la flecha de cursor que sube, la opción a remarcarse será la última y si se está en la última y se pulsa la flecha de cursor que baja, se remarcará la primera opción, a esta característica se le conoce como *menú circular.*

Se distinguirá entre las ventanas que albergan menús de opciones y las que presentan y leen información de otro tipo que no sea un menú.

El proceso de declarar un menú dentro de una aplicación es el siguiente:

- En un archivo (preferentemente de extensión *.h*, es decir, de encabezado, se declara una variable global de tipo arreglo de apuntadores a caracteres misma cuyo nombre será el nombre del menú en cuestión. Tal arreglo de apuntadores a caracteres deberá estar inicializado (se sugiere al momento de declaración) con la lista de cadenas de caracteres que denotarán los identificadores de cada opción.

- Una vez declarado e incializado el arreglo de opciones del menú en cuestión, se debe hacer una llamada a la función *crea_menú()* a la cual se le pasarán cinco parámetros: el **número con el cual se identificará al menú**, el **nombre** de la variable de tipo arreglo de apuntadores a caracteres descrita en el párrafo anterior, una cadena de caracteres que contenga las **teclas clave** para hacer uso de la selección rápida de opciones mediante una letra, y las **coordenadas x,y** de la posición en la pantalla de la esquina superior izquierda del menú.

El proceso anterior deberá repetirse por cada menú de opciones que se desee utilizar.

Para utilizar los menús de opciones simplemente se hace una llamada a la función *menú_encadenado()* a la cual se le pasará como único parámetro el número de identificación del menú deseado. Esta función devolverá la opción elegida por el usuario con un número a partir de cero o menos uno (si el usuario no seleccionó ninguna opción). Este número de opción deberá ser guardado en una variable para utilizar una construcción *switch* que se encargue de definir la acción a seguir de acuerdo con la opción solicitada por el usuario.

Debido a que, en ocasiones, es deseable que después de haberse hecho una selección, se requiera solicitar mayor información para llevarla a cabo mediante otro menú de opciones y, por lo tanto, es conveniente que la primera ventana de menú no desaparezca inmediatamente, cada ventana de menú permanece visible hasta que el programador de aplicaciones lo desea. Para hacer desaparecer una ventana de opciones se debe hacer una llamada a la función *restaura_video()* a la cual se le deberá pasar como único parámetro el número de menú que se desea desaparecer; esta función toma la información de lo que estaba debajo de la ventana de opciones y vuelve a presentar dicha información en pantalla en el lugar que ocupaba el menú que está desapareciendo, lo cual mantiene la estética e integridad de información de la pantalla.

4. 2. Mensajes al usuario

Debido a que es frecuente informar al usuario de situaciones tales como que no hay más registros activos en el archivo que está consultando, que se está imprimiendo un reporte, o que la llave buscada no se encuentra en el archivo, se cuenta con una función llamada *aviso()*, la cual recibe como un apuntador a una cadena de caracteres.

La función *aviso()* procede a presentar en pantalla la ventana designada para los avisos al usuario en color rojo para llamar la atención, y escribe el mensaje que se recibió como parámetro centrado dentro de dicha ventana, espera a que el usuario pulse una tecla para indicar que ya leyó el mensaje y restablece la información que estaba en el lugar del aviso para volver a dejar la pantalla igual que como estaba antes del mensaje.

Para el caso especial de errores catastróficos que ameritan la salida inmediata del programa (como por ejemplo, el caso de no poderse leer el archivo de configuración del sistema o haber fallado en proporcionar una contraseña válida) se tiene la función *error()*, a la cual se le deben enviar dos apuntadores a cadenas de caracteres, mismas que serán concatenadas para emitir la razón de la salida del sistema después de la pantalla de despedida con la información de dónde localizar al autor o autores de la aplicación en caso de ser necesario.

Otra función útil para comunicarse con el usuario es la que solicita información. Aquí se distinguen dos casos, cuando se hace una pregunta directa y se solicita que se responda sí o no; y cuando se solicita que se escriba una palabra clave, como una contraseña, o la llave a buscar en un archivo de datos. Para el primer caso, se cuenta con la función *confirmación()*, la cual recibe como parámetro un apuntador a una cadena de caracteres que contiene el texto de la pregunta, el cual es concatenado a el carácter "¿", y el resultado se concatena con la cadena " (S/N)?:" para centrar dicho texto en la ventana reservada para mensajes al usuario, desplegada en rojo para captar su atención, obtiene un carácter como respuesta del usuario y emite como valor de retorno la comparación de la respuesta del usuario convertida a mayúsculas con 'S' o 'Y'.

4. 2. 1. Ventanas de información

El proceso de creación y utilización de otro tipo de ventanas es similar al descrito anteriormente, sólo que más fácil.

Para inicializar una ventana se cuenta con la función *ini_area()* a la cual se le pasan cinco parámetros: las coordenadas de pantalla **x,y** tanto de la esquina superior izquierda como de la inferior derecha de la ventana, y un número para usarlo como identificador del área.

Para utilizar la ventana se hace una llamada a la función *presenta_ventana()* la cual recibe como parámetros el número de la ventana deseada y el color del borde de la misma; esta función procede a guardar la información del área de la pantalla que cubrirá la ventana, para poder volver a presentarla más tarde, y dibuja un borde alrededor de la ventana, limpiando la zona interna de la ventana y llenándola con el carácter de sombra definido para que quede lista para que el programador de aplicaciones escriba lo que desee en ella, posiblemente, lea campos de un registro, etc. y cuando termine de usar la ventana deberá hacer una llamada a la función *restaura_area()* a la cual simplemente le dará el número de identificación de la ventana que desea desaparacer de pantalla.

Todas las funciones para el manejo de menús y ventanas se encuentran en el archivo **menu.c** , las funciones que tienen que ver con el manejo de la pantalla están en **videostd.c**, y las funciones que controlan la operación del ratón están en el módulo **mouse.c**.

4. 3. Almacenamiento y recuperación de la información.

Para almacenar información de su aplicación el programador debe definir primeramente las bases de datos -y la estructura de sus respectivos registros- que va a utilizar para el almacenamiento.

La definición de la estructura de los registros se recomienda hacerla en un archivo de inclusión (con extensión .h) de manera tal que pueda ser incluído en aquellos módulos que necesiten conocer dicha estructura.

Las funciones básicas de creación de una base de datos, agregación de nuevos registros, modificación de un registro, baja lógica y física, creación y mantenimiento de índices, se explican a continuación.

4. 3. 1. Creación de una base de datos

Se recomienda programar el manejo de cada una de las bases de datos en módulos separados; esto con el fin de agrupar todas las operaciones sobre esa base de datos en un sólo código fuente, de manera de poder localizar e identificar fallas, así como mantener una simetría en el manejo de cada una de las bases de datos.

En las aplicaciones que se han desarrollado con *SABAD* se ha adoptado la política de asumir siempre que las bases de datos existen en el disco duro, con la salvedad de que si al intentar abrilrlas se observa que el archivo no se encuentra, se procede de forma automática a crearlo y a solicitar la información del primer registro al usuario. La ventaja de este enfoque radica en la sencillez del diseño y la facilidad para el usuario de respaldar los datos de un cierto periodo de su operación y, si desea volver a empezar a acumular transacciones, eliminar del disco los archivos con extensión **.cat** y dejar que el sistema automáticamente vuelva a crear otros archivos nuevos.

4. 3. 2. Agregación de nuevos registros a una base de datos

En *SABAD* se utiliza las teclas de función para que el usuario de manera directa pueda ordenar la agregación de un nuevo registro (o vista de un registro); típicamente se ha utilizado la tecla de función **F8**, aunque el programador de aplicaciones

puede definir cualquier tecla que desee con la función de agregar registro.

4. 3. 3. Baja de Registros

Se tiene la posibilidad de manejar bajas lógicas y físicas en este sistema, para ello cada estructura de registro deberá contener un campo que indique si el registro está activo o no. La condición de no estar activo corresponde a un cero en dicho campo y representa una baja lógica (el registro sigue formando parte de la base de datos pero está marcado para denotar que se le ha dado de baja).

Cuando se desea proceder a dar físicamente de baja a los registros marcados como bajas lógicas, se deberá invocar una rutina que vaya leyendo registro por registro a la base de datos original, copiando todos los registros activos en una base de datos temporal. Una vez concluída la copia de registros activos se procede a cerrar ambas bases de datos, se borra de disco el archivo original, y se renombra la base de datos temporal con el nombre del archivo original. Finalmente, se reconstruye el índice de la base de datos.

4. 4. Reportes

La impresión de los reportes tiene dos vertientes: la impresión a disco y la impresión en papel. La razón de esta separación es permitir al usuario revisar, si así lo desea, el reporte generado antes de imprimirlo. El hecho de imprimir el reporte a disco también permite hacer varias copias del mismo en la impersora sin necesidad de volverlo a generar, proceso que pudiera ser lento en los casos en l os cuales se efectúan extensos cruces de información entre archivos de datos para generar el reporte. Por otra parte, para el caso de reportes muy extensos, es preferible enviar el reporte a disco y activar una utilería del sistema opera-

tivo conocida como *spooler* para que controle proceso de impresión dejando libre la computadora para efectuar otro trabajo en tanto se imprime el reporte.

Adicionalmente a los códigos de sustitución de variables -definidas por el programador de aplicaciones- se tiene códigos para indicar la impresión de letras subrayadas, condensadas y negritas dentro de un reporte. La manera de hacer ésto es incluir una marca de inicio de cambio de modo de impresión (p. ej. «*S*» para subrayado) en la posición del reporte a partir de la cual se desea empiece a surtir efecto dicha orden, y otra marca idéntica en la posición del reporte donde se desee cancelar el efecto de impresión anterior.

También se tiene otras variables predefinidas por el sistema, tales como la fecha del día, la hora, un contador de páginas (basado en una estimación del volumen de datos a imprimir en el reporte), etc.

4. 4. 1. Forma de agregar nuevos reportes a una aplicación

Para poder utilizar la posibilidad de combinar el sistema de menús con el de reportes para hacer que el usuario pueda seleccionar más cómodamente el reporte que desee imprimir se deben seguir los siguientes pasos para agregar un nuevo reporte a una aplicación:

- Se debe declarar el nombre del menú en el arreglo de apuntadores de caracteres que identifica al menú de reportes existentes en el cual se desea agregar el reporte.

- Se debe modificar la llamada a la función *crea_menu()* para que tome en cuenta la adición del nuevo reporte y su tecla de acceso rápido.

- Se debe modificar la estructura de control de flujo *switch* para que incorpore un nuevo caso que maneje la impresión del nuevo reporte.

- Se debe crear con un procesador de textos un archivo en código ASCII con los encabezados y todos los textos fijos que se desea que contenga el reporte.

- Donde se desea imprimir algún dato variable el el formato de reporte se inserta la marca de sustitución de variables, la cual consta de las siguientes partes: un doble paréntesis angular que abre « (ASCII 174) que denota el inicio del lugar donde se sustituirá la información, un código de variable que puede ser de una o varias letras, que indicará exactamente qué variable se desea sustituir en ése lugar (la forma de hacerlo se explica más adelante), el espacio en blanco suficiente para indicar el tamaño que deberá ocupar el dato a sustituir[3], y, finalmente, un doble paréntesis angular que cierra » (ASCII 175), que indica el final de la sustitución. Por ejemplo una variable se podría indicar así: «*Nombre* ».

El formato de reporte listo deberá ser codificado con el programa de utilería que se creó como auxiliar del sistema manejador de bases de datos llamado **encripta.c**, el cual escribirá

3 el cual, si se desea, puede extenderse por más de una linea, o puede acomodarse junto con otras marcas de sustitución por columnas

la versión cifrada del reporte con el mismo nombre y con extensión **.crp.**

4. 4. 2. Forma General de operación del módulo de reportes

Para el manejo de las funciones de impresión hay un módulo llamado **prn_std.c** el cual contiene las funciones básicas de manejo de impresora, entre las que se encuentran las siguientes:

Puede reconocer las claves «*B*», «*C*», y «*S*» para activar o desactivar la impresión de letras negritas, condensadas o subrayadas; estas claves funcionan como un botón de encendido y apagado; es decir, la primera vez que se les encuentra activan la opción y la segunda vez la apagan.

También contiene funciones para la impresión precisa de números, cantidades monetarias (con separador de miles y número fijo de decimales), cadenas de caracteres y fechas (que son un tipo definido por *SABAD*) sin necesidad de utilizar las funciones de impresión de la biblioteca estándar. En particular para la definición de las funciones válidas sobre el tipo de datos fecha, se cuenta con el módulo **fechas.c**.

Una función fundamental es la llamada *imp_docto()*, a la cual se le pasa como argumento un apuntador al nombre del documento a ser impreso. Todos los reportes se escriben a disco primero y luego se envían a la impresora utilizando el programa **print.exe** del sistema operativo, para de esta manera liberar a la computadora para que haga otras cosas mientras se está imprimiendo el reporte.

Esta función *imp_docto()* realiza de manera automática los siguientes procesos:

- solicita al usuario confirmación de si la impresora está lista y en linea, a lo cual en caso de obtener una respuesta negativa aborta la impresión.

- concatena al parámetro recibido la extensión **.prn** que es la que tendrá el archivo en disco resultado de la impresión.

- avisa al usuario en pantalla que se está imprimiendo el reporte correspondiente, mencionando su nombre en pantalla para mayor información.

- manda la orden al sistema operativo para que imprima concurrentemente el archivo en disco correspondiente utilizando el programa **print.exe**.

- emite una señal auditiva al usuario al finalizar de escribir al disco para que sepa que la computadora está disponible para alguna otra operación que desee realizar.

4. 5. Procesamiento de la información

Esta es la parte que no es factible preparar de antemano y que va a depender exclusivamente de los requerimientos de la aplicación y será la responsabilidad del programador de aplicaciones.

Lo único que se requiere es escribir las funciones en C que realicen las tareas necesarias y conectarlas con los sistemas de menús de la siguiente manera: se deberá incluir una opción en la estructura del menú adecuado que sirva para identificar la opción de realizar el procesamiento de los datos tal como se describe en la parte referente al proceso de menús de la pág. 71.

El paso siguiente es insertar la operación en la función manejadora de menús, dentro de la estructura *switch*, en la cual se haga la llamada a la función procesadora de la información.

Integrando todas las partes antes mencionadas, queda lista cualquier aplicación en muy poco tiempo y con características de calidad, sin necesidad de recurrir a un DBMS de origen extranjero.

4. 6. Respaldos y restauraciones de información.

Para volver a utilizar (restaurar) la información respaldada de formatos de reportes o catálogos de datos lo único que se necesita hacer es copiarla nuevamente al directorio adecuado del disco duro. Es conveniente borrar los archivos con extensión **.ind** al restaurar catálogos de datos para forzar al sistema a recrear sus índices y, así, evitar la posibilidad de trabajar con índices que no correspondan a los datos de los catálogos.

4. 7. Seguridad.

Para ayudar a mantener la integridad de los datos, al leer la información se puede utilizar algunas funciones de verificación, integradas en el módulo **verifstd.c**, que ayudan a filtrar errores de entrada de datos por el usuario. Entre estas funciones se tiene las siguientes:

- *caracter_valido()*, recibe como parámetro un carácter y devuelve **1** si el carácter recibido se encuentra dentro del conjunto de caracteres válidos en el idioma español y **0** en caso contrario.

- *fecha_correcta()*, recibe un apuntador a una estructura de tipo fecha y devuelve **1** se la fecha es válida o **0** de no serlo, considera años bisiestos y el número máximo de días que tiene cada mes.

- *lectura_aparte()*, esta función se utiliza para leer campos de texto demasiado grandes como para caber completamente en la ventana del registro correspondiente, al utilizar esta función se abre una ventana especial superpuesta sobre la ventana del registro para leer (en varias lineas de ser necesario) la información de texto del campo en cuestión.

- *lee_char()*, *lee_long()*, *lee_num()*, *lee_numf()*, *lee_numfd()* , *lee_palabra()*, y *lee_palabra_clave()*, todas estas funciones reciben como parámetros las coordenadas de la columna y la linea donde se desea leer el dato, el tamaño máximo del dato (número de dígitos o caracteres) y un apuntador a la variable (tipo carácter, largo, entero, flotante, flotante de doble precisión, o cadena de caracteres, respectivamente). La diferencia entre *lee_palabra()* y *lee_palabra_clave()* es que esta última no hace eco a la pantalla de los caracteres que teclea el usuario sino que presenta un asterisco por cada carácter tecleado, lo cual es útil para leer contraseñas de acceso. Una ventaja de utilizar estas funciones en lugar de las de la biblioteca estándar de ANSI C, es que, además de especificar las coordenadas donde se desea realizar la lectura, se acepta que al leer un dato y pulsar la tecla ENTER, se mantenga el valor anterior que tuviera la variable, esta característica es muy útil al estar haciendo una modificación de un sólo campo de un registro, debido a que no es necesario volver a escribir la información de todos los campos, sino que se teclea ENTER en aquellos que no se desea modificar.

- *numero_valido()*, esta función verifica si un determinado carácter recibido como parámetro es aceptable para formar un número válido; además de los dígitos decimales se reconocen los espacios en blanco, el signo de menos y la coma de separación de miles, que muchos usuarios están acostumbrados a escribir cuando teclean cantidades monetarias.

- *mete_comas()*, esta función recibe como argumento un apuntador a una cadena de caracteres que representa un número válido y procede a ubicar comas como separadores de miles de la cantidad; esta es otra función pensada para la presentación de cantidades de tipo monetario especialmente.

- *redondea()*, es una función que recibe como parámetro un apuntador a una cadena de caracteres que representa un número entero y la precisión (número de posiciones decimales) con la cual se desea el redondeo y procede a redondear el número modificando la cadena de entrada para dejar ahí mismo el resultado.

- *redondeaf()*, y *redondeaf3()*, reciben como parámetro un número en punto flotante de doble precisión y devuelven como valor de retorno un número flotante de doble precisión con redondeo a dos o tres dígitos decimales respectivamente; no se hizo una sóla función a la cual se le indique la precisión para simplificar la utilización del redondeo de cantidades monetarias, que son las que con mayor frecuencia deben ser redondeadas en las aplicaciones administrativas. Por otra parte, no se modifica el parámetro de entrada debido a que con mucha frecuencia el redondeo se desea únicamente para propósitos de impresión a pantalla o impresora y no para los cálculos o el almacenamiento de las cantidades, también hace posible que la función que

llama envíe el resultado de una operación sobre números flotantes y no una variable necesariamente (p. ej. *redondeaf*(percepciones-descuentos).

4. 8. Ejemplo.

En el apéndice B se encuentra el código de un ejemplo real de una aplicación realizada con *SABAD*. El ejemplo realiza la función de almacenar y operar los datos del personal de una empresa en lo que se refiere a contrataciones, sueldos, departamento en el que trabajan, empresa y sus datos generales.

Observando el código se notará que el código es muy compacto, contiene funciones pequeñas pero eficientes, y la elegancia del diseño modular es posible gracias a las funcionalidades que se han descrito en este capítulo y que están compiladas por separado en módulos estándares de *SABAD*.

Evaluación y perspectivas

Se procederá a evaluar a *SABAD* frente a otros manejadores de bases de datos para computadoras personales que utilicen el sistema operativo MS-DOS, que son las mismas características que posee *SABAD* actualmente.

dBase es un paquete comercial que se ha difundido grandemente en todo el mundo como la herramienta más sencilla de manejo de bases de datos pseudo-relacionales[1] para computadoras personales y compatibles. Su éxito comercial fue tan rotundo en el ambiente de las microcomputadoras PC, que a la fecha el formato de grabación de sus datos en disco se toma como un estándar con el cual son compatibles un gran número de otros manejadores de bases de datos. Se puede considerar a dBase no ya como un producto comercial específico sino como un estándar de facto en los manejadores de bases de datos. En la actualidad hay bastantes sistemas comerciales que pertenecen a la familia de dBase tales como Foxpro, Clipper, Codebase, etc. De tal manera que cuando se mencione dBase la intención es hacer una referencia más amplia a este grupo de sistemas DBMS.

El sistema objeto de esta tesis también es capaz de trabajar con archivos de formato dBase, excepto que una vez leídos por el sistema y grabados en su formato propio (el cual es una combinación de código ASCII y números en binario) ya no pueden volver a ser utilizados por

1 El término pseudo-relacional no se utiliza aquí en forma despectiva; lo que se pretende indicar es que en realidad no se tiene una herramienta completamente relacional en los términos en que se explica el concepto en esta tesis, de acuerdo a las bases teóricas presentadas por Codd.

dBase, a menos que se exporten en formato sólo texto y sean importados a dBase de esa misma forma.

Se procederá a realizar una comparación de las capacidades más sobresalientes de *SABAD* y dBase, en su versión II; la razón de utilizar la versión dBase II para efectuar un contraste de la herramienta aquí presentada frente a una herramienta comercial de amplia difusión, es que el proceso de desarrollo de *SABAD* es equivalente a la primera versión comercial del producto dBase, la cual fue dBase II, por otro lado, fue precisamente dicha herramienta la que le dio inicialmente su gran popularidad y generó los recursos suficientes para financiar y justificar el desarrollo posterior que ha presentado el paquete hasta llegar hoy en día a la versión dBase IV, v.2.0, la cual, paradójicamente, no cuenta con el éxito tan rotundo e indiscutible del que gozó la versión original de dBase II en su tiempo, debido, en gran parte, por la mayor competencia de otros manejadores de bases de datos basados en el modelo presentado originalmente por el mismo dBase.

El sistema dBase II puede ser fácilmente dividido en dos grandes áreas[2]: intérprete en modo terminal e intérprete en modo de procesamiento por lotes.

De tal forma que la primera diferencia significativa entre los dos sistemas en comparación es que *SABAD* no tiene un modo de intérprete, únicamente acepta aplicaciones que necesitan ser compiladas para ejecutarse; por su parte dBase II no tenía ninguna capacidad de compilar sus programas, de ahí que uno de los primeros productos basados en el sistema dBase en tener un gran éxito comercial a expensas del mismo fuera el compilador Clipper, el cual sí tenía la capacidad de leer código en un lenguaje muy similar a dBase y com-

2 Aunque esta división no es común encontrarla en los libros que hablan de Dbase, considero que es una terminología más precisa en el ámbito de la programación profesional.

pilarlo para producir código ejecutable de manera independiente (*stand alone*) por la computadora.

En muchas ocasiones se ha debatido las ventajas y desventajas de interpretar código frente a tener código compilado. Para no caer en una tediosa e innecesaria repetición de los argumentos vertidos a favor y en contra de cada una de esas alternativas, permítaseme mencionar, tan sólo un argumento por el cual considero que es preferible trabajar con código compilado y no con código que deba ser interpretado para ejecutarse; la razón es muy simple, una vez concluido el proceso de refinar y poner a punto una rutina o un programa, por lo general, será utilizado numerosas veces sin modificaciones. Una y otra vez se le requerirá que ejecute las mismas instrucciones para el usuario; de forma tal que si dicho código fuera interpretado cada vez que se desea trabajar con él, se estaría repitiendo innecesariamente todo el trabajo que implica realizar un *análisis léxico* de la entrada (para comprobar que el código no contiene errores de escritura), el *análisis sintáctico* (para verificar que los componentes del código se relacionan entre sí de la manera prevista por la gramática del lenguaje), su *análisis semántico* (para descartar la posibilidad de que se utilicen operadores incompatibles con el tipo de sus operandos, etc.), se genere un *código intermedio* y se genere *código ejecutable* y se ejecute dicho código; el proceso anterior se repite por cada instrucción del programa, y se repite cada vez que se requiera ejecutar el programa, aún cuando ello sucediera apenas minutos después de haber sido ejecutado previamente.

Por otra parte, el código compilado, no recorre los pasos anteriores sino que tomando en cuenta que el código no haya sido modificado desde la última vez que fue ejecutado, simplemente se carga en memoria y ejecuta. Además, por lo general, en la actualidad la mayoría de los compiladores -al menos de C- optimizan el código que generan por lo cual éste es más compacto y veloz. En cambio los intérpretes al tener que realizar tantas funciones, y al no tener la visión completa del código que están interpretando, sino solamente una

visión por lineas, son poco aptos para realizar optimizaciones significativas a los códigos con los que trabajan.

Además hay que considerar que en la última versión de dBase (dBase IV) ya se puede compilar el código.

Por lo anterior, me atrevo a decir que en ese sentido *SABAD* es superior al dBase II.

Con el objeto de proseguir una comparación de los servicios que ofrece *SABAD* frente a dBase II, tomaremos como punto de referencia el lenguaje de programación de aplicaciones en modo de procesamiento de lotes de dBase II; esto es debido a que en mi sistema no hay ninguna forma de utilizar los datos en modo de intérprete de comandos.

El lenguaje de procesamiento de lotes de dBase II se basa principalmente en trece comandos o instrucciones: **Append, Close, Delete, Edit, Find, Index, Pack, Replace, Report, Skip, Sort, Total** y **Zap.**

El comando **Append** se utiliza para agregar nuevos registros a la base de datos. Esta misma función se realiza en *SABAD* con la función *escribe_registro()*, la cual se encarga propiamente de agregar un nuevo registro a un archivo de datos. La forma en que esta función realiza su trabajo consiste en abrir el archivo de datos adecuado en modo de escritura, posicionar el apuntador al final del archivo (físicamente) y agregar la información de la variable de memoria con el tipo de registro adecuado en el archivo de disco. Una vez hecho lo anterior se vuelve a cerrar el archivo de datos.

Previamente a la utilización de esta función se deberá leer la información al registro adecuado en RAM -ya sea proveniente del teclado o de un archivo en código ASCII, que podría ser un archivo exportado desde dBase II.

Posteriormente a la función *escribe_registro()* se deberá actualizar el índice (o los índices que utiliza el archivo de datos en cuestión).

El comando **Close** sirve para cerrar un archivo de datos y en *SABAD* esta operación se realiza de manera totalmente transparente para el usuario (y el programador de aplicaciones) debido a que las funciones que manipulan los datos abren y cierran sus archivos cuando se necesita, esto para evitar que en una falla de energía eléctrica queden archivos sin cerrar, con la consecuente pérdida de información.

La instrucción **Delete** en dBase se utiliza para realizar bajas lógicas. En *SABAD* se realizan las bajas lógicas mediante una llamada a la función *borra_registro()*, la cual procede a poner a cero el campo activo del registro de datos, señalando de esa forma que ese registro no está activo (baja lógica).

La instrucción **Edit** permite al usuario visualizar el contenido de un registro del archivo de datos. En *SABAD* para realizar lo mismo se realiza una llamada a la función *presenta_registro()*, la cual pone en pantalla la información del registro de memoria que corresponde a la estructura del archivo de datos con el que se esté trabajando (de ser necesario, es fácil utilizar varias pantallas en lugar de una y presentar en cada pantalla parte de la información del registro).

La orden **Find** se utiliza en dBase para localizar un registro dentro de un archivo de datos indexado utilizando una llave. Por su parte, *SABAD realiza este trabajo con la función localiza_registro()* y *loc_registro()*; en la primera de ellas se utiliza el método *quicksort* para hallar el registro deseado, en la segunda se procede a realizar un búsqueda secuencial para encontrar el registro solicitado. La razón de tener dos alternativas en cuanto a las búsquedas proviene de la necesidad manifiesta de algunos usuarios de poder tener claves idénticas, en ocasiones, para especificar el registro a buscar; otra razón de esto es que, por comodidad de los mismos usuarios, ha sido conveniente permitirles escribir tan sólo las primeras letras de la llave para que

el sistema la localice. En ambos casos, el método ***quicksort***, aunque muy veloz, no garantiza que el registro hallado sea el primero que hay en el archivo, y cuando hay la posibilidad de que la llave o pseudollave se repita, no se garantiza que con dicho método se puedan localizar todos los registros.

La orden de dBase **Index** se utiliza para organizar lógicamente un archivo de datos en base a un campo en particular, de manera alfabética ascendente o descendente. En *SABAD* esta es otra instrucción que es transparente para el usuario debido a que las funciones que manejan los archivos de datos generalmente lo hacen utilizando índices como intermediarios. Lo que debe tomar en cuenta el desarrollador de aplicaciones es qué índices se van a necesitar por la aplicación de forma tal que pueda elaborar menús de cambio de índice y le facilite al usuario la elección del ordenamiento que desea de los datos. Es pertinente hacer notar que el sistema que aquí se presenta no utiliza los índices que pudieran tener los archivos originales en dBase, en lugar de ello, procede a crear los suyos propios. La forma en que se describe ampliamente en el capítulo 2, sección 2. 3. 2. 1., de página 39.

Por su parte, la instrucción **Pack** en dBase II procede a purgar todos los registros marcados como baja lógica de forma que el resultado de esta instrucción es una nueva base de datos compacta en la cual no quedó ninguno de los registros marcados como baja lógica, realizando en la práctica las bajas físicas. En *SABAD* se utiliza exactamente el mismo mecanismo para realizar las bajas físicas, es decir, se copia el archivo de datos en otro filtrando aquellos registros que estén marcados con el campo de activo puesto a 0 (baja lógica).

La instrucción **Replace** de dBase sirve para modificar información de un registro del archivo de datos. En *SABAD*, dicho trabajo lo realiza la función *pregunta_registro()*, la cual va leyendo del usuario la información correspondiente a los campos del registro del archivo de datos en proceso. Cuando esta función se invoca con un registro que ya contiene información se procede primero a presentarla en pantalla

para permitirle al usuario utilizar la tecla ENTER para mantener intacta la información de aquellos campos que no desee cambiar; en los que sí desea hacer cambios simplemente se procese a escribir la nueva información encima de la anterior.

Report es la instrucción que permite emitir reportes desde dBase II, ya sea a pantalla, disco o impresora. En nuestro sistema la función equivalente es *imp_docto()*, la cual se describe ampliamente en la sección fig. 5.11. de la pág. 44.

Skip, por otro lado, es una instrucción de dBase para avanzar el apuntador de registro actual dentro de un archivo de datos, ya sea hacia adelante o atrás, pudiendo especificarse opcionalemente el número de registros a avanzar o retroceder. En *SABAD* se aplica un avance automático cada vez que se lee un registro de la base de datos; además, se tiene la opción de avanzar utilizando la tecla de *avance de página*, retroceder con la de *regreso de página*, pero no hay la opción de avanzar *n* posiciones de una sóla vez, en lugar de eso se recomienda a los usuarios utilizar las búsquedas, tomando en cuenta que todos los archivos de datos se manejan con un índice activo. La diferencia estriba principalmente en que el apuntador al registro actual en realidad trabaja sobre el arreglo de RAM que guarda el índice activo, y no sobre el archivo de datos, debido a que éste por lo general se abre y cierra únicamente para actualizar la información.

Sort, por su parte se utiliza para ordenar físicamente la base de datos en disco. En *SABAD* no se tiene tal instrucción debido a que la considero innecesaria debido a que si se puede organizar mucho más rápidamente la información de manera lógica utilizando índices, no hay razón de imponer al archivo físicamente un órden preestablecido; de hecho considero que esta instrucción contradice la funcionalidad de manejar electrónicamente la información en lugar de hacerlo de forma manual, ya que al tener la información en la computadora, ésta la puede presentar al usuario en el orden en el que se le necesite independientemente del orden en el cual la información se fue capturando. Por otra parte, un cambio en el ordenamiento de los datos

electrónicos no debe implicar un reordenamiento físico de la información--con los retrasos en tiempo que ello acarrea- tal y como sucede con los archivos manuales clásicos.

La orden **Total** de dBase II, se utiliza para obtener la suma de los campos numéricos de todos los registros de un archivo de datos. En nuestro sistema no existe tampoco una función genérica que se pueda aplicar para sumar dichos campos numéricos. Es posible que esta sea una debilidad de *SABAD* aunque en realidad no es muy difícil agregarle tal capacidad. En la práctica, la manipulación de la información de los registros se ha mantenido fuera del alcance de las capacidades del manejador de las bases de datos, debido a que, las más de las veces, dicha manipulación es parte del problema específico a resolver por la aplicación en particular. De esta forma, estas funciones han sido responsabilidad directa del programador de aplicaciones.

Por último, la instrucción **Zap** en Dbase produce el efecto de eliminar toda información de un archivo de datos dejando únicamente la definición del registro base. En nuestro sistema no se tiene tal función debido a que la única utilización de esta operación en las aplicaciones que he programado sería la de reinicializar algún archivo de datos al final de un año antes de empezar a manejar la información de un nuevo año. Esta necesidad en las aplicaciones se ha resuelto proporcionando al usuario la posibilidad de respaldar sus datos desde la misma aplicación de manera sencilla (lo cual, por otra parte, es una buena práctica de administración de información) y se le recomienda que cuando desee volver a empezar un archivo de datos nuevo, lo único que necesita hacer es borrar de disco el archivo en cuestión; esto funciona muy bien debido a que el sistema procede a crear -de manera transparente para el usuario- un nuevo archivo cuando no encuentra el archivo de datos que necesita utilizar.

Aventurando una crítica personal al sistema dBase en general, puedo decir que el problema fundamental de este paquete, y demás paquetes afines, es que maneja una filosofía obsoleta de manejo de bases de datos relacionales, misma que ha sido revisada y ampliada

por su autor principal, Codd, además de trabajar todo en modo de intérprete lo cual lo hace mucho más lento que el sistema que he realizado.

Sin embargo, debido al gran impacto comercial que han tenido productos basados en dicha filosofía (como lo son dBase y similares), hay una gran resistencia en el ámbito de las microempresas y pequeñas empresas a sustituir dichos paquetes con productos más modernos y capaces. Además que el costo más elevado de las mejores alternativas como lo son Informix, Oracle, DB2, etc.

La herramienta que aquí se presenta ofrece una alternativa a dicho modelo, y plantea la posibilidad de ser moldeada con gran facilidad para satisfacer las expectativas de los usuarios al tener una arquitectura abierta[3].

Finalmente, si contara con un respaldo comercial similar al que se le dio a dBase II, estoy seguro que *SABAD* evolucionaría hasta ser una herramienta no solo muy eficiente sino muy representativa del paradigma de la programación estructurada.

Limitaciones y posibilidades de crecimiento

A continuación se presentan algunas reflexiones acerca de lo que se hizo y lo que se dejó pendiente para este sistema.

3 Los códigos fuente están disponibles sin costo alguno para quien desee utilizarlos, y han sido explicados ampliamente (en clase en la Universidad del Mayab y en esta misma tesis) y utilizados para prácticas por estudiantes de la licenciatura en informática de la Universidad del Mayab.

Partiendo de que un sistema como este nunca se acaba en realidad, siempre hay posibilidades de mejorarlo y hacerlo crecer, de cambiarle la tecnología por otra más reciente, etc., es preciso hablar de las limitaciones y perspectivas de desarrollo futuro.

Incorporar esquemas de compresión de datos eficientes que operen de forma automática y transparente para el usuario, mejoraría la utilización del espacio de los medios externos de almacenamiento y apoyaría aún más la seguridad, debido a que los datos grabados de manera comprimida necesariamente implican un proceso adicional de codificación que sería una barrera para los usuarios que no poseen una sólida formación en computación, barrera que vendría a sumarse al esquema de encriptamiento de los datos (una vez descomprimida la información).

A nivel de archivos de datos, actualmente mantienen la información acerca de su estructura en un archivo de encabezado llamado *catálogo.h* de manera global, es decir, en dicho archivo está la organización de campos de cada uno de los archivos de datos. Esta podría ser una limitación a la independencia física de los datos, y se podría superar distribuyendo la información acerca del número, tipo y tamaño de los diferentes registros como un encabezado (físicamente) al principio de cada uno de los archivos de datos.

Actualmente *SABAD* carece de un lenguaje de consulta adecuado, como podría ser el SQL[4]; para hacer las consultas se debe realizar una función que ligue las bases de datos que contienen la información buscada y realice las acumulaciones o despliegues de la información según sea el caso. Sin embargo, considero que es factible agrregarle la interfaz con SQL para hacer más cómoda y estándar la manipulación de los datos.

4 SQL son las iniciales de Structured Query Language (Lenguajede Consultas Estructuradas).

En cuanto a las búsquedas dentro del índice de claves, actualmente se realizan utilizando las primeras letras de la misma; sería más conveniente modificar la función para que sea capaz de localizar subcadenas en cualquier posición de la clave. Esto haría más fácil al usuario especificar el registro deseado.

Otra limitación que afecta a la velocidad de las consultas del sistema es que sólo tiene capacidad en el buffer de datos para un registro de cada uno de los archivos de datos, utilizar un buffer de capacidad de más registros, incluso tal vez un buffer único grande para todo tipo de registros podría hacer más veloz la operación del sistema.

El sistema de manejo de índices es muy sencillo, y aunque funciona muy bien para aplicaciones simples, pudiera ser ineficiente para bases de datos enormes donde hubiera frecuentes modificaciones a los archivos; para suplir dicha deficiencia sería recomendable sustituir el módulo de manejo de índices por otro que haga uso de los **Arboles B**, los cuales aunque son más difíciles de implantar (especialmente con manejo de páginas virtuales de registros), han demostrado su eficiencia en aplicaciones de altas exigencias.

Es necesario hacer hincapié en el hecho de que *SABAD* está en un estado incipiente y aún queda mucho por hacer para convertirlo en una herramienta comercial competitiva en el mercado nacional e internacional, sin embargo, así como está ha demostrado en varias ocasiones ser una herramienta adecuada para el diseño de aplicaciones rápidas, portátiles y modulares.

Estamos en medio de un cambio de paradigma en el ámbito de la programación por lo cual considero que este sistema de administración de bases de datos podría ser revisado y complementado para ubicarse en la corriente de Orientación a Objetos y multimedia y, de esa manera, estar más *a la moda*, lo cual de ninguna manera significa que *SABAD* sea menos útil, ni eficiente por no ubicarse en la ola de la orientación a objetos, la cual aún no está completamente madura como la programación estructurada.

La programación estructurada aún proporciona elementos válidos de diseño y desarrollo de software, tal como el sistema realizado para esta tesis pretende demostrar, una prueba de lo anterior es la aparición de una revista de investigación denominada *STRUCTURED PROGRAMMING*[5]. Esta importante publicación está dirigida por un consejo editorial con un prestigio muy bien ganado que incluye a personalidades del mundo de la computación como Gustav Pomberger, David Gries, Donald E. Knuth y Niklaus Wirth, por mencionar algunos de los más conocidos.

5 *STRUCTURED PROGRAMMING* es publicada por Springer International (ISSN 0935-1183 STPGEM 14(3) 93-148 (1993)

Conclusiones

SABAD está diseñado para poder desarrollar aplicaciones de manejo de bases de datos administrativas en poco tiempo y con un alto grado de confiabilidad y seguridad.

Uno de los puntos fuertes de este sistema es su adaptabilidad, derivada del lenguaje de programación en el cual está implantado, para migrar a otras plataformas de cómputo, otros ambientes de trabajo (tales como redes o Windows[1])u otros sistemas operativos(Unix, NeXTSTEP, etc., con muy poca reprogramación (únicamente de aquellas partes que son muy dependientes del hardware[2] como son las funciones que generan los tonos de diferentes frecuencias y duración que utilizan las funciones de menús).

Este manejador de bases de datos presenta un conjunto de herramientas de programación en lenguaje C que son de ayuda para el desarrollador de aplicaciones de bases de datos, especialmente de tipo administrativo. Al estar diseñado para adaptarse al estilo personal de programar de cada quien, tiene buenas posibilidades de ser aceptado con preferencia de otros manejadores de bases de datos para computadoras personales provenientes del extranjero, como es el caso de dBase, y su familia de productos compatibles. El uso del ratón para la interfaz así como sus sistema de ventanas le dan a las aplicaciones

1 Windows es una marca registrada de Microsoft Corporation.

2 hardware, a falta de un término mejor, se utilizará para denotar aquellos aspectos de un sistema de cómputo que son físicos, es decir que ocupan un lugar en el espacio, a diferencia de los aspectos lógicos y de programación (software).

desarrolladas en este sistema una presentación moderna y agradable para el usuario final.

SABAD es un manejador de bases de datos relacional extensible. El concepto de la extensibilidad del manejador significa que los programadores de aplicaciones pueden contar con un conjunto de funciones básicas para el manejo de los datos, a manera de *kernel* o núcleo, y un conjunto de herramientas de complementación y apoyo que hacen más fácil el trabajo con las bases. Este núcleo es fácilmente adaptable al gusto y estilo personal de programar del desarrollador de aplicaciones y se puede concebir como un *shell*, o capa superior del sistema básico.

Por otra parte, el diseño estrictamente modular de *SABAD* permite combinar el poder del lenguaje de programación C y la elegancia del modelo relacional de representación de datos.

SABAD también cumple con los requisitos necesarios para poder desarrollar aplicaciones con tecnología propia sin necesidad de adquirir tecnología extranjera en software (excepto la adquisición de un sistema operativo -que de cualquier forma se tiene que comprar, por lo general- y un compilador del lenguaje C), ni pagar regalías a la compañia propietaria del software .

Notas

Apéndice A:

Ejemplo de una aplicación realizada con SABAD

/* PROGRAMA : CAT_GB.H CATALOGO DE ESTRUCTURAS PARA EL SISTEMA GBNOMINA

```
   Compilador : BORLAND C++ v.2.0 y
        Microsoft (R) C Optimizing Compiler Ver. 6
   Elaborac. : 10-Jun-1993
   Ultima rev.: 08-Feb-1994
   Autor    : Jorge E. Rodriguez Buenfil  */
/*        ------ CATALOGO ------        */
# include "cat_std.h"
struct cine       /* ver. de 22-Jul-1993 */
{ char activo;
 unsigned char
cp[6],ciudad[15],domicilio[41],imss[16],num_infonavit[12],

num_sar[13],lada[4],razon_soc[31],rfc[16],telefono[13];
 float imp_estatal;
};

struct configuracion { unsigned char
password[NUM_PWD][MAX_PWD]; };

struct detalle_nomina  /* de fecha 08-Feb-1994 */
{ unsigned char
activo,cine,num_empleado,num_nomina,num_dias_norm,
        num_domingos,tipo_nomina;
 float
fonacot,gratificaciones,horas_extra,imss,infonavit,desc_ine
ap,
incapacidad,ispt,ispt_nod,otras,otros_desc,prestamo,segur
os,
    sal_diario_base,sal_diario_integrado,sal_exento;
};

struct empleado /* Con los cambios de 15-Aug-1993 */
{ unsigned char
apellido[31],ciudad[11],domicilio[41],imss[16],
        nombre[21],rfc[16],sar[13];
 char activo,cine,depto,tipo_contrato; /* activo guarda el
núm. de empleado*/
 float sal_diario_base,sal_diario_integrado,base_sar[6];
 struct fecha f_baja,f_ingreso;
};

struct nomina
{ unsigned char activo,cine,tipo_nomina;/* el campo de
activo guardará */
 float total;              /* el núm. de nómina */
 struct fecha desde,hasta;
};
/*        ------ CATALOGO ------        */
```

/* PROGRAMA : EMPLEADO.H

```
   Compilador : BORLAND C++ v.2.0 y
        Microsoft (R) C Optimizing Compiler Ver. 6
   Elaborac. : 10-Jun-1993
   Ultima rev.: 10-Feb-1994
   Autor     : Jorge E. Rodriguez Buenfil   */
/*        ------ EMPLEADO ------          */
# include <conio.h>
# include <search.h>
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include "areas_gb.h"
# include "cat_gb.h"
# include "cat_std.h"
# include "colores.h"
# include "def_gb.h"
# include "teclas.h"
# define SUPLENTE    0
# define SUPLENTE_FIJO 1
# define PLANTA      2
# define IMP_CAR(c)  fpute(c,impresora)
extern FILE *entrada,*impresora;
extern void asigna_var(char *variable,char
num_menu,char num_tit,char tit_ant),
        aviso(char *),
        ayuda(char),
        dsncript_campo(char *),
        encript_campo(char *),
        escribe_cadena(int lin,int col,char *p,unsigned
char atrib),
        escribe_descripcion(int,int,int,char *),
        escribe_fecha(int,int,const struct fecha *,unsigned
char),
        esc_fecha_corta(int,int,const struct fecha
*,unsigned char),
        escribe_posicion(int indice,char num_are.s),
        escribe_tam_total(char num_area,int total),
        formato_numerico_f(int col,int lin,float num),
        imp_cad(char *,int),
        imp_docto(char *archivo,char margen),
        imp_nombre(char *,int,int *),
        lee_fecha(int,int,struct fecha *),
        lee_ind_adj(void),
        lee_ind_tel(void),
        lee_indice(char *,int *,int,void *),
        lee_lin(long *,FILE *),
        lee_num_f(int,int,unsigned char,float *),
        lee_pal_c(int,int,int,int,int,char *),
        letr_empresa(int,int,char),
        limpia_arreglo(char cual[],char tamanio),
        llena_bcos(int),
        phaser(char vueltas),
        posiciona(char lugar,int *i,int ultimo),
        pregunta_clave(char *llave,char modo),
        presenta_ventana(unsigned char num,unsigned
char color),
        restaura_area(unsigned char num),
        sombra(int,int,int,int),
        tam_cursor(int,int),
        titulo(char);
extern char ayudando,
        *concatena(char *,char *),
        confirmacion(char *),
        *describe_depto(char),
        *docto,
        lee_tecla_especial(void),
        num_empresa,
        *tipo_contrato[];
extern int busca_final(char *,int),
        col,
        comp_fecha(const struct fecha *,const struct
fecha *),
        get_key(void),
        i,
        tam_reg_ind;
struct empleado empleador;
struct arr_ram ind_claves_emp[50];
void abre_ind_emp(char),
   esc_modif_emp(int),
   imp_padron_emp(void),
   inic_emp(void),
   lee_emp(int),
   lee_empleado(void),
   muestra_datos_emp(char sar);
static void agrega_emp(void),
   borra_emp(int),
   dsncript_emp(void),
   encript_emp(void),
   esc_emp(void),
   imp_emp(int),
   lee_baja(void),
   lee_contrato(void),
   lee_depto(void),
   lee_ingreso(void),
   lee_salario(void),
   machote_empleado(void),
   machote_nvo(void),
   modifica_empleado(int,char),
   prepara_indice(char),
   presenta_depto(void),
   presenta_cto(void),
   presenta_emp(void),
   presenta_sar(void);
static char area_actual,
   nuevo_emp[2],
   pregunta_empleado(void),
   pregunta_sar(void);
int abre_emp(char),
   busca_emp(char),
   loc_emp(char *,char),
   tam_emp=sizeof(empleador),
   total_emp;
static int cuenta_empleados(void),
   localiza_emp(char *,char),
   prep_arr_ram(char);
/*        ------ EMPLEADO ------          */
```

/* PROGRAMA : EMPLEADO.C (CATALOGO DE EMPLEADOS)

```
Compilador : BORLAND C++ v.2.0 y
     Microsoft (R) C Optimizing Compiler Ver. 6

Elaborac. : 10-Jun-1993
Ultima rev.: 10-Feb-1994
```

```
   Autor    : Jorge E. Rodriguez Buenfil   */
# include "empleado.h"
/*       ------ abre_emp ------       */
int abre_emp(char indice)
{ FILE *arch=fopen("emp.cat","rb");
 fclose(arch);
 if (!arch) return 0;
 abre_ind_emp(indice);
 if (total_emp) return TRUE;
 else        return FALSE;
}
/*      ------ abre_ind_emp ------      */
void abre_ind_emp(char indice)
{ FILE *arch=NULL;
 char nom_indice[12];
 if (indice) strcpy(nom_indice,"emp1_?.ind");
 else       strcpy(nom_indice,"emp0_?.ind");
 nom_indice[5]=num_empresa+65;
 arch=fopen(nom_indice,"rb");
 fclose(arch);
 if (!arch) prepara_indice(indice);
 else
 { total_emp=busca_final(nom_indice,tam_reg_ind);

lee_indice(nom_indice,&total_emp,tam_reg_ind,ind_claves
_emp);
 }}
/*      ------ agrega_emp ------      */
static void agrega_emp()
{ if (area_actual!=7)
  { aviso("AGREGAR DESDE
EMPLEADOS...");return;}
 inic_emp();machote_nvo();
 if (pregunta_empleado())
 { esc_emp();
  prepara_indice(0);prepara_indice(1);
  escribe_tam_total(area_actual,total_emp);
 }}
/*      ------ borra_emp ------      */
static void borra_emp(int num)
{ if (area_actual!=7)
  { aviso("BORRAR DESDE EMPLEADOS...");return;}
  if (!confirmacion("BORRAR")) return;
  empleadox.activo=FALSE;
  esc_modif_emp(num);
  prepara_indice(1);prepara_indice(0);
  escribe_tam_total(area_actual,total_emp);
}
/*      ------ busca_emp ------      */
int busca_emp(char indice)
{ char llave[MAX_CLAVE_IND]="";
  pregunta_clave(llave,NORMAL);
  if (indice) {llave[0]=atoi(llave);llave[1]=0;}
  return localiza_emp(llave,indice);
}
/*      ------ cuenta_empleados ------      */
static int cuenta_empleados()
{ register int cuenta=0,i=0;
 int total=busca_final("emp.cat",tam_emp);
 FILE *arch=fopen("emp.cat","rb");
 while (i++<total)
 { fread(&empleadox,tam_emp,1,arch);
  if (empleadox.cine==num_empresa) cuenta++;
 }
 fclose(arch);
 return cuenta;
}
/*      ------ dsncript_empleado ------      */
static void dsncript_empleado()
{ dsncript_campo(empleadox.nombre);
 dsncript_campo(empleadox.apellido);
 if (empleadox.tipo_contrato!=SUPLENTE)
 { dsncript_campo(empleadox.ciudad);
  dsncript_campo(empleadox.domicilio);
  dsncript_campo(empleadox.imss);
  dsncript_campo(empleadox.rfc);
  dsncript_campo(empleadox.sar);
 }}
/*      ------ encript_empleado ------      */
static void encript_empleado()
{ encript_campo(empleadox.nombre);
 encript_campo(empleadox.apellido);
 if (empleadox.tipo_contrato!=SUPLENTE)
 { encript_campo(empleadox.ciudad);
  encript_campo(empleadox.domicilio);
  encript_campo(empleadox.imss);
  encript_campo(empleadox.rfc);
  encript_campo(empleadox.sar);
 }}
/*      ------ esc_modif_emp ------      */
void esc_modif_emp(int num)
{ FILE *arch;
 encript_empleado();
 arch=fopen("emp.cat","r+b"); /* Se localiza el reg.
deseado */

fseek(arch,(long)ind_claves_emp[num].pos_fis*tam_emp,0
);
 fwrite(&empleadox,tam_emp,1,arch);
 fclose(arch);
}
/*      ------ esc_emp ------      */
static void esc_emp()
{ FILE *arch;
 encript_empleado();
 arch=fopen("emp.cat","ab");
 fwrite(&empleadox,tam_emp,1,arch);
 fclose(arch);
}
/*      ------ imp_emp ------      */
static void imp_emp(int j)
{ char letras[6];
 int i=0,ind_n=0;
 if (j%60==0) { IMP_CAR(12);while (i++<3)
IMP_CAR('\n'); }
 IMP_CAR('\n');IMP_CAR('\r');
 imp_cad(itoa(empleadox.activo,letras,10),2);imp_cad("-
",3);

imp_cad(empleadox.nombre,strlen(empleadox.nombre)+1
);

imp_cad(empleadox.apellido,strlen(empleadox.apellido));
}
/*      ------ imp_padron_emp ------      */
void imp_padron_emp()
{ int j=0;
 abre_ind_emp(1);
 while(j<total_emp)
 { lee_emp(ind_claves_emp[j++].pos_fis);
  imp_emp(j);
 }}
/*      ------ inic_emp ------      */
void inic_emp()
{ register int i;
 empleadox.activo=cuenta_empleados()+1;
 nuevo_emp[0]=empleadox.activo;nuevo_emp[1]=0;
 empleadox.cine=num_empresa;

empleadox.apellido[0]=empleadox.ciudad[0]=empleadox.d
omicilio[0]=

empleadox.imss[0]=empleadox.nombre[0]=empleadox.rfc
[0]=
```

```
empleadox.sar[0]=empleadox.f_ingreso.anio=empleadox.
f_baja.anio=0;

empleadox.sal_diario_base=empleadox.sal_diario_integrad
o=0.0f;
 for (i=0;i<6;i++) empleadox.base_sar[i]=0.0f;
}
/*      ------ lee_baja ------        */
static void lee_baja()
{ escribe_cadena(l+10,col+32,"BAJA:", GRS_AZL);

escribe_fecha(l+10,col+38,&empleadox.f_baja,VIDEO_N
ORM);
 sombra(col+46,l+10,col+46,l+10);tam_cursor(3,6);
 do
 { lee_fecha(col+38,l+10,&empleadox.f_baja);}
 while
(comp_fecha(&empleadox.f_baja,&empleadox.f_ingreso)!
=1);
 tam_cursor(17,16);
}
/*      ------ lee_contrato ------        */
static void lee_contrato()
{ asigna_var(&empleadox.tipo_contrato,3,8,6);
 escribe_descripcion(col+47,l+8,8,
            tipo_contrato[empleadox.tipo_contrato]);
}
/*      ------ lee_depto ------        */
static void lee_depto()
{ switch(num_empresa)
 { case 0: case 1:
asigna_var(&empleadox.depto,num_empresa+1,7,6);break
;
   case 3: asigna_var(&empleadox.depto,17,7,6);break;
   case 5: asigna_var(&empleadox.depto,19,7,6);break;
   case 7: case 8:
   case 9:
asigna_var(&empleadox.depto,num_empresa+13,7,6);brea
k;
   case 11: case 12:
asigna_var(&empleadox.depto,num_empresa+12,7,6);brea
k;
   case 14: asigna_var(&empleadox.depto,25,7,6);break;
   case 17: case 18:
asigna_var(&empleadox.depto,num_empresa+9,7,6);break
;
   case 20: case 21:
asigna_var(&empleadox.depto,num_empresa+8,7,6);break
;
   default: empleadox.depto=0;
 }

escribe_descripcion(area[area_actual].c1+19,area[area_ac
tual].l1+8,14,
            describe_depto(empleadox.depto));
}
/*      ------ lee_emp ------        */
void lee_emp(int num)
{ FILE *arch=fopen("emp.cat","rb");
 fseek(arch,(long)num*tam_emp,0);
 fread(&empleadox,tam_emp,1,arch);
 fclose(arch);
 dsncript_empleado();
}
/*      ------ lee_ingreso ------        */
static void lee_ingreso()
{ sombra(col+30,l+10,col+31,l+10);

escribe_fecha(l+10,col+22,&empleadox.f_ingreso,VIDEO
_NORM);
 lee_fecha(col+22,l+10,&empleadox.f_ingreso);
}
/*      ------ lee_salario ------        */
static void lee_salario()
{ lee_num_f(col+33,l+9,3,&empleadox.sal_diario_base);
 do
 { escribe_cadena(l+9,col+37,"(INTEGRADO)
N$",GRS_AZL);
  sombra(col+52,l+10,col+57,l+10);
  lee_num_f(col+55,l+
9,3,&empleadox.sal_diario_integrado);
 }
 while
(empleadox.sal_diario_integrado<empleadox.sal_diario_b
ase);
}
/*      ------ loc_emp ------        */
int loc_emp(char *llave,char indice)
{ register int i=0;
 struct arr_ram *apus;
 abre_ind_emp(indice);
 if (indice)
 { do
  { if (ind_claves_emp[i].clave[0]==llave[0]) return i;
  } while (++i<total_emp);
  if (i==total_emp) return -1;
 }
 else
  apus=bsearch(llave,(void
*)ind_claves_emp,(size_t)total_emp,
          tam_reg_ind,strcmp);
 if (apus) return apus-ind_claves_emp;
 else    return -1;
}
/*      ------ localiza_emp ------        */
static int localiza_emp(char *llave,char indice)
{ int resp;
 char clave[11],mensa[41];
 resp=loc_emp(llave,indice);
 if (resp!=-1) return resp;
 else
 { indice?strcpy(mensa,concatena("NO HAY
",itoa(llave[0],clave,10))):
      strcpy(mensa,concatena("NO HAY ",llave));
  aviso(mensa);
  return -1;
 }}
/*      ------ machote_empleado ------        */
static void machote_empleado()
{ presenta_ventana(7,GRS_AZL);
 escribe_cadena(1  ,col+14,"EMPLEADO/",
GRS_AZL);
 letr_empresa(l,col+23,num_empresa);
 escribe_cadena(l+7,col+1,
 "-------------------------------------------------------",
GRS_AZL);
 escribe_cadena(l+8,col+5,"DEPARTAMENTO:",
GRS_AZL);
 escribe_cadena(l+8,col+38,"CONTRATO:",
GRS_AZL);
}
/*      ------ machote_sar ------        */
static void machote_sar()
{ presenta_ventana(11,GRS_AZL);
 escribe_cadena(l,col+10,"BASE DEL SAR/",
GRS_AZL);
 letr_empresa(l,col+23,num_empresa);
 escribe_cadena(l+2,col+20,"SAR:",  GRS_AZL);
 escribe_cadena(l+3,col+3,"SALARIO
DIARIO:",GRS_AZL);
 escribe_cadena(l+3,col+20,"(BASE) N$",
GRS_AZL);
```

```
 escribe_cadena(1+ 3,col+37,"(INTEGRADO)
NS",GRS_AZL);
 escribe_cadena(1+ 4,col+13,"MONTO BASE PARA
EL CALCULO DEL SAR",GRS_AZL);
 escribe_cadena(1+ 6,col+ 7,"ENE-FEB:",GRS_AZL);
 escribe_cadena(1+ 6,col+35,"JUL-AGO:",GRS_AZL);
 escribe_cadena(1+ 7,col+ 7,"MAR-ABR:",GRS_AZL);
 escribe_cadena(1+ 7,col+35,"SEP-OCT:",GRS_AZL);
 escribe_cadena(1+ 8,col+ 7,"MAY-JUN:",GRS_AZL);
 escribe_cadena(1+ 8,col+35,"NOV-DIC:",GRS_AZL);
}
/*      ------ machote_mvo ------      */
static void machote_mvo()
{ sombra(col+ 3,l+1,col+58,l+1);
 sombra(col+ 1,l+2,col+59,l+6);

sombra(col+19,l+8,col+37,l+8);sombra(col+47,l+8,col+
59,l+8);
 sombra(col+ 3,l+9,col+59,l+10);
}
/*      ------ machote_mvo_sar ------      */
static void machote_mvo_sar()
{ sombra(col+ 3,l+1,col+58,l+1);
 sombra(col+26,l+2,col+36,l+2);

sombra(col+29,l+3,col+35,l+3);sombra(col+51,l+3,col+
57,l+3);

sombra(col+16,l+6,col+23,l+8);sombra(col+43,l+6,col+
51,l+8);
}
/*      ------ modifica_emp ------      */
static void modifica_emp(int num,char indice)
{ char confirm;

confirm=(area_actual==7?pregunta_empleado():pregun
ta_sar());
 if (confirm)
 { esc_modif_emp(num);
  prepara_indice(!indice);prepara_indice(indice);
 }}
/*      ------ muestra_datos_emp ------      */
void muestra_datos_emp(char sar)
{ char indice=0,seguir=PGDN,otra_vez=1;
 int i=0;
 area_actual=sar?11:7;
 col=area[area_actual].c1;l=area[area_actual].l1;
 ayuda(ayudando);
 sar?machote_sar():machote_empleado();
 if (!abre_emp(indice)) agrega_emp();
 escribe_tam_total(area_actual,total_emp);
 while (otra_vez)
 { lee_emp(ind_claves_emp[i++],pos_fis);
  sar?presenta_sar():presenta_emp();
  escribe_posicion(i,area_actual);
  switch (seguir=lee_tecla_especial())
  { case ESCAPE: otra_vez=FALSE;              break;
    case F1   : ~
ayudando=!ayudando;ayuda(ayudando);--i;break;
    case F2   : if (!sar) imp_docto("list_emp",15);break;
    case F5   : modifica_emp(--i,indice);       break;
    case F6   :
agrega_emp();indice=1;i=loc_emp(nuevo_emp,1);
          if (i==-1) i=0;
          break;
    case F10    : if ((i=busca_emp(indice))==-1)
i=0;break;
    case CTRL1 : indice=!indice;
          abre_ind_emp(indice);i=0;       break;
    case CTRLY : borra_emp(--i);i=0;          break;
     default : posiciona(seguir,&i,total_emp);
  }}
 ayuda(FALSE);restaura_area(area_actual);
}
/*      ------ pregunta_empleado ------      */
static char pregunta_empleado()
{ char sigue,ok=FALSE;
 do
 { tam_cursor(3,6);
  titulo(15);lee_pal_c(1+ 1,col+
7,20,0,20,empleadox.nombre);
  titulo(16);lee_pal_c(1+
1,col+28,30,0,30,empleadox.apellido);
  tam_cursor(17,16);lee_contrato();
  if (empleadox.tipo_contrato!=SUPLENTE)
  { tam_cursor(3,6);
   escribe_cadena(1+ 4,col+ 1,
    "-------------------------------------------------------------",
GRS_AZL);
   escribe_cadena(1+ 2,col+ 5,"DIRECCION:",
GRS_AZL);
   escribe_cadena(1+ 5,col+ 5,"RFC:",GRS_AZL);
   escribe_cadena(1+ 5,col+28,"IMSS:",GRS_AZL);
   escribe_cadena(1+ 6,col+20,"SAR:",GRS_AZL);
   titulo(10);lee_pal_c(1+
2,col+16,40,0,40,empleadox.domicilio);
   titulo( 5);
   lee_pal_c(1+ 3,col+26,10,0,10,empleadox.ciudad);
   titulo( 6);
   lee_pal_c(1+ 5,col+11,15,0,15,empleadox.rfc);
   lee_pal_c(1+ 5,col+34,15,0,15,empleadox.imss);
   lee_pal_c(1+ 6,col+26,15,0,15,empleadox.sar);
   tam_cursor(17,16);
  }
  lee_depto();tam_cursor(3,6);
  if (empleadox.tipo_contrato==SUPLENTE)
  { escribe_cadena(1+ 9,col+15,"BASE PARA SAR
:",GRS_AZL);
   lee_num_f(col+33,1+
9,3,&empleadox.sal_diario_integrado);
  }
  if (empleadox.tipo_contrato!=SUPLENTE)
  { if (empleadox.tipo_contrato==PLANTA)
   { escribe_cadena(1+ 9,col+ 3,"SALARIO
DIARIO:",GRS_AZL);
    escribe_cadena(1+ 9,col+20,"(BASE) NS",
GRS_AZL);
    lee_salario();
   }
   escribe_cadena(1+10,col+13,"INGRESO:",
GRS_AZL);
   lee_ingreso();
   if (confirmacion("DADO DE BAJA")) lee_baja();
   else empleadox.f_baja.anio=0;
  }
  if (confirmacion("DATOS CORRECTOS"))
{sigue=FALSE;ok=TRUE;}
  else sigue=confirmacion("REPETIR LECTURA");
 } while (sigue);
 return ok;
}
/*      ------ pregunta_sar ------      */
static char pregunta_sar()
{ char dummy[5],sigue,ok=FALSE;
 register int i;
 machote_mvo_sar();
 escribe_cadena(1+ 1,col+
3,itoa(empleadox.activo,dummy,10),VIDEO_NORM);
 escribe_cadena(1+ 1,col+ 7,empleadox.nombre,
VIDEO_NORM);
 escribe_cadena(1+ 1,col+28,empleadox.apellido,
VIDEO_NORM);
 escribe_cadena(1+ 2,col+26,empleadox.sar,
VIDEO_NORM);
```

```
formato_numerico_f(col+33,l+3,empleadox.sal_diario_ba
se);

formato_numerico_f(col+55,l+3,empleadox.sal_diario_int
egrado);
  do
  { tam_cursor(3,6);
    for(i=0;i<3;i++)
lee_num_f(col+21,l+6+i,4,&empleadox.base_sar[i]);
    for(i=0;i<3;i++)
lee_num_f(col+50,l+6+i,4,&empleadox.base_sar[i+3]);
    if (confirmacion("DATOS CORRECTOS"))
(sigue=FALSE;ok=TRUE;)
    else sigue=confirmacion("REPETIR LECTURA");
  } while (sigue);
  return ok;
}
/*        ------ prep_arr_ram ------          */
static int prep_arr_ram(char indice)
{ FILE *arch;
  int posicion=0,vuelta;
  total_emp=busca_final("emp.cat",tam_emp);
  arch=fopen("emp.cat","rb");
  for (vuelta=0;vuelta<total_emp;vuelta++)
  { fread(&empleadox,tam_emp,1,arch);
    if (empleadox.activo AND
empleadox.cine==num_empresa)
    { if (indice)
      {
limpia_arreglo(ind_claves_emp[posicion].clave,MAX_CLA
VE_IND);

ind_claves_emp[posicion].clave[0]=empleadox.activo;
        ind_claves_emp[posicion].clave[1]=0;
      }
      else
      { dsncript_campo(empleadox.apellido);

strncpy(ind_claves_emp[posicion].clave,empleadox.apellido
,3);
        ind_claves_emp[posicion].clave[3]=0;
      }
      ind_claves_emp[posicion++].pos_fis=vuelta;
    }}
    fclose(arch);
    return posicion;
}
/*        ------ prepara_indice ------          */
static void prepara_indice(char indice)
{ FILE *arch;
  char nom_indice[12];
  total_emp=prep_arr_ram(indice);

qsort(ind_claves_emp,(size_t)total_emp,tam_reg_ind,strcm
p);
  if (indice) strcpy(nom_indice,"emp1_?.ind");
  else        strcpy(nom_indice,"emp0_?.ind");
  nom_indice[5]=num_empresa+65;
  arch=fopen(nom_indice,"wb");
  fwrite(ind_claves_emp,tam_reg_ind,total_emp,arch);
  fclose(arch);
}
/*        ------ presenta_cto ------          */
static void presenta_cto()
{ escribe_descripcion(col+47,l+8,8,
                tipo_contrato[empleadox.tipo_contrato]);
}
/*        ------ presenta_depto ------        */
static void presenta_depto()
{ escribe_descripcion(col+19,l+8,13,
                describe_depto(empleadox.depto));
}
/*        ------ presenta_emp ------          */
static void presenta_emp()
{ char dummy[5];
  machote_nvo();
  escribe_cadena(l+ 1,col+
3,itoa(empleadox.activo,dummy,10),VIDEO_NORM);
  escribe_cadena(l+ 1,col+ 7,empleadox.nombre,
VIDEO_NORM);
  escribe_cadena(l+ 1,col+28,empleadox.apellido,
VIDEO_NORM);
  if (empleadox.tipo_contrato!=SUPLENTE)
  { escribe_cadena(l+ 4,col+ 1,
    "-------------------------------------------------",
GRS_AZL);
    escribe_cadena(l+ 2,col+ 5,"DIRECCION:",
GRS_AZL);
    escribe_cadena(l+ 5,col+ 5,"RFC:", GRS_AZL);
    escribe_cadena(l+ 5,col+28,"IMSS:",GRS_AZL);
    escribe_cadena(l+ 6,col+20,"SAR:", GRS_AZL);
    escribe_cadena(l+
2,col+16,empleadox.domicilio,VIDEO_NORM);
    escribe_cadena(l+ 3,col+26,empleadox.ciudad,
VIDEO_NORM);
    escribe_cadena(l+ 5,col+11,empleadox.rfc,
VIDEO_NORM);
    escribe_cadena(l+ 5,col+34,empleadox.imss,
VIDEO_NORM);
    escribe_cadena(l+ 6,col+26,empleadox.sar,
VIDEO_NORM);
  }
  presenta_depto();
  presenta_cto();
  if (empleadox.tipo_contrato==SUPLENTE)
  { escribe_cadena(l+ 9,col+15,"BASE PARA
SAR:",GRS_AZL);

formato_numerico_f(col+33,l+9,empleadox.sal_diario_int
egrado);
  }
  if (empleadox.tipo_contrato!=SUPLENTE)
  { escribe_cadena(l+ 9,col+ 3,"SALARIO
DIARIO:",GRS_AZL);
    escribe_cadena(l+ 9,col+20,"(BASE) N$",
GRS_AZL);
    if (empleadox.tipo_contrato==PLANTA)
    {
formato_numerico_f(col+33,l+9,empleadox.sal_diario_ba
se);
      escribe_cadena(l+ 9,col+37,"(INTEGRADO)
N$",GRS_AZL);
      sombra(col+52,l+10,col+57,l+10);

formato_numerico_f(col+55,l+9,empleadox.sal_diario_int
egrado);
    }

escribe_cadena(l+10,col+13,"INGRESO:",GRS_AZL);

esc_fecha_corta(l+10,col+22,&empleadox.f_ingreso,VIDE
O_NORM);
    if (empleadox.f_baja.anio)
    { escribe_cadena(l+10,col+32,"BAJA:", GRS_AZL);

esc_fecha_corta(l+10,col+38,&empleadox.f_baja,VIDEO
_NORM);
    }}}
/*        ------ presenta_sar ------          */
static void presenta_sar()
{ char dummy[5];
  register int i;
  machote_nvo_sar();
```

```
 escribe_cadena(l+ 1,col+
3,itoa(empleadox.activo,dummy,10),VIDEO_NORM);
 escribe_cadena(l+ 1,col+ 7,empleadox.nombre,
VIDEO_NORM);
 escribe_cadena(l+ 1,col+28,empleadox.apellido,
VIDEO_NORM);
 escribe_cadena(l+ 2,col+26,empleadox.sar,
VIDEO_NORM);

formato_numerico_f(col+33,l+3,empleadox.sal_diario_ba
se);

formato_numerico_f(col+55,l+3,empleadox.sal_diario_int
egrado);
 for(i=0;i<3;i++)
formato_numerico_f(col+21,l+6+i,empleadox.base_sar[i]
);
 for(i=0;i<3;i++)
formato_numerico_f(col+50,l+6+i,empleadox.base_sar[i+
3]);
}
/*    ------ ubica_empleado ------    */
int ubica_empleado(char numero)
{ char clave[2];
 int posicion;
 clave[0]=numero;clave[1]=0;
 posicion=loc_emp(clave,1);
 if (posicion!=-1)
lee_emp(ind_claves_emp[posicion].pos_fis);
 else inic_emp();
 return posicion;
}
/*    ------ EMPLEADO ------    */
```

Apéndice B:

Bibliografía comentada

Abramson, A.
INFORMATION THEORY AND CODING
McGraw-Hill, New York, USA, 1966
Contiene ideas básicas acerca de la compresión de datos

Aho, A. V., Sethi, R., Ullman, J.D.
COMPILADORES. PRINCIPIOS, TECNICAS Y HERRAMIENTAS
Addison-Wesley Iberoamericana, USA., 1990
Proporcionó las técnicas utilizadas para la realización del pequeño intérprete de reportes.

Aho, Hopcroft, y Ullman
ESTRUCTURA DE DATOS Y ALGORITMOS
Addison-Wesley Iberoamericana, México, 1988
Se explica la teoría del uso de *árboles B* para el manejo de archivos externos.

Byron S. Gottfried
PROGRAMACION EN C
McGraw-Hill, México, 1991

Codd, E.F.
THE RELATIONAL MODEL FOR DATABASE MANAGEMENT, VERSION 2
Addison-Wesley, USA, 1990
En este libro, Codd revisa su teoría original del modelo relacional de bases de datos y presenta algunas extensiones al modelo.

Dahl O.-J., Dijkstra, E.W., and Hoare, C.A.R.
STRUCTURED PROGRAMMING
Academic Press, London, 1972
Texto clásico de programación estructurada partiendo de un modelo matemático de lo que debe ser la programación de computadoras.

Darnell, Peter A., and Margolis, Phillip E.
C A SOFTWARE ENGINEERING APPROACH
Springer-Verlag, N.Y. USA, 1991

Date, C.J.
INTRODUCCION A LOS SISTEMAS DE BASES DE DATOS
Addison-Wesley Iberoamericana, USA, 1986
Texto clásico de bases de datos escrito por uno de los más grandes propulsores de la teoría de las bases de datos relacionales.

Dijkstra, E. W.
A METHOD OF PROGRAMMING
Academic Press, London, 1989
Actualización de las ideas expresadas por el autor en la obra citada de 1972.

Dijkstra, E. W.
SELECTED WRITINGS ON COMPUTING: A PERSONAL PERSPECTIVE
Springer-Verlag, USA, 1982
Compendio de notas de viaje, opiniones filosóficas acerca de la ciencia de la computación, impresiones personales acerca de reconocidos teóricos en computación, ejemplos de soluciones elegantes a problemas complejos, así como reflexiones personales de la vida de este distinguido autor.

Elmasri, R., Navathe, S.B.
FUNDAMENTALS OF DATABASE SYSTEMS
The Benjamin Cummins Publishing Company, Inc. USA, 1989.
Excelente texto que presenta una visión moderna de las diferencias y contrastes entre los diferentes modelos teóricos de estructuras de bases de datos (tales como el relacional, jerárquico, de red, etc.)

Gasser, Morrie
BUILDING A SECURE COMPUTER SYSTEM
Van Nostrand Reinhold, N.Y. USA, 1988.
Presenta ideas acerca de cómo proteger la seguridad de los datos en un sistema computarizado de información.

Huffman, D.A. [1952]
A METHOD FOR THE CONSTRUCTION OF MINIMUN REDUNDANCY CODES
Proc. IRE, vol. 40, No. 10, pp. 1098-1101
Este es el trabajo clásico de Huffman en el que plantea su ahora famoso método para generar un código cuasi-óptimo para la representación mínima de información. Este método se puede utilizar para la compresión de datos originalmente codificados en ASCII, como es el caso de muchas bases de datos para sistemas personales.

Kernighan, B.W., Ritchie, D.M.
EL LENGUAJE DE PROGRAMACION C
Prentice-Hall Hispanoamericana, México, 1991

Korth, Henry F., Silberschatz, Abraham
FUNDAMENTOS DE BASES DE DATOS
McGraw-Hill, México, 1986
Tiene una sección muy interesante acerca del cifrado de las bases de datos para mejorar la privacía de la información.

Martin, James
ORGANIZACION DE LAS BASES DE DATOS
Prentice-Hall Internacional, España, 1977
Libro clásico de la teoría de las bases de datos.

Martin, J., and McClure, Carma
SOFTWARE MAINTENANCE. THE PROBLEM AND ITS SOLUTIONS
Prentice-Hall, USA, 1983

McGrath, Richard A.
WHERE'S THE MANUAL? PREPARING AND PRODUCING THE SOFTWARE USER'S MANUAL
Van Nostrand Reinhold, N.Y., USA, 1991
Metodolgía para la elaboración de manuales de software, donde se presenta varias alternativas de presentación de los manuales de un sistema utilizando entre ellas el paquete Ventura Publisher de Xerox.

Schildt, Herbert
C GUIA PARA USUARIOS EXPERTOS
McGraw-Hill, España, 1989
Se presenta la teoría y ejemplos de la interfase de menús por ventanas escritos directamente a RAM.

Schildt, Herbert
LENGUAJE C, PROGRAMACION AVANZADA
McGraw-Hill, México, 1988.
Se presenta un programa de ejemplo para la cuenta de la frecuencia relativa de aparición de los caracteres del alfabeto en un idioma, lo cual es un paso útil para la construcción de los códigos de Huffman para la compresión de datos.

Schildt, Herbert
PROGRAMACION EN LENGUAJE C
McGraw-Hill, México, 1988

Schulmeyer, Gordon G.
ZERO DEFECT SOFTWARE
McGraw-Hill, USA, 1990

Tenenbaum, A.M., Langsam, Y., Augenstein, M.A.
ESTRUCTURAS DE DATOS EN C
Prentice-Hall Hispanoamericana, México, 1993
Este libro contiene la mejor explicación de la formación de los códigos de Huffman que haya visto. También explica la teoría de árboles, en particular los árboles B, de forma muy comprensible.

Tsai, Alice Y.H.
SISTEMAS DE BASES DE DATOS
Prentice-Hall Hispanoamericana, México, 1990.
Este libro me proporcionó una buena guía para el diseño del sistema de administración de datos, así como una perspectiva actual del estado que guarda la teoría de bases de datos.

Wirth Niklaus
ALGORITMOS Y ESTRUCTURAS DE DATOS
Prentice-Hall Hispanoamericana, México, 1987

Yourdon, Edward
ANALISIS ESTRUCTURADO MODERNO
Prentice-Hall Hispanoamericana, México, 1993.
En esta obra el autor actualiza y revisa los conceptos vertidos en su obra clásica elaborada con Constantine diez años atrás, adecuándolos al ambiente de programación de PC's, redes y herramientas CASE.

Yourdon, E., and Constantine, L.
STRUCTURED DESIGN: FUNDAMENTALS OF A DISCIPLINE OF COMPUTER PROGRAM AND SYSTEMS DESIGN
Prentice-Hall, USA, 1979
Libro clásico de diseño esstructurado, mismo que dio la pauta para el desarrollo de gran parte de los postulados de la ingeniería de software.

Weinberg, Gerald M.
THE PSYCHOLOGY OF COMPUTER PROGRAMMING
Van Nostrand Reinhold, Canada, 1971.
Se presenta un punto de vista humano acerca de la actividad de programación de computadoras, con un énfasis particular en el proceso de depuración de programas y la interfase con el usuario, así como los procesos mentales que llevan a desarrollar buenos hábitos de programación.

Wiederhold, Gío
DISEÑO DE BASES DE DATOS, 2ª Ed.
McGraw-Hill, México, 1985.
Contiene conceptos interesantes acerca de privacía, integridad, codificación y compresión de bases de datos.

Indice Temático

A

B

C

D

E

F

G

I

L

M

N

O

P

Q

R

M

N

O

S

T

U

V

W