UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

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

INSTITUTO DE INVESTIGACION EN MATEMATICAS APLICADAS Y EN SISTEMAS

SISTEMA MANEJADOR DE PANTALLAS PARA LA ACTUALIZACION DE BASES DE DATOS

T E S I S

QUE PARA OBTENER EL TITULO DE:

MAESTRO EN CIENCIAS DE LA COMPUTACION

P R E S E N T A

DIEGO URIBE AGUNDIS

MEXICO, D. F. ENERO 1990

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

Capítulo 1 Introducción

1.1 Objetivo del sistema 2

1.2 Alternativas y selección de estrategias 2

1.3 Problemas en la actualización a través de vistas 5

1.4 Descripción global del sistema 6

Capítulo 2 Fundamentos de bases de datos

2.1 ¿Qué son las bases de datos? 14

2.2 Modelos de datos 22

2.3 Modelo Entidad-Relación 23

2.4 Modelo Relacional 31

2.5 Modelo de Red 42

2.6 Modelo Jerárquico 47

2.7 Vistas 50

Capítulo 3 Diseño del editor

3.1 Definición de la vista 61

3.2 Interfaz con el DBMS 74

3.3 Breve descripción de los módulos principales ... 76

Capítulo 4 Diseño del manejador de pantallas

4.1 Especificación de su estructura 87

4.2 Interfaz con el DBMSS 105

Capítulo 5 Casos de estudio

5.1 Informix 108

5.2 dBASE III PLUS 120

Capítulo 6 Conclusiones 130

Apéndice A. Manual del usuario

Editor 132

Manejador de la Pantalla 136

Bibliografía 139

INDICE DE FIGURAS

Capítulo 1 Introducción

1.1 Ejemplo de un reporte 7

1.2 Desarrollo de la plantilla (forma) 9

1.3 Estructura del sistema 12

Capítulo 2 Fundamentos de bases de datos

2.1 Niveles de abstracción de los datos 17

2.2 Estructura de un sistema de base de datos 21

2.3 Relaciones entre Empleado y Departamento 26

2.4 Relación uno a uno 27

2.5 Relación muchos a uno 28

2.6 Relación muchos a muchos 28

2.7 Diagrama E-R del sistema de proyectos 30

2.8 Relación EMPLEADO 33

2.9 Representación del modelo relacional 35

2.10 Parte de la base de datos proyectos 38

2.11 Procesamiento del primer ejemplo 38

2.12 Pasos del Join natural 41

2.13 Red para la base de datos proyectos 44

2.14 Base de datos proyectos 46

2.15 Un árbol para la base de datos proyectos 48

2.16 Parte de la base de datos proyectos 49

2.17 Una vista de la relación EMPLEADO 51

2.18 Base de datos para la ilustración de vistas 53

2.19 Definición e instancia de la vista ED 54

Capítulo 3 Diseño del editor

3.1 Cajas con diferente estrategia (MP) 63

3.2 Vista para EMPLEADO 65

3.3 Vista para ASIGNACIONES 66

3.4 Estructura de control de los objetos 68

3.5 Estructura de la vista 73

3.6 Uso del editor 74

3.7 Diagrama jerárquico del editor 77

Capítulo 4 Diseño del manejador de pantallas

4.1 Funcionamiento del Manejador de la Pantalla 87

4.2 Diagrama jerárquico del MP 90

4.3 Diagrama jerárquico de reportes 94

4.4 Diagrama jerárquico de edición 100

Capítulo 5 Casos de estudio

5.1 Area dinámica de datos 112

5.2 Correspondencia entre tipos de datos SQL y C .. 116

5.3 Estructura de un archivo de dBASE 121

Apéndice A. Manual del usuario

A.1 Estado inicial del editor 133

1. INTRODUCCION

Inmersos en un mundo donde el uso eficiente de la información es esencial para una adecuada toma de decisiones, es evidente que día con día un mayor número de personas abandonan sistemas de procesamiento de datos obsoletos -debido a su hermetismo y complejidad en el uso- por otros cuyas características principales sean integridad y sencillez.

No cabe duda que los Sistemas Manejadores de Bases de Datos (DBMS) han sido un gran avance en el manejo de la información, en lo que corresponde a factores tales como integridad, seguridad y redundancia. Sin embargo, los módulos que conforman la interfaz del usuario están siendo sometidos a un "refinamiento" constante hoy en día. Actualmente, adquiere mayor atención y análisis el diseño de herramientas que permitan a los desarrolladores concentrarse en la significancia (cuáles y qué) de los datos más que en su localidad (cómo y dónde están). La calidad de tales herramientas es la que permite distinguir y seleccionar a un DBMS relacional de otro. Es precisamente en este aspecto, el diseño de una herramienta (generador de formas no-procedural) que analiza la interfaz del usuario, en el cual está enfocado el trabajo.

1.1 Objetivo del sistema

Expresado de una manera sucinta y clara, el objetivo del sistema es crear vistas (views) de la base de datos en forma sencilla, ya que es a través de èstas que el usuario podrá consultar y actualizar la información contenida en los archivos de la base de datos.

Para tal efecto, el sistema proporciona un ambiente interactivo en el cual el usuario tiene el control necesario sobre su interfaz (pantalla) y al mismo tiempo requiere un mínimo de fundamentación tècnica para su uso.

Como desarrollar un sistema con tales características ofrecía diversas alternativas, èstas tuvieron que ser analizadas para posteriormente seleccionar una. El siguiente punto se ocupa de ello.

1.2 Alternativas y selección de estrategia

Huelga decir que el diseño y desarrollo de herramientas cada vez adquiere mayor importancia. ¿Por què? La respuesta es simple: su orientación es general y su objetivo particular. Esto es, quien diseña herramientas debe tener como objetivo facilitar y optimizar el trabajo de aquèllos que se encuentran inmersos en cierta tarea.

Por lo que toca al trabajo aquí expuesto, la tarea que nos ocupa es el manejo de la información, de aquí que a los que va dirigido son:

- el programador
- el administrador de la base de datos
- el usuario

En cualquier sistema generador de pantallas, hay dos componentes esenciales: uno, la **geometría** -la especificación de la forma-, y el otro, la **lógica** -la especificación de los datos a manejar así como su relación-, siendo esta la causa por la cual se clasifican en:

* Bottom-Up: su orientación es geométrica, ya que lo que se elabora es una plantilla, que será tomada por un programa de aplicación.
* Top-Down: su orientación es lógica, ya que la declaración o determinación de los datos se caracteriza por el rigor formal.

De esta manera, al construir el sistema manejador de pantallas (SMP) teníamos dos opciones:

* Enfocarlo sobre la geometría: su inconveniente es el aspecto lógico, ya que éste tendría que ser controlado por un programa de aplicación.
* Enfocarlo sobre la lógica: tendríamos que hacer uso de un lenguaje algorítmico (o extensiones) el

cual, con todo y su potencial, requerirá de un usuario experimentado.

De acuerdo a lo anterior, el SMP pretende obtener las ventajas de esas dos opciones. ¿Qué nos condujo a tal decisión? Pretender un sistema a través del cual el programador o el administrador de la aplicación pueda modificar la interfaz del usuario sin necesidad de "ensuciarse" las manos con el programa de aplicación. Esto libera al programador del desarrollo de tareas rutinarias en una base de datos (edición y consulta), permitiéndole concentrarse en la solución de un problema específico y diferente.

El significado de tal decisión dio lugar a un sistema en el que, en el aspecto geométrico, el usuario tendrá el control de la pantalla para obtener la forma que más le agrade, mientras que en el aspecto lógico le permitirá indicar fácilmente lo que desea ver -el fondo- por medio de facilidades como ventanas, predicados, agrupamiento, etc.

Por último, hacemos énfasis en que se escogió tal alternativa concientes de que la selección de una de ellas implicaba el rechazo de las otras. Ello significa -por ejemplo- que hemos renunciado al poder de una herramienta

lógica (un lenguaje procedural) a cambio de una interfaz más amigable para el usuario.

1.3 Problemas en la actualización a través de vistas

Los problemas principales en la actualización de una base de datos por medio de vistas generadas por parte del usuario son dos:

* la aparición de valores nulos
* la integridad de la base de datos

En el caso de los valores nulos el problema radica en la indefinición de aquella parte de la base de datos que "no entró" en la definición de la vista. Con este problema, para efectos de consulta, la determinación es que todas las comparaciones que involucren valores nulos serán falsas por definición; mientras que para la actualización, habrá que observar cuáles son los campos que pueden tener valores nulos. Por lo que toca al caso de la integridad, el problema estriba en la inconsistencia provocada al actualizar la base de datos cuando la definición de la vista se basa en dos o más relaciones; de aquí que la actualización de la base de datos será permitida sólo si la vista es definida en términos de una relación.

1.4 Descripción global del sistema

Para el logro de los propósitos señalados previamente, se proporciona un editor dedicado a pantallas para manipular la geometría y especificar las relaciones entre los componentes. Las ideas principales son:

* que el usuario sea responsable del control de la interfaz (pantalla).
* que la lógica (relaciones entre los datos) dentro de la pantalla sea función de la interfaz del usuario, y no de un programa de aplicación.

Para adentrarnos rápidamente en el funcionamiento del sistema, supóngase que deseamos un reporte similar al de la figura 1.1. La plantilla usada para controlar esta pantalla consiste en una lista de objetos textuales (para indicar de que se trata la información) y objetos variables (de entrada y salida o de salida). En este último caso es posible que el objeto contenga una expresión aritmética, siendo ésta evaluada cuando los valores son proporcionados, o cuando alguna variable que forme parte de ella sea modificada.

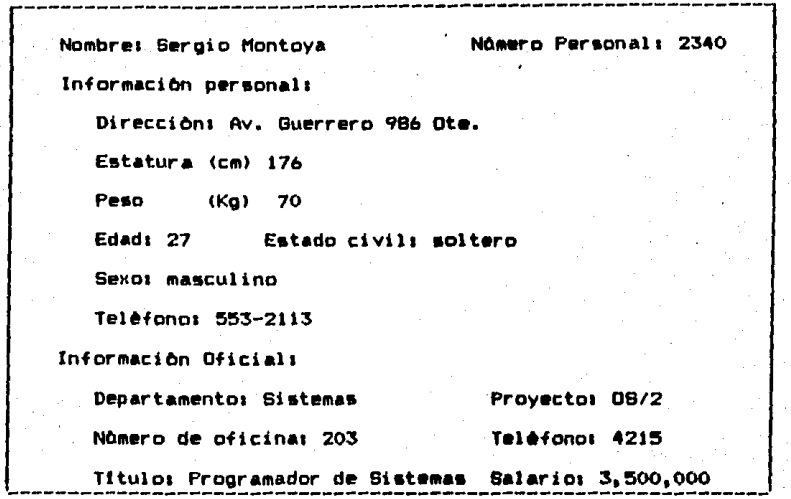


Figura 1.1 Ejemplo de un reporte

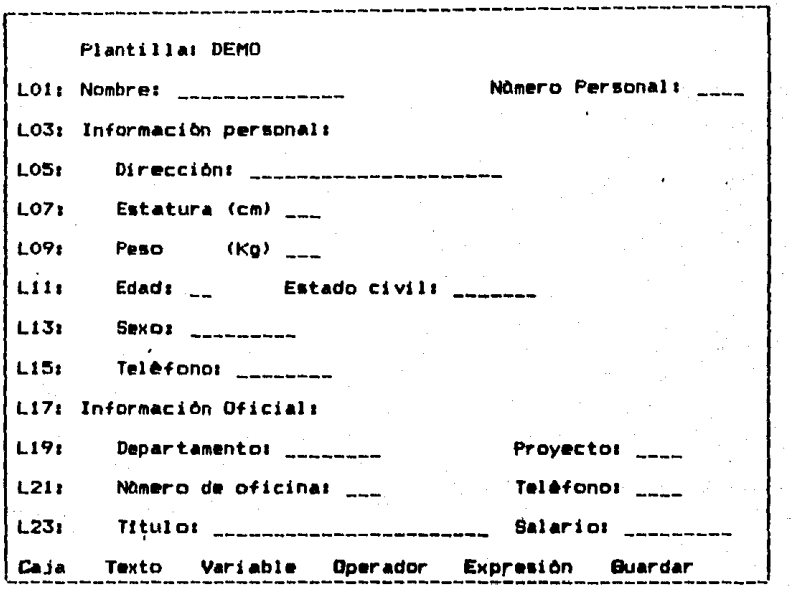
La plantilla total la constituyen 24 lineas, cada una de ellas con un conjunto de objetos ligados. Los diversos tipos de objetos son descritos internamente por sus propios descriptores, siendo algunos de ellos comunes: la longitud, la justificaciòn, video, etc.

Ahora, para lograr lo que se muestra en la figura 1.1 tenemos que definir la plantilla, y es aqui donde hacemos uso del editor (ver figura 1.2). El usuario empieza con una plantilla vacia, introduciendo los objetos deseados en las posiciones deseadas por medio del teclado y los controles del cursor. Por ejemplo, las variables son declaradas digi

tando una 'V', y aparece de inmediato una ventana que muestra los archivos que definen la vista, y que permite seleccionar el archivo; posteriormente, en otra ventana aparecerá el esquema del archivo seleccionado, y por medio de ella podremos escoger el campo deseado -obteniendo inmediatamente del sistema el tipo del campo sin necesidad de preguntar por ello al usuario. Para el logro de tal interacción es fundamental el papel del módulo encargado de la interfaz con el diccionario de datos.

Debe usarse algún mecanismo similar al de digitar una letra indicativa del objeto deseado -T, V, O, E-, de manera que el editor sea capaz de distinguir entre los diversos objetos. Cuando el usuario desea guardar la plantilla, hay una orden para este propósito.

El siguiente componente del sistema es el manejador de la pantalla, el cual recibe como entrada la vista (plantilla) que fue generada con el editor y que contiene la información geométrica así como lógica -la definición de la vista. Su funcionamiento inicia con el despliege de la plantilla, la preparación de los archivos de datos a manipular y la interacción tanto con el usuario como con el DBMS de acuerdo con el modo de trabajo (actualización o consulta) especificado en la invocación del sistema.



los campos con ___ representan variables o expresiones

Figura 1.2 Desarrollo de la plantilla por parte del usuario a través del editor de pantallas

Para un manejo eficiente, tanto de los datos como de la pantalla, atisbamos que una de las relaciones más frecuentes entre los datos es la de "muchos a uno" -lo cual significa que varios elementos comparten ciertas características. ¿Qué ofrece el sistema cuando la vista que deseamos es como la descrita? Ante tales circunstancias, el sistema permite dividir la pantalla en "cajas" -cuyo componente básico son

los objetos- y será a través del desplazamiento (scrolling) en la caja inferior como podremos apreciar mejor la relación entre los datos. En el capítulo tres ilustraremos más ampliamente esta facilidad del sistema.

El último componente del sistema es la interfaz con el DBMS. Esta interfaz complementa el funcionamiento del editor así como del manejador de la pantalla. La interfaz entre el editor y el DBMS es únicamente el diccionario de datos; mientras que entre el manejador de la pantalla y el DBMS es: el diccionario de datos y las rutinas para consultar y editar la base de datos. En lo que toca al diccionario de datos, son tres aspectos los que nos interesan:

* mostrar el **esquema** de una tabla (archivo).
* observar los **tipos** de datos.
* observar los **índices primarios**.

Una vez expuesta la descripción global del sistema, la resumimos haciendo énfasis en sus tres componentes principales:

* EDITOR: que permite la especificación de la vista con un mínimo esfuerzo y entrenamiento por parte del usuario.

* MANEJADOR de PANTALLAS: que usa la salida del editor (vista), con una interfaz simple con el usuario,

para permitir la especificación de comandos y el envío de valores a/de el manejador. La interfaz pretende evitar "comprometer" al sistema de pantallas con algún DBMS en particular.

* INTERFAZ con el DBMS: toma la información que corresponde a los archivos de la base de datos que el usuario desea observar, y los actualiza debidamente.

La figura 1.3 muestra la estructura del sistema. En ella se puede apreciar tanto los componentes como las relaciones entre los mismos. Por último, el uso del sistema se efectúa en dos etapas:

a. crear la vista a través del editor.

b. invocar al manejador de la pantalla con la vista y comandos deseados (ver figura 1.3).

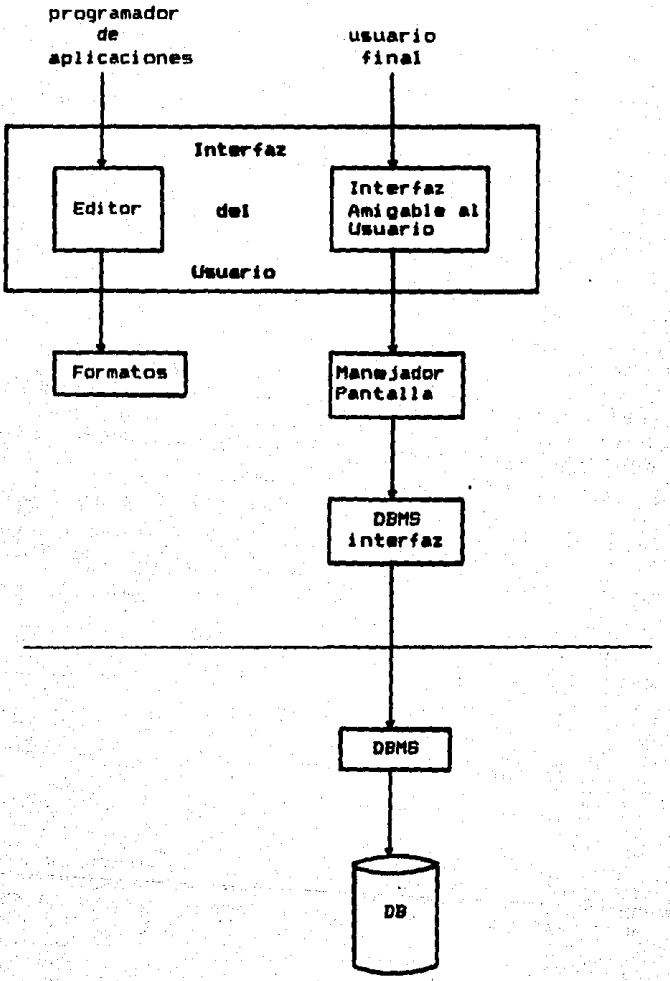


Figura 1.3 Estructura del sistema

2. FUNDAMENTOS DE BASES DE DATOS

Este capítulo pretende ubicar al lector en el contexto teórico básico que le permita entender y juzgar el trabajo aquí presentado. Para tal efecto, empezaremos dando respuesta a una pregunta primordial: ¿qué se entiende por una base de datos?, para posteriormente continuar con la presentación de los modelos de datos -herramientas para la descripción de los datos así como de las relaciones entre ellos-, del modelo relacional -por la aceptación tan amplia que tiene actualmente y por haber realizado el trabajo con dos manejadores de este modelo de datos-, para terminar con la exposición de los problemas que aparecen al actualizar una base de datos a través de vistas -tema central de la tesis.

Sin embargo, antes de iniciar con el primer punto -¿qué son las bases de datos?-, no está por demás el hecho de mencionar algunas de las causas que dieron lugar a un estudio más sistematizado para el manejo de los datos. Anteriormente, el control de los datos de las organizaciones consistía en un conjunto de archivos y programas de aplicación escritos en algún lenguaje de alto nivel -COBOL, ALGOL, etc. Conforme las organizaciones crecían, se hacían nuevos programas de aplicación y se definían nuevos archivos, de aquí que, al paso del tiempo, no era difícil encontrarse con una gran cantidad de archivos con formatos diferentes y con programas de aplicación similares en lenguajes también

diferentes. Este ambiente descrito en forma sucinta se conoce como **sistema de procesamiento de archivos.** Algunas de las principales desventajas de este sistema son:

* Redundancia de datos e inconsistencia -siendo ésta provocada por aquélla.
* Dificultad en el acceso a los datos -no hay un lenguaje para la manipulación de los datos.
* Usuarios múltiples -carencia de coordinación entre los programas de aplicación al accesar los datos.
* Actualización y seguridad -ante la gran cantidad de programas de aplicación, tanto en el mantenimiento como la integridad del sistema se van tornando cada vez más difíciles.

Otras fuentes mencionan más desventajas pero pienso que son consecuencia de las señaladas anteriormente. Ante los problemas tan serios que presentaba el sistema anterior, surgen las **bases de datos.**

2.1 ¿Qué son las bases de datos?

Consideremos una institución educativa que tiene una gran cantidad de datos no sólo a manejar sino también a almacenar por periodos de tiempo grandes. Algunos de los datos a considerar son: estudiantes, personal académico, personal administrativo, instalaciones, mobiliario, etc. Algunas de las relaciones entre los datos son: cursos (materia, profesor),

grupos (curso, salón, hora), etc.

Datos tales como los descritos anteriormente son los que conforman lo que conocemos como **bases de datos (DB)**. El programa que permite que una o más personas use o modifiquen estos datos se denomina **sistema manejador de bases de datos (DBMS)**. El papel principal del DBMS es permitir que el usuario interactúe con los datos en términos abstractos -alto nivel-, y no en los términos en los cuales la computadora almacena los datos. A continuación se definen los conceptos básicos de las DB.

Abstracción de los datos

La abstracción de los datos es necesaria en vista de que se usan estructuras de datos complejas para la representación de los datos en la DB y, puesto que el usufructo de los datos es llevado a cabo por personas no expertas en computación, es imprescindible ocultar tal complejidad. ¿Cómo se logra esto? Definiendo varios niveles de abstracción a través de los cuales la DB pueda ser observada.

- Nivel físico: es el nivel más bajo de abstracción en el cual se describe cómo son almacenados los datos. Es en este nivel en el que la complejidad de las estructuras de los datos es mayor.
- Nivel conceptual: es el nivel de abstracción intermedio en el cual se describe cuáles son los datos y

las relaciones entre los datos que deberán ser almacenados. En este nivel, a través de un número pequeño de estructuras relativamente simples, se especifica toda la DB. Aunque las estructuras simples pueden involucrar estructuras complejas del nivel físico, éstas permanecerán ocultas al usuario. El responsable directo de este nivel es el administrador de la base de datos.

* Nivel de vista: es el nivel más alto de abstracción, en el cual se describe parte de la DB. La finalidad de este nivel es facilitar la interacción del usuario con la DB, de aquí que sea el nivel de mayor uso.

Los tres niveles de abstracción de los datos se ilustran en la figura 2.1. Para apreciar más claramente la diferencia entre los niveles, tomemos la siguiente declaración de un arreglo en Pascal:

```
type
    matriz = array [1..n, 1..m] of integer ;
var
    A : matriz ;
```

La declaración corresponde al nivel conceptual, mientras que en el nivel físico podemos ver el arreglo como un bloque de localidades de memoria consecutivas, con A[i,j] en la loca

lidad $a_0 + 4(m(i - 1) + j - 1)$. Una vista del arreglo puede ser una función f(i) que obtiene la suma desde j = 1 hasta m de las A[i,j] -suma del iésimo renglón.

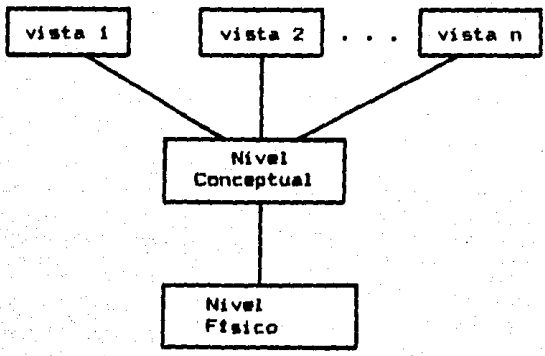


Figura 2.1 Niveles de abstracción de los datos

Instancias y esquemas

Para definir los conceptos señalados arriba, habrá que recordar que la información contenida en la DB cambia constantemente debido a las inserciones o eliminaciones de la misma. De aquí que una **instancia** de la DB viene a ser toda la información almacenada en la DB en un momento dado, mientras que el **esquema** de la DB es lo que corresponde a todo el diseño. Para ilustrar la diferencia entre los conceptos anteriores, nuevamente hacemos uso de la analogía con los conceptos de tipos y variables de Pascal. De acuerdo con las

declaraciones presentadas en el tema anterior, observamos que el concepto de esquema corresponde a la definición del tipo en Pascal, mientras que el concepto de instancia corresponde al valor que tiene una variable en un instante dado.

Independencia de los datos

Las abstracciones de la figura 2.1 -desde la visual a la física, pasando por la conceptual- dan lugar a la independencia de los datos, entendiendo por ésta la capacidad de modificar la definición de un esquema en un nivel sin afectar la definición del esquema que se encuentra en el nivel superior. Hay dos niveles de independencia de datos:

- **Física:** la capacidad de modificar el esquema físico sin dar lugar a que los programas de aplicación sean reescritos.
- **Lógica:** la capacidad de modificar el esquema conceptual sin dar lugar a que los programas de aplicación sean reescritos.

Mientras que las alteraciones en el nivel conceptual son más comunes -anexar o eliminar campos-, las modificaciones en el nivel físico son raras, y su objetivo es optimizar la velocidad de ejecución. Por otra parte, la independencia de datos lógica es más difícil de lograr, ya que los programas de aplicación dependen de la estructura lógica de los datos

que manejan.

Lenguajes de las bases de datos

Para la especificación y manejo de los datos, el DBMS contiene dos lenguajes:

* **Lenguaje de definición de datos (DDL):** permite especificar el esquema de la DB por medio de un conjunto de declaraciones que describen las entidades y sus relaciones, en términos de un modelo de datos particular. La compilación de las declaraciones efectuadas con el DDL arrojan como resultado un conjunto de tablas que serán almacenadas en un archivo especial denominado diccionario de datos. Como podrá observarse, este archivo especial contiene **metadatos:** datos acerca de datos.
* **Lenguaje de manejo de datos (DML):** permite a los usuarios accesar los datos de acuerdo al modelo de datos que los organizó. Hay dos tipos de DML:
  + Procedural: requiere que el usuario especifique los datos que necesita y la manera de obtenerlos.
  + No-Procedural: se requiere que el usuario especifique los datos que necesita sin determinar la manera de obtenerlos.

Estructura del sistema

Con lo expuesto anteriormente, llegamos al punto en el cual podemos tener una perspectiva global de todo lo que conforma un sistema de bases de datos. Como se había señalado, un DBMS tiene como objetivo principal proporcionar un ambiente adecuado en el cual la **eficiencia** y la **comodidad** se conjuguen para hacer del manejo de la información contenida en la DB una labor sencilla, segura y rápida. Para el logro de tal objetivo, el DBMS consta de una colección de componentes interrelacionados, los cuales se muestran en la figura 2.2 y se presentan a continuación:

- Manejador de archivos (MA): controla la asignación de espacio en disco y las estructuras de datos usadas para representar la información.
- Manejador de la DB (MDB): es la interfaz entre los programas de aplicación así como también entre las preguntas de los usuarios y el MA. Se encarga de expresarle las preguntas al MA en términos de operaciones sobre los archivos, y no sobre las estructuras de datos abstractas, que son las que describen la DB.
- Procesador de preguntas: es el compilador de las preguntas, cuya salida no es el lenguaje máquina sino una serie de instrucciones que el MDB interpreta. La compilación de la pregunta podrá ser optimizada ya que la velocidad de la respuesta depende de los

pasos a seguir por parte del sistema.

* Compilador del DDL: convierte las declaraciones hechas con el DDL en un conjunto de tablas que contienen metadatos. Estas tablas son almacenadas en el diccionario de datos.

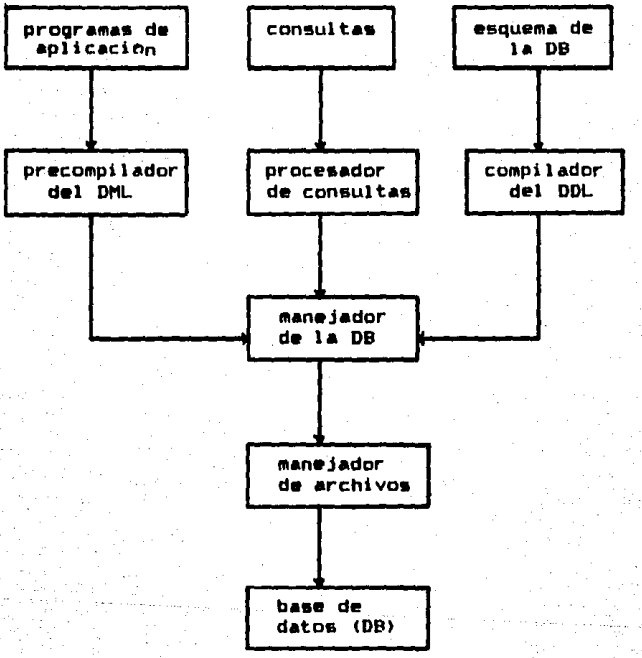


Figura 2.2 Estructura de un sistema de base de datos

2.2 Modelos de datos

Para definir la estructura de la DB necesitamos un modelo de datos. Ahora bien, ¿què es un modelo de datos?: un conjunto de herramientas conceptuales para la descripción de los datos, de las relaciones entre los datos, de la semántica de los datos y de las restricciones de los datos. Existen varios modelos de datos, los cuales han sido divididos en tres grupos: modelos lògicos basados en objetos, modelos lògicos basados en registros, y modelos fisicos.

2.2.1 Modelos lògicos basados en objetos

Los modelos lògicos basados en objetos son usados para la descripción de los datos en los niveles conceptual y de vista. Las principales caracteristicas de estos modelos son tanto su capacidad de representar situaciones reales como que pueden hacer explicitas las restricciones de los datos. El modelo de datos a observar aquí será el de mayor aceptación actualmente: el modelo Entidad-Relación (E-R).

2.2.2 Modelos lògicos basados en registros

Los modelos lògicos basados en registros son usados para la descripción de los datos en el nivel conceptual y de vista. La aplicación principal de estos modelos es la especificación de la estructura lògica de toda la DB y la descripción a un alto nivel de la implantación de la DB. Sin embargo, su principal desventaja radica en no poder hacer ex

plícitas las restricciones que ocurren entre los datos. En este trabajo veremos los tres modelos de datos mayormente usados: Relacional, Red y Jerárquico.

2.2.3 Modelos de datos físicos

Los modelos de datos físicos son usados para describir datos en el nivel más bajo. En vista de que el interés principal de estos modelos se encuentra en la implantación de la DB a un nivel bajo, con la finalidad de controlar factores muy importantes como el espacio y el tiempo, no serán observados en este trabajo.

2.3 Modelo Entidad-Relación (E-R)

Este modelo se fundamenta en que la representación de situaciones reales se hacen por medio de un conjunto de objetos básicos denominados **entidades** y sus **relaciones.**

Entidades

Una entidad es un objeto que existe y es distinguible de otros objetos. Por ejemplo, cada persona o cada automóvil. Un grupo de entidades similares conforman un conjunto de entidades. Por ejemplo, todas las personas o todos los automóviles.

Atributos y llaves

Las propiedades o características de las entidades se conocen como **atributos**. En un cojunto de entidades cada entidad tiene para cada uno de sus atributos un valor asociado, el cual se obtiene de un dominio de valores conocido como el dominio del atributo. Por otra parte, un atributo o un conjunto de atributos cuyos valores identifiquen a cada entidad en un conjunto de entidades es denominado una llave para ese conunto de entidades. Para ilustrar tales conceptos, de aquí en adelante trataremos con tres conjuntos de entidades de una institución cuyo objetivo es administrar eficientemente a su personal y sus proyectos:

* Empleado, es el conjunto de todos los investigadores de la institución. Cada empleado es descrito por los atributos ENUM (número del investigador), ENOMBRE (nombre del investigador), EFECHAIN (fecha de ingreso), y ESALARIO (salario).
* Departamento, es el conjunto de todos los departamentos de la institución. Cada departamento es descrito por los atributos DNUM (número del departamento) y DNOMBRE (nombre del departamento).
* Proyecto, es el conjunto de todos los proyectos de la institución. Cada proyecto es descrito por los atributos PNUM (número del proyecto), PNOMBRE (nombre del proyecto), PFECHAIN (fecha de inicio), PFECHATER (fecha de terminación) y PDINERO

(presupuesto).

Algunos de los dominios de los atributos son los números enteros (para atributos como pnum, dnum, pdinero), las cadenas (para atributos como pnombre, dnombre), las fechas (para atributos como pfechain, pfechater), etc. Las llaves serían enum (número de identificación del empleado), dnum (número del departamento) y pnum (número del proyecto).

Relaciones

Una relación es una asociación entre varias entidades. Por ejemplo, podemos asociar a un proyecto con un departamento. Un conjunto de relaciones son todas aquellas relaciones del mismo tipo. Por ejemplo, todos los proyectos de un departamento. Más formalmente, un conjunto de relaciones es una relación matemática entre n >= 2 conjuntos de entidades, las cuales pueden ser iguales. Si E_1, E_2, ..., E_n son conjuntos de entidades, entonces un conjunto de relaciones R es un subconjunto de

$$\{(e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

donde $(e_1, e_2, \ldots, e_n)$ es una relación. Para apreciar esto, obsérvese la figura 2.3.

Una relación puede también tener atributos descriptivos. Por ejemplo, INGR_DEP podría ser un atributo del conjunto de

relaciones EmpDep. Esto especifica la fecha en que ingresó el empleado al departamento. La relación EmpDep de (E03, C11) es descrita por ((INGR_DEP, 02/01/85)), lo cual significa que la fecha en que Montoya ingresó al departamento de graficación fue el 2 de enero de 1985.

E M P L E A D O			
ENUM	ENOMBRE	EFECHAIN	ESALARIO
E01	Flores	05/06/84	2000
E02	Ramírez	02/04/82	1750
E03	Montoya	09/21/84	1500
E04	Iepes	07/12/86	1300
E05	Cortez	11/11/85	1300

DEPARTAMENTO	
DNUM	DNOMBRE
C10	Sistemas
C11	Graficación
C12	Redes

Figura 2.3 Conjunto de relaciones entre los conjuntos de entidades Empleado y Departamento

Funcionalidad

Para el diseño adecuado de una DB es necesario clasificar las relaciones de acuerdo a cuántas entidades de un conjunto de entidades pueden ser asociadas con cuántas entidades de otro cojunto de entidades. Tenemos tres casos:

* Uno a uno: es la más simple y menos común. Significa que para cada entidad de ambos conjuntos hay a lo más un elemento asociado del otro conjunto (ver la figura 2.4)

* **Muchos a uno:** una entidad en el conjunto de entidades E_2 es asociada con cualquier número de entidades en el cojunto E_1, pero cada entidad en E_1 es asociada con a lo más una entidad en E_2. (ver figura 2.5)
* **Muchos a muchos:** en este caso no hay restricciones en cuanto al número de entidades relacionadas, lo cual significa que una entidad en E_1 puede estar asociada con cualquier número de entidades de E_2 y a su vez una entidad en E_2 puede estar asociadacon cualquier número de entidades de E_1. (ver firgura 2.6)

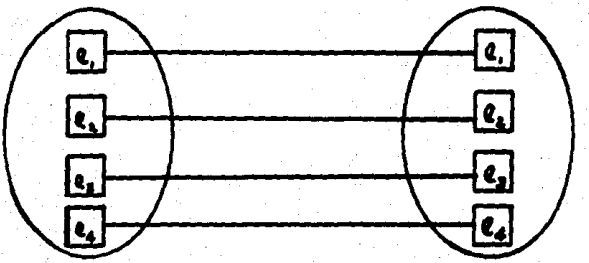


Figura 2.4 Relación uno a uno

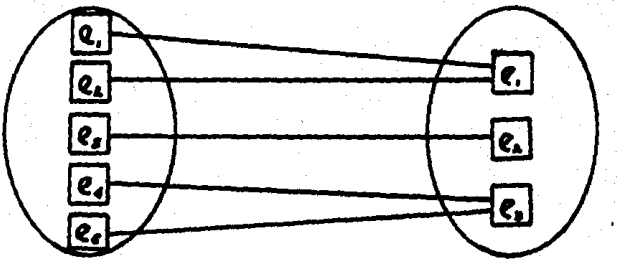


Figura 2.5 Relación muchos a uno

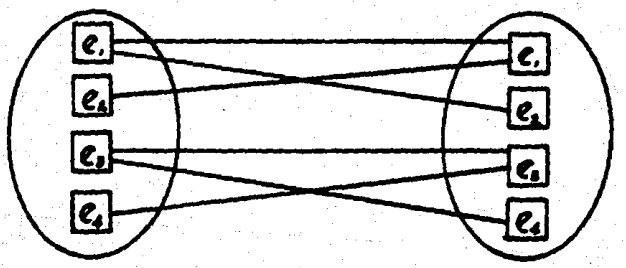


Figura 2.6 Relación muchos a muchos

Como ejemplo de lo presentado, observemos el caso de la educación. Si la enseñanza es privada, la relación que podemos observar entre Alumno-Profesor es uno a uno. Por otra parte, en la etapa de la enseñanza básica (primaria), la relación que atisbamos entre Alumno-Profesor es muchos a uno. Por último, en la enseñanza superior la relación entre Alumno-Profesor es muchos a muchos.

Diagrama entidad-relación

La estructura lógica de toda la DB puede ser expresada en forma gráfica por medio de un diagrama E-R, que consta de los siguientes componentes:

* **Rectángulos**, que representan conjuntos de entidades.
* **Elipses**, que representan atributos, siendo éstos ligados a sus respectivos conjuntos de entidades por medio de líneas.
* **Rombos**, que representan conjuntos de relaciones, siendo éstas señaladas por medio de líneas dirigidas y no dirigidas. En el caso de la relación "a uno" se usa una línea dirigida, mientras que en la relación "a muchos" se usa la línea no dirigida.

En vista de lo anterior, en la relación uno a uno se usarán líneas dirigidas a todas la entidades que la conforman. En la relación muchos a uno se usará la línea dirigida hacia el conjunto de entidades cuyos miembros pueden ser

asociados con varios elementos de los demás conjuntos de entidades que constituyen la relación. Por último, en la relación muchos a muchos se usarán líneas no dirigidas en todos los conjuntos de entidades de la relación. La figura 2.7 muestra el diagrama E-R del sistema de proyectos.

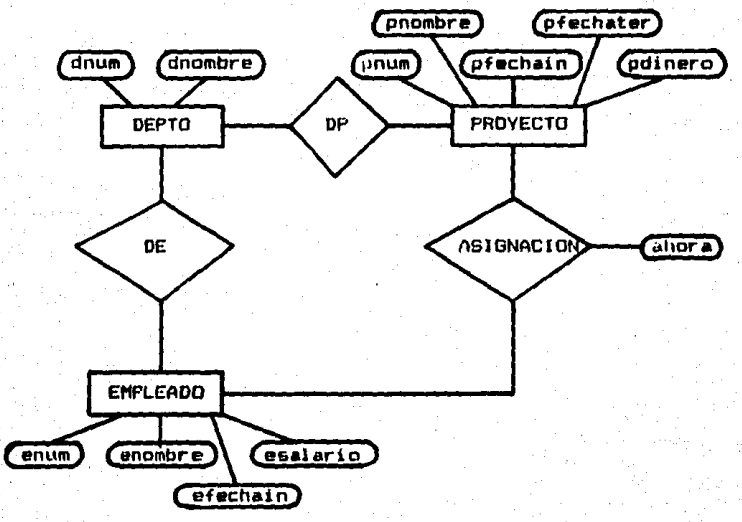


Figura 2.7 Diagrama E-R del sistema de proyectos

2.4 Modelo relacional

El modelo relacional representa la base de datos como una serie de tablas, siendo la correspondencia directa entre èstas y el concepto matemàtico de relaciòn uno de los motivos principales en el auge de este modelo.

¿Què es una relaciòn? Una relaciòn es un subconjunto del producto Cartesiano de una lista de dominios. Tal definiciòn nos lleva a recordar los conceptos de dominio y de producto Cartesiano. Un **dominio** es un cojunto de valores. El **producto Cartesiano** de la lista de dominios: D_1, D_2, ..., D_n, escrito $D_1 \times D_2 \times \ldots \times D_n$, es el conjunto de todas las n-tuplas $(v_1, v_2, \ldots, v_n)$ tales que v_1 està en D_1, v_2 està en D_2, y asi sucesivamente. Por ejemplo, si tenemos n = 2, D_1 = (0, 1), y D_2 = (a, b, c), entonces $D_1 \times D_2$ es ((0,a), (0,b), (0,c), (1,a), (1,b), (1,c)).

Ahora que hemos definido los conceptos en los cuales se fundamenta el de relaciòn, podemos apreciar que el conjunto ((0,a), (0,c), (1,b)) es una relaciòn, ya que es un subconjunto de $D_1 \times D_2$. El conjunto vacio tambièn es otro caso de una relaciòn.

Los elementos de una relaciòn se denominan **tuplas**, mientras que el nùmero de dominios que fundamentan la relaciòn se denomina el **grado** de la relaciòn. En el caso descrito

líneas arriba, una tupla puede ser (1, b) y el grado de la relación es dos.

El hecho de que podamos ver una relación como una tabla es esencial para la representación de los datos. ¿Cómo se da este "mapeo"? Asociando cada renglón de la tabla con una tupla de la relación, y cada columna de la tabla con un dominio de la relación. De esta manera, al hablar del modelo relacional estamos hablando de tablas.

Haciendo referencia a las tablas, diremos que las columnas que la conforman se denominan comúnmente atributos. El conjunto de atributos para una relación se conoce como el esquema de la relación. Si tenemos una relación conocida como REL y su esquema de relación tiene los atributos A_1, A_2, ..., A_n, escribiremos el esquema de la relación como REL(A_1, A_2, ..., A_n). Como ejemplo, obsérvese la figura 2.8. Ahí tenemos una relación cuyos atributos son: ENUM, ENOMBRE, EFECHAIN, ESALARIO. El grado de la relación es cuatro, y una tupla puede ser: (E03, Montoya, 09/21/84, 1500). Si la relación se conoce como EMPLEADO, el esquema de la relación es EMPLEADO(ENUM, ENOMBRE, EFECHAIN, ESALARIO).

EMPLEADO			
ENUM	ENOMBRE	EFECHAIN	ESALARIO
E01	Flores	05/06/84	2000
E02	Ramirez	02/04/82	1750
E03	Montoya	09/21/84	1500
E04	Iepes	07/12/86	1300
E05	Cortez	11/11/85	1300

Figura 2.8 Relación EMPLEADO

Representación de diagramas E-R en el modelo relacional

Para el diseño de una base de datos, una buena práctica es iniciar la representación de los datos haciendo uso del modelo Entidad-Relación, ya que, como vimos anteriormente, este modelo permite tener una perspectiva más amplia y clara en cuanto al manejo de datos de una organización. Sin embargo, cuando transitamos hacia el desarrollo del sistema, tenemos que mapear el diagrama E-R al modelo de datos que usa el manejador con el que se va a trabajar. Como efectuar este mapeo es lo que nos ocupa en estas líneas.

Los datos de un diagrama E-R son representados por dos tipos de relaciones (tablas):

a. Un conjunto de entidades puede ser representado por una tabla cuyo esquema lo conforman todos los atributos del conjunto de entidades. Cada renglón de la tabla -tupla de la relación- representa una

entidad del conjunto de entidades.

b. Una relación entre los conjuntos de entidades E_1, E_2, ..., E_n es representada por una tabla cuyo esquema lo determinan aquellos atributos que conforman las llaves para cada una de las E_1, E_2, ..., E_n.

$$\bigcup_{i=1}^{n} \text{llave-primaria}(E_i)$$

Un renglón t en esta tabla denota una lista de entidades e_1, e_2, ..., e_n, donde e_i es un elemento del conjunto E_i, para cada i. En el caso de que la relación tenga atributos descriptivos, digamos $(a_1, a_2, \ldots, a_m)$, entonces la tabla correspondiente será definida por los siguientes atributos

$$\bigcup_{i=1}^{n} \text{llave-primaria}(E_i) \cup (a_1, a_2, \ldots, a_m)$$

Por último, cuando la relación que se da entre los n conjuntos de entidades no tiene atributos descriptivos, y además es de muchos a uno, la llave de ésta podrá ser añadida dentro del esquema de las otras.

El ejemplo con el que se ilustra esto consiste en representar el diagrama de la figura 2.7 en el modelo relacional de la figura 2.9, que consta de cuatro tablas. En las

primeras tres tenemos la observancia de los dos primeros casos: la tabla EMPLEADO corresponde al conjunto de entidades EMPLEADO (primer caso) y a la relación muchos a uno que se da entre EMPLEADO y DEPTO, observando que dicha relación no tiene atributos descriptivos (segundo caso), mientras que en la última tabla tenemos el segundo caso -en la relación entre EMPLEADO y PROYECTO hay un atributo descriptivo: AHORAS.

EMPLEADO(ENUM, ENOMBRE, EFECHAIN, ESALARIO, DNUM)
DEPTO(DNUM, DNOMBRE)
PROYECTO(PNUM, PNOMBRE, PFECHAIN, PFECHATER, PDINERO, DNUM)
ASIGNACION(ENUM, PNUM, AHORAS)

Figura 2.9 Representación del modelo relacional

Operaciones en bases de datos relacionales

Para el usufructo de las bases de datos será necesario disponer de un repertorio de operaciones que permitan accesarla. La satisfacción de tal necesidad es cubierta por medio de los **lenguajes de preguntas** -Query Languages-. Estos son lenguajes de alto nivel que se clasifican en procedurales y no procedurales. En un lenguaje **procedural**, el usuario instruye al sistema a ejecutar una secuencia de operaciones sobre la base de datos para computar el resultado deseado. En un lenguaje **no procedural**, el usuario describe la información deseada sin dar un procedimiento

específico para obtener esa información.

El diseño de tales lenguajes se fundamenta en tres notaciones abstractas: el álgebra relacional, el cálculo relacional de tuplas y el cálculo relacional de dominios, siendo esta la causa de la clasificación de los lenguajes. Los lenguajes procedurales se basan en el álgebra relacional, mientras que los no procedurales en el cálculo relacional.

Uno de los lenguajes comerciales de mayor uso actualmente es SQL -Strucutured Query Language-. En vista de que gran parte del diseño de este lenguaje se basa en el álgebra relacional y de que la temática principal del trabajo aquí expuesto no son las notaciones abstractas, daremos una lacónica reseña de éste.

Cinco son las operaciones fundamentales del álgebra relacional: **selección** (σ), **proyección** (π), **producto cartesiano** ($\times$), **unión** ($\cup$), y **diferencia** ($-$) de conjuntos. Todas estas operaciones producen una nueva tabla como resultado. Además de estas cinco operaciones, se conocen otras no menos importantes: **intersección** ($\cap$) de conjuntos, **join natural** ($\bowtie$), y **división** ($\div$). Estas últimas podríamos denominarlas "compuestas", ya que se apoyan en las cinco operaciones básicas.

Para efecto de ilustración, mostraremos sólo algunas operaciones. Supóngase que deseamos obtener de la base de datos de la figura 2.10 el nombre de todos los proyectos cuya fecha límite de terminación sea 31/12/89. Para obtener la respuesta, requerimos de dos operaciones: la selección de aquellos proyectos que deban concluir a más tardar el 31/12/89 y la proyección de todas estas tuplas -que han sido seleccionadas previamente- sobre el campo PNOMBRE. El procesamiento de la pregunta se muestra en la figura 2.11.

ENUM	ENOMBRE	EFECHAIN	ESALARIO
E01	Flores	05/06/84	2000
E02	Ramirez	02/04/82	1750
E03	Montoya	09/21/84	1500
E04	Iepes	07/12/86	1300
E05	Cortez	11/11/85	1300

Relación EMPLEADO

DNUM	DNOMBRE
C10	Sistemas
C11	Graficacion
C12	Redes

Relación DEPTO

ENUM	PNUM	AHORAS
E01	102	800
E01	103	800
E02	101	1600
E03	104	1600
E04	105	1600
E05	105	800

Relación ASIGNACION

PNUM	PNOMBRE	PFECHAIN	PFECHATER	PDINERO	DNUM
101	C 5.1	01/01/89	31/10/89	500	C10
102	DOS 4.0	01/01/89	30/06/90	1000	C10
103	OS/2	01/10/88	31/10/89	1000	C10
104	MathCAD	01/11/88	31/12/89	1000	C11
105	LAN 2.0	01/01/86	30/06/90	1500	C12

Relación PROYECTO

Figura 2.10 Parte de la base de datos proyectos

$$\pi_{PNOMBRE}(\sigma_{PFECHATER \leq 31/12/89}(PROYECTO))$$

PNUM	PNOMBRE	PFECHAIN	PFECHATER	PDINERO	DNUM
101	C 5.1	01/01/89	31/10/89	500	C10
103	OS/2	01/10/88	31/10/89	1000	C10
104	MathCAD	01/11/88	31/12/89	1000	C11

$$NR \longleftarrow \sigma_{PFECHATER \leq 31/12/89}(PROYECTO)$$

PNOMBRE
C 5.1
OS/2
MathCAD

$$\pi_{PNOMBRE}(NR)$$

Figura 2.11 Procesamiento del primer ejemplo

Para el resultado de la pregunta anterior sólo tuvimos que hacer uso de una tabla. Ahora veremos una operación muy común que involucra dos tablas. Suponga que deseamos obtener el nombre de todos los proyectos del departamento de sistemas. Para computar la respuesta tenemos que conectar los datos de dos tablas -DEPTO y PROYECTO- a través de todos aquellos atributos que les son comunes. En este caso, ese atributo es DNUM. La operación que realiza este enlace se denomina Join natural. La ejecución de esta operación se realiza en tres pasos (la figura 2.12 los ilustra):

a. Computar el producto Cartesiano de las dos tablas
b. Para cada atributo común de ambas tablas, seleccionar aquellas tuplas cuyos valores coincidan en estos atributos.
c. Para cada atributo común A, descartar los de la segunda tabla (en la notación algebraica esto se hace implícito en la unión de los dos esquemas).

Para expresar esta operación en notación algebraica tenemos que si A_1, A_2, ..., A_n son todos los atributos de R y S, entonces el Join natural es:

$$R * S = \pi_{DNUM}(\sigma_{R.A1 = S.A1 \wedge R.An = S.An}(R \times S))$$

Por último, para concluir con el cómputo de la respuesta a la pregunta anterior, una vez que se ha realizado el Join

natural continuamos con la selección de las tuplas -cuyo valor en el atributo DNOMBRE sea sistemas-, finalizando con la proyección sobre el atributo PNOMBRE. La expresión de la pregunta en notación algebraica es la siguiente.

$$\Pi_{PNOMBRE}(\sigma_{DNOMBRE = sistemas}(PROYECTO \bowtie DEPTO))$$

PNUM	PNOMBRE	PFECHAIN	PFECHATER	PDINERO	DNUM	DNUM	DNOMBRE
101	C 5.1	01/01/89	31/10/89	500	C10	C10	Sistemas
102	DOS 4.0	01/01/89	30/06/90	1000	C10	C10	Sistemas
103	OS/2	01/10/88	31/10/89	1000	C10	C10	Sistemas
104	MathCAD	01/11/88	31/12/89	1000	C11	C10	Sistemas
105	LAN 2.0	01/01/86	30/06/90	1500	C12	C10	Sistemas
101	C 5.1	01/01/89	31/10/89	500	C10	C11	Grafic.
102	DOS 4.0	01/01/89	30/06/90	1000	C10	C11	Grafic.
103	OS/2	01/10/88	31/10/89	1000	C10	C11	Grafic.
104	MathCAD	01/11/88	31/12/89	1000	C11	C11	Grafic.
105	LAN 2.0	01/01/86	30/06/90	1500	C12	C11	Grafic.
101	C 5.1	01/01/89	31/10/89	500	C10	C12	Redes
102	DOS 4.0	01/01/89	30/06/90	1000	C10	C12	Redes
103	OS/2	01/10/88	31/10/89	1000	C10	C12	Redes
104	MathCAD	01/11/88	31/12/89	1000	C11	C12	Redes
105	LAN 2.0	01/01/86	30/06/90	1500	C12	C12	Redes

a. Primer paso

PNUM	PNOMBRE	PFECHAIN	PFECHATER	PDINERO	DNUM	DNUM	DNOMBRE
101	C 5.1	01/01/89	31/10/89	500	C10	C10	Sistemas
102	DOS 4.0	01/01/89	30/06/90	1000	C10	C10	Sistemas
103	OS/2	01/10/88	31/10/89	1000	C10	C10	Sistemas
104	MathCAD	01/11/88	31/12/89	1000	C11	C11	Grafic.
105	LAN 2.0	01/01/86	30/06/90	1500	C12	C12	Redes

b. Segundo paso

PNUM	PNOMBRE	PFECHAIN	PFECHATER	PDINERO	DNUM	DNOMBRE
101	C 5.1	01/01/89	31/10/89	500	C10	Sistemas
102	DOS 4.0	01/01/89	30/06/90	1000	C10	Sistemas
103	OS/2	01/10/88	31/10/89	1000	C10	Sistemas
104	MathCAD	01/11/88	31/12/89	1000	C11	Grafic.
105	LAN 2.0	01/01/86	30/06/90	1500	C12	Redes

c. Tercer paso

Figura 2.12 Pasos del Join natural

2.5 Modelo de red

Escuetamente, el modelo de red es el modelo de datos E-R con una gran restricción en todas sus relaciones: deben ser binarias, y su mapeo o funcionalidad es muchos a uno. Por lo que toca al modelo relacional, el modelo de red difiere en que los datos son representados por registros, y las relaciones entre los datos se representan por medio de ligas.

Una base de datos red consiste en una colección de registros que son conectados uno con otro a través de ligas. Un registro se puede ver como un conjunto de entidades en el modelo E-R. Cada registro es configurado por una serie de campos (atributos). Una liga es una asociación entre dos registros.

Para la representación de una base de datos de red se dispone de una gráfica dirigida denominada red. Los nodos corresponden a los tipos de registros. Si hay una liga entre dos tipos de registros T_1 y T_2, y la liga es muchos a uno de T_1 a T_2, entonces trazamos una flecha que parte del nodo T_1 hasta T_2. Los nodos y las flechas son etiquetados con los nombres de sus tipos de registro y ligas, respectivamente.

Representación de diagramas E-R en el modelo de red

En lo que corresponde a las entidades, éstas tienen una representación directa por medio de registros. En cuanto a

las relaciones, sólo aquéllas que son binarias y cuya funcionalidad sea muchos a uno podrán ser declaradas por ligas. ¿Qué hacer cuando las relaciones a considerar no son binarias o no son muchos a uno?

Para la representación de relaciones arbitrarias, sea R una relación entre los conjuntos de entidades E_1, E_2, ..., E_k. Creamos un nuevo tipo de registro T -conocido como **conector**- que representará a k-tuplas (e_1, e_2, ..., e_k) de las entidades que conforman la relación R. El esquema para este tipo de registro puede consistir en un solo campo que contiene un identificador único, cuyo objetivo es identificar a instancias de este registro -si hya atributos descriptivos en la relación, éstos serán añadidos o formarán parte del esquema de este nuevo tipo de registro T-. Después, creamos k ligas: L_1, L_2, ..., L_k. La liga L_i es desde el tipo de registro T (conector) al tipo de registro T_i para el conjunto de entidades E_i. De esta manera, el registro T que representa a k-tuplas (e_1, e_2, ..., e_k) es ligado al registro T_i, el cual representa a la tupla e_i.

El ejemplo que nos permitirá aplicar lo expuesto previamente consiste en obtener la red equivalente al diagrama E-R de la figura 2.7. Dicha red se muestra en la figura 2.13. Para comprender esta figura, en el diagrama E-R de la figura 2.7 se aprecian tres relaciones: DE, DP, y

ASIGNACION. Las dos primeras son binarias y su funcionalidad es muchos a uno, por lo que su representación en la red es directa. Para la última relación, ASIGNACION, observamos que, aunque es binaria, su funcionalidad es muchos a muchos lo cual da origen a un nuevo registro: ASIGNACION, y a dos ligas: EA y PA. La descripción de los registros que conforman la red son:

EMPLEADO(ENUM, ENOMBRE, EFECHAIN, ESALARIO)

DEPTO(DNUM, DNOMBRE)

PROYECTO(PNUM, PNOMBRE, PFECHAIN, PFECHATER, PDINERO)

ASIGNACION(AID, AHORAS)

mientras que las ligas son:

DE de EMPLEADO a DEPTO

DP de PROYECTO a DEPTO

EA de ASIGNACION a EMPLEADO

PA de ASIGNACION a PROYECTO

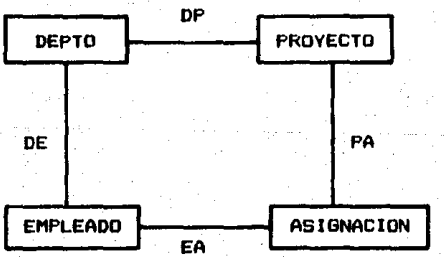


Figura 2.13 Red para la base de datos proyectos

Operaciones sobre redes

Dos son los tipos de operaciones que podemos ejecutar en las bases de datos de red. La primera de ellas es la selección de sobre los registros de la red. La operación es similar a la selección del modelo relacional. Como ejemplo para esta operación, supóngase que de acuerdo a la base de datos de la figura 2.14 deseamos imprimir todos los investigadores que ingresaron a la institución antes de 1985. Para obtener la respuesta, accesamos al registro EMPLEADO en la red y seleccionamos aquellas ocurrencias cuyo valor en el campo EFECHAIN < 01/01/85.

La otra operación se conoce como **navegación:** es el seguimiento de las ligas -que representan las relaciones entre los datos- en una dirección o en otra. Para ilustrar esta operación, supóngase que se pretende encontrar el número de horas dedicadas por parte del investigador E03 al proyecto 104. Para responder a tal pregunta, debemos accesar la única ocurrencia del conector (registro ASIGNACION) que liga tanto a la cadena para E03 como a la cadena para 104. ¿Cómo llegar a tal ocurrencia? Tenemos dos caminos. Uno empieza en el investigador, rastreando su cadena en búsqueda de un conector -ocurrencia- ligado al proyecto. El otro camino empieza en el proyecto, rastreando su cadena en búsqueda de un conector ligado al investigador.

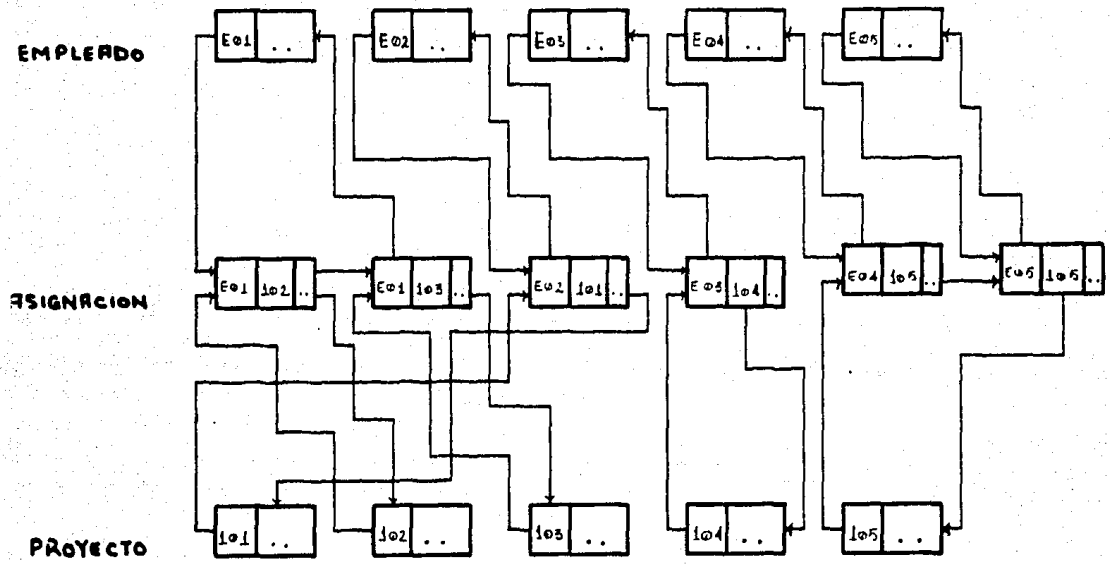


Figura 2.14 Base de datos Proyectos

2.6 Modelo jerÀrquico

El modelo de datos jerÀrquico tiene mucha similitud con el modelo de red. Su semejanza radica en la represenatciòn de los datos, lo cual significa que tanto los datos como las relaciones entre los datos se expresan por medio de registros y ligas, respectivamente. Sin embargo, el modelo jerÀrquico difiere del modelo de red en cuanto a la forma de organizar los registros: en èste, conforman una grÀfica arbitraria, mientras que en aquèl, un bosque -por medio de un conjunto de Àrboles, por supuesto.

Representaciòn de diagramas E-R en el modelo jerÀrquico

El diagrama que representa una base de datos jerÀrquica es el Àrbol. Como es del dominio comùn, esta grÀfica esta compuesta por nodos y ramas. Los nodos simbolizan resgistros y las ramas, ligas. Ahora, en cuanto a la representaciòn de un diagrama E-R en el modelo jerÀrquico, tenemos que un conjunto de entidades serà expresado a travès de los nodos, mientras que las relaciones entre las entidades se manifiestan vìa las ramas. Las relaciones uno a uno y muchos a uno tienen representaciòn directa, no asì la relaciòn muchos a muchos. En este ùltimo caso, habrà que hacer uso de **"registros virtuales"**, los cuales dan la soluciòn a dos graves problemas: inconsistencia en los datos y desperdicio de memoria. En la figura 2.15 podemos apreciar una posible representaciòn -sin ser la ùptima- del diagrama E-R de la

figura 2.7.

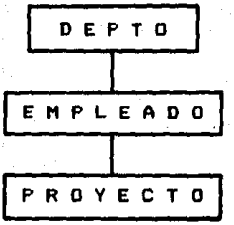


Figura 2.15 Un árbol para la base de datos proyectos

Operaciones sobre jerarquías

La operación básica en una base de datos jerárquica es recorrer el árbol: dado un nodo, podremos rastrear todos sus hijos. Lo anterior implica que la operación básica es **unidireccional**, procediendo de padres a hijos solamente -lo cual difiere de las ligas de una red, ya que la operación básica en ésta (navegación) es bidireccional-. Esta limitante en el recorrido es superada mediante el uso del registro virtual, ya que no sólo permitirá "subir" en el árbol -la otra dirección-, sino también pasar de un árbol a otro en el bosque que simboliza la base de datos.

De acuerdo con la instancia de la base de datos de la figura 2.16, supóngase que se desea conocer todos los proyectos en los cuales participan los investigadores del departamento de sistemas. La respuesta se obtiene empezando

en la raíz, buscando en DEPTO la ocurrencia de sistemas. Posteriormente rastreamos los hijos de sistemas, lo cual significa accesar todos sus investigadores. Por último, para cada investigador obtenemos sus hijos, lo cual equivale a conocer los proyectos en que están involucrados.

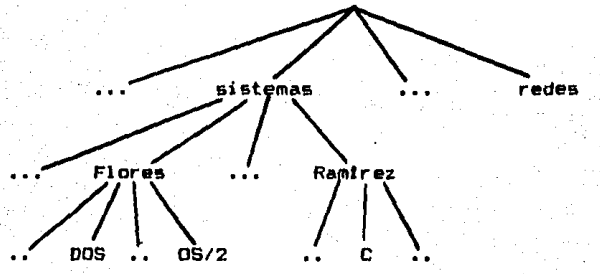


Figura 2.16 Parte de la base de datos proyectos

2.7 Vistas

Las vistas son el punto teórico (en el estudio de las bases de datos) que concierne al trabajo aquí presentado. Las abordaremos indicando primeramente què son, qué problemas originan al actualizar una base de datos y a qué metodo (de los ya existentes) nos apegamos para desarrollar el trabajo.

A la gran mayoría de los que interactúan con una base de datos sólo les interesan ciertas partes de ella: abstracciones definidas por el usuario, conocidas como **vistas.** Tal abstracción se realiza a través de del esquema conceptual de la base de datos. ¿Cuáles son las causas de su origen? Podemos hablar de tres causas:

a. contribuyen en gran parte a la independencia lógica de los datos -como vimos al inicio del capítulo, esta independencia es un objetivo al que se aspira en el diseño de la base de datos-, ya que muchas de las modificaciones que se realizen en el esquema de la base de datos no necesariamente repercutirán en la vista.

b. son un medio de protección (seguridad), pues la perspectiva del usuario no va más allá de lo estipulado en la definición de la vista, lo cual impide que el usuario accese datos que no debe.

c. son la facultad que permite hacer un usufructo sencillo del sistema, ya que la atención del

usuario está ceñida a aquellos datos que sólo a él incumben.

Los detalles de la definición de la vista varían en cuanto al sistema de que se disponga. La figura 2.17 muestra una posible vista de la relación (tabla) EMPLEADO de la figura 2.10. Se puede apreciar que en la tabla aparece la columna ANTIGUEDAD, que no pertenece al esquema del empleado. ¿Cómo se obtuvieron los valores de tal columna? Por medio de una expresión que computa la diferencia entre la fecha actual y la fecha de ingreso del investigador a la institución. Esto último ilustra el caso de las expresiones en la definición de una vista.

ENUM	ENOMBRE	ANTIGUEDAD
E01	Flores	5
E02	Ramirez	7
E03	Montoya	5
E04	Iepes	3
E05	Cortez	4

Figura 2.17 Una vista de la relación EMPLEADO

Problemas al actualizar una base de datos a través de vistas Hemos dicho que la estructura de una vista se obtiene a través de una serie de operaciones aplicadas al esquema conceptual; a su vez, el contenido (instancia) de la vista

es definido por la misma secuencia de operaciones aplicadas a la instancia conceptual (o de la base de datos). Lo anterior implica que la instancia de la vista no es independiente, o en otras palabras, las alteraciones a la instancia de la base de datos dan lugar (sin ninguna ambigüedad o indeterminación) a modificaciones en la instancia de la vista.

Para que una vista sea útil, el usuario deberá disponer de dos operaciones: el acceso y la actualización de la información. En la operación de acceso -conocida también como "read-only"- no hay modificaciones a la vista. Cuando se trata de accesar, lo que se va a ejecutar es la aplicación de la definición de la vista a la instancia de la base de datos. Esto significa que el acceso de información desde una vista se logra mediante accesos adecuados a la instancia conceptual. En el caso de actualizar, la modificación desde una vista también se logra mediante alteraciones correspondientes sobre la instancia de la base de datos. Sin embargo, en este caso (actualización), tales alteraciones no siempre existen, y aún cuando existan, podrán no ser únicas. De esta manera, una modificación en la instancia de la vista puede dar lugar a ambigüedad al aplicar las modificaciones equivalentes en la base de datos.

Para ilustrar tal ambigüedad haremos una pequeña modificación al esquema de la base de datos que hemos venido manejando. Supóngase que es la mostrada en la figura 2.18, Asimismo, considérese la vista de la figura 2.19, y que se desea insertar en la vista la tupla: (Murguía, IA, Rich). Si la inserción de tal tupla en la vista sobre la base de datos da lugar a la siguiente operación:

a. insertar la tupla (Murguía, IA) en EMPLEADO

b. insertar la tupla (IA, Rich) en DEPTO

entonces podemos ver la siguiente ambigüedad

$$VISTA_{modificada} \xrightarrow{cambios} \text{BASE DE DATOS} \dashrightarrow VISTA_{modificada}$$

lo cual significa que al hacer la modificación a la base de datos, ésta, una vez modificada, si aplicamos la definición de la vista, da lugar a una instancia de la vista que no es la esperada -aparecería también la tupla (Coronado, IA, Rich).

ENOMBRE	DNOMBRE
Flores	Sistemas
Ramirez	Sistemas
Montoya	Graficacion
Iepes	Redes
Cortez	Redes
Coronado	IA

Relación EMPLEADO

DNOMBRE	JEFES
Sistemas	Ullman
Sistemas	Silberschatz
Graficación	Newell
Redes	Tenenbaum
Análisis N.	Burden

Relación DEPTO

Figura 2.18 Base de datos para ilustración de vistas

ENOMBRE	DNOMBRE	JEFES
Flores	Sistemas	Ullman
Flores	Sistemas	Silberschatz
Ramirez	Sistemas	Ullman
Ramirez	Sistemas	Silberschatz
Montoya	Graficación	Newell
Iepes	Redes	Tenenbaum
Cortez	Redes	Tenenbaum

```
create view[1] ED as
        select ENOMBRE, DNOMBRE, JEFES
        from EMPLEADO, DEPTO
        where EMPLEADO.DNOMBRE = DEPTO.DNOMBRE
```

Figura 2.19 Definición e instancia de la vista ED

Ahora veamos un caso en el cual no existe una secuencia de operaciones que pueda ejecutar la alteración en una vista. Considérese que deseamos eliminar de la vista la tupla (Flores, Sistemas, Ullman). Si para la eliminación de tal tupla de la vista se efectúa la siguiente operación:

a. eliminar la tupla (Flores, Sistemas) de EMPLEADO

entonces hay incongruencia con la tupla (Flores, Sistemas, Silberschatz). Si la operación a efectuar es la siguiente:

a. eliminar la tupla (Sistemas, Ullman) de DEPTO

también hay incongruencia con la tupla (Ramírez, Sistemas, Ullman). Ello nos permite observar que, en este caso, no

[1]La definición de la vista se hace en base a SQL.

existen operaciones sobre la base de datos que ejecuten correctamente este movimiento (eliminación) en la vista.

Genéricamente hablando, el problema de actualizar una base de datos a través de vistas es el siguiente: para traducir una actualización u sobre la vista, debemos encontrar una actualización t sobre la base de datos, tal que t transforme el estado actual s de la base de datos en un nuevo estado s' si y sólo si, u modifica el estado actual de la vista v (generado de s) en un nuevo estado de la vista v', que es generado a partir de s'. Este grave problema[8] ha ocasionado que la mayoría de los sistemas actuales permitan

[8]Hasta donde sé, sobre este problema se investiga mucho actualmente. Por mi parte, consulté dos trabajos:

a. "On the Correct Translation of Update Operations On Relational Views", [Dayal y Bernstein 1982]. Aquí se exponen una serie de procedimientos de traducción, los cuales recibirán como entrada la definición de la vista y la alteración (edición) deseada para la actualización de la vista (view update), generando, en caso de ser factible, la traducción de tal edición en un conjunto adecuado de movimientos sobre la instancia de la base de datos que satisfagan algúnas propiedades.

b."Updates of Relational Views", [Cosmadakis y Papadimitriou 1984]. Es similar al anterior en cuanto a que lo que se produce es una traducción (en caso de ser posible). La diferencia estriba en que en este caso el sistema requiere una segunda vista denominada "complemento" de la vista, en la cual se concentra toda la información omitida por la primera.

En ambos casos, la conclusión es que las traducciones a las actualizaciones por medio de vistas sólo son posibles bajo condiciones estrictas, siendo esta rigurosidad lo que ha implicado que la mayoría de los sistemas actuales (para el manejo de la información) permitan la edición de la base de datos a través de vistas restringidas.

actualizar la base de datos a través de vistas, siempre y cuando la definición de éstas se fundamente en una sóla relación (tabla). Sin embargo, aún cuando la vista esté definida con base en una tabla, la integridad deberá vigilarse. El siguiente punto se ocupa de ello.

La integridad al actualizar con vistas de una sola relación

Hemos observado en el punto previo que actualizar una base de datos a través de una vista cuya definición es arbitraria --sin restricciones- provoca graves problemas. Ello nos llevó a permitir la actualización de la base de datos, siempre y cuando la definición de la vista se fundamente en una relación. No obstante lo anterior, el problema de integridad de los datos está latente.

El problema consiste en que la actualización a la vista puede repercutir en puntos o sectores de la base de datos que no forman parte de la vista. Por ejemplo, considérese la relación muchos a uno en el sistema jerárquico. Si la vista tiene el registro padre -a éste y no al registro hijo, ya que la vista es restringida- y se pretende eliminarlo, sus hijos deberán ser eliminados también, o no deberá permitirse tal movimiento. En nuestro caso -sistema relacional- y de acuerdo con la base de datos de que hemos venido haciendo uso, podemos apreciar este problema suponiendo que tenemos una vista de la relación DEPTO y se desea dar de baja una

tupla a cuya relación EMPLEADO -que por supuesto no está en la vista- hace referencia (esto es, hay investigadores en en ese departamento).

Como se pudo apreciar, al actualizar la base de datos, estamos expuestos a una posible pérdida de consistencia en la información. ¿Qué hacemos para resolver esta probable anomalía? Disponer de las **dependencias de datos** (una forma de restricción de integridad), entendiendo por éstas a todas aquellas declaraciones concernientes a la descripción de la empresa u organismo que se está modelando. Resulta entonces que son estas dependencias las directrices en el diseño de la base de datos, ya que pretendemos que contenga únicamente aquellas relaciones que satisfagan las dependencias. El análisis que se lleva a cabo para obtener tales relaciones se conoce como **normalización**: descomposición de los esquemas en otros que cumplan todas las dependencias señaladas.

Existen diversas formas de normalización (funcional, multivaluada, etc.), siendo la conocida como **forma normal de Boyce-Codd (BCNF)** la que hemos usado en este trabajo porque es ideal desde el punto de vista de la eficiencia, ya que las únicas formas de restricción que necesitamos probar son restricciones de llave y restricciones de dominio[3]. Las

[3]En [Korth y Silberchatz 1986] hablan de una forma normal de dominio-llave (DKNF) la cual permite probar el cumplimiento de las limitantes **generales** empleando solamente las de dominio y llave. De acuerdo a lo anterior, esta forma normal

restricciones de llave son verificadas por medio de los índices primarios, mientras que las restricciones de dominio requieren de la observancia en cuanto a la admisión de valores nulos. Estos no serán permitidos para ciertos atributos -por ejemplo, los que conformen la llave- pero sí para otros.

es más ambiciosa y por lo tanto más difícil de lograr (en el libro le llaman forma normal "extrema" o "idealizada"). Sin embargo, la exposición de esta forma normal es muy superficial.

3. DISEÑO DEL EDITOR

Me gustaría iniciar este capítulo con un fragmento de una entrevista reciente con Jean-Claude Sperandio, profesor de ergonomía[1] y director de la Unidad de Informática de Ciencias Humanas en la Universidad de París V, realizada por el periódico PC Journal[2]:

PCJ.- Parecería que se habla menos de la ergonomía, pero que se le toma más en cuenta. Por ejemplo, vemos aparecer en las grandes empresas a grupos de "ergónomos" que participan en proyectos de computación. ¿Qué opina usted como investigador y como consultor?

JCS.- ... efectivamente, la ergonomía se esta esparciendo por el medio de la computación; al principio se hacía sobre

---

[1] pretendo justificar el uso de este vocablo (ergonomía) en el trabajo aquí expuesto, mediante las dos observaciones siguientes:

- el libro "Para saber lo que se dice" de Arrigo Coen en la página 77 dice: <<Ergonomía es una palabra que, al primer intento de hallar su procedencia, resulta engañosa, porque una síncopa la ha hecho perder uno de sus lexemas; originalmente fue ergoeconomía, por lo que el concepto cabal es 'economía o ahorro de trabajo'. La supresión del elemento -eco- (en griego oikos, 'casa') le da una fisonomía falaz, ya que nomía, solo, quiere decir 'ley', con lo que el etimólogo poco avisado propende a interpretar ergonomía, equivocadamente, como 'ley del trabajo' o 'norma de la energía' ...

- de acuerdo a la observación anterior, debería usar la palabra ergoeconomía; sin embargo, es innegable que el vocablo ergonomía tiene mayor aceptación (lo he visto en periódicos, revistas y en la enciclopedia del idioma de Martín Alonso). En vista de que el uso crea la regla, y por otra parte, mi objetivo es que se entienda lo aquí expuesto, decidí usar esta palabra (ergonomía).

[2] PC Journal, MEXICO, Núm. 22, julio 25, 1988.

todo **ergonomía física**, relacionada con el material y más específicamente con las pantallas ... actualmente se interesan por la **ergonomía de los programas**, pero de un modo un tanto superficial, concentrándose en aplicarla a los diálogos y a los códigos, un punto que, a mi modo de ver, es poco importante.

PCJ.- ¿Se refiere usted a la interfaz hombre-máquina?

JCS.- Sí. Pero existen dos niveles de interfaz; un primer nivel que corresponde a la **interfaz física**, el ratón, el teclado, el color, etc., y el segundo, que se relaciona con los **programas**. Como les decía, hoy se está trabajando apenas con la parte exterior del iceberg, los códigos, los diálogos, los retornos, las interrupciones, etc.

PCJ.- ¿Cuál es la parte sumergida del iceberg que se está ignorando?

JCS.- Son las capas más profundas las que tocan la búsqueda de funciones adaptadas a la tarea, las que obligan a realizar un **análisis del trabajo** y las necesidades de los usuarios ...

Como podrá apreciarse, hoy en día adquiere una mayor relevancia la ergonomía aplicada en el desarrollo del software. Mi trabajo trata de cumplir con lo dicho por Jean-Claude Sperandio, esto es, los dos componentes (editor y manejador de pantallas) del sistema se procuró realizarlos mediante la observancia de dos factores: el análisis de la

función a realizar y, el análisis de las necesidades de los usuarios.

3.1 Definición de la vista

La idea o motivación para el desarrollo del sistema la tomé de la exposición que se hace del sistema Karlsruhe[9]. Es precisamente el editor la sección del sistema que fundamenta parte de su estructura, particularmente el manejo de los objetos.

¿Cómo definir una vista sin la rigidez o hermetismo impuesto por la sintaxis de un lenguaje? ¿Cómo proporcionar en su definición no sólo elementos de tipo geométrico, sino también lógicos? Preguntas como las anteriores fueron esenciales en la etapa de diseño del sistema. Esta etapa de diseño se basa en dos factores: geométricos y lógicos.

3.1.1 Factores geométricos

Aquí nos interesa la forma en que se presentan los datos. El elemento geométrico principal es la pantalla. Esta puede ser dividida en dos partes o cajas, lo cual, como se señala en el capítulo uno, permite manejar la relación más frecuente entre los datos: "uno a muchos". Cada caja es una estructura "TEs" cuya descripción es la siguiente:

---

[9] J.J. Bass, "A Generalized User Interface for Applications Programs", Commun. ACM 28, 6 (Jun. 1985).

```
typedef struct box {
    UCHAR  hor_line ;      /* línea horizontal */
    UCHAR  vert_line ;     /* línea vertical   */
    UCHAR  ulcorner ;      /* esquina superior izquierda */
    UCHAR  urcorner ;      /* esquina superior derecha   */
    UCHAR  llcorner ;      /* esquina inferior izquierda */
    UCHAR  lrcorner ;      /* esquina inferior derecha   */
    UCHAR  fillc ;         /* caracter con que se llena  */
    ATTR   lineattr ;      /* atributo para las líneas   */
    ATTR   fillattr ;      /* atributo para el campo fillc*/
    UCHAR  row ;
    UCHAR  col ;
    UCHAR  high ;
    UCHAR  wide ;
    TOKEN  *crtimage ;  /* localidad de la imagen guardada */
    struct box  *lte ;     /* caja izquierda          */
    struct box  *rte ;     /* caja derecha            */
    char   screen ;        /* estrategia de la caja */
    UCHAR  no_op ;         /* operadores en la caja */
    UCHAR  x1, x2 ;
} TEs, *TEPTR ;
```

Los componentes de la estructura se definieron basándose en tres consideraciones: forma de la caja, manejo de la pantalla en la caja y el número de operadores en la caja. A continuación, describimos brevemente cada uno de ellos.

Forma de la caja

El control de la forma y dimensiones de la caja se hace con los primeros campos de la estructura (desde hor_line hasta *crtimage). Con estos campos manejamos ventanas.

Manejo de la pantalla en la caja

Para el manejo de la pantalla existen dos opciones: fijo y móvil ("scroll"). Un manejo de pantalla fijo indica que los objetos que sean definidos en la vista deberán aparecer

siempre en el mismo lugar. La figura 1.1 sería un ejemplo. Por otra parte, si la estrategia es móvil, los objetos irán apareciendo en la siguiente línea disponible, haciendo una especie de "enrrollamiento" por la parte superior de la caja. Tal estrategia es muy útil en el caso de los reportes. El usuario puede definir una caja superior con estrategia fija para el encabezado y otra caja inferior con estrategia móvil para las ocurrencias de los registros. La figura 3.1 muestra un típico reporte, que hace uso de las dos partes o cajas.

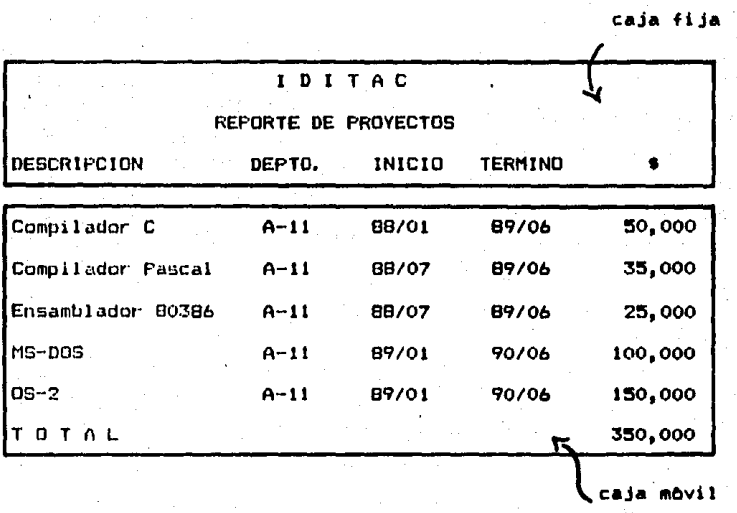


Figura 3.1 Cajas con diferente estrategia en el manejo de la pantalla

Número de Operadores en la Caja

Cada vez que el usuario añade un operador a la vista, este campo se incrementa. El manejador de los operadores -que forma parte del manejador de pantallas- es el que utiliza este campo.

3.1.2 Factores lógicos

Aquí nos interesa saber qué datos -fondo- se van a presentar. Para tal efecto, los elementos lógicos de la vista son:

* archivos de datos principal y secundarios
* cuatro tipos de objetos: texto, variables, expresiones y operadores
* un predicado
* cláusula de agrupamiento

Archivos de datos principal y secundarios

A través del editor se genera la vista de un archivo de datos, siendo éste el archivo principal. La definición de la vista podrá comprender otros archivos siempre y cuando éstos tengan relación con el principal. A tales archivos se les denominará secundarios. El módulo crel.c es el encargado de observar la presumible relación entre los archivos principal y secundarios. Para que el módulo crel.c pueda determinar la existencia de tal relación, debemos obtener los esquemas de cada uno de los archivos involucrados. La obtención de los

esquemas se realiza a través de la interfaz con el diccionario de datos.

Si en la definición de la vista se desea incorporar otros campos de archivos diferentes al pricipal -secundarios-, éstos serán unica y exclusivamente de **salida.** Las figuras 3.2 y 3.3, pretenden ilustrar la gran importancia que tienen estos campos en la definición de una vista -las vistas presentadas en estas figuras han sido definidas a partir del esquema de la base de datos Proyectos (ver capitulo dos). Es imperativo señalar que los campos o variables de salida no podrán ser mostrados si el valor de la llave del archivo al que pertenece está indefinido. Por ejemplo, si en la figura 3.2 el número del departamento (dnum) está indefinido, el nombre del departamento no podrá ser determinado.

CONTROL DE EMPLEADOS

Información Personal

No. Empleado : _____ Nombre: _____

Fecha Ingreso: _____

Salario ($) : _____

Información Laboral

No. Depto. : _____ Nombre del Depto.: __**(sólo salida)**__

Figura 3.2 Vista para EMPLEADO

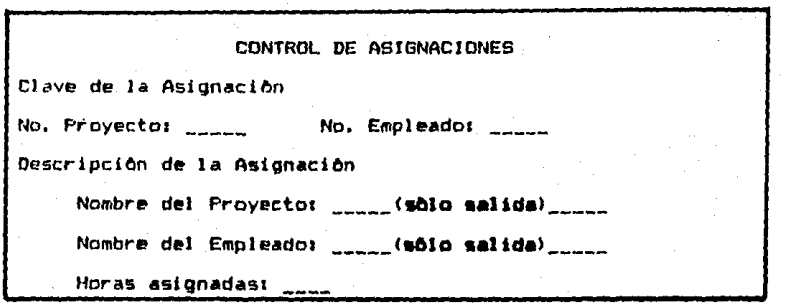


Figura 3.3 Vista para ASIGNACIONES

Tipos de Objetos

En el editor se tiene cuatro tipos de objetos:

* texto (T)
* variables (V)
* expresiones (E)
* operadores (O)

cada uno con su estructura propia (descrita más adelante en este mismo capítulo). Para su manejo, el editor hace uso de un arreglo de control (ctr[]) de 24 elementos ya que existe un miembro por cada línea de la pantalla. Cada miembro del arreglo lo conforman:

* el número de objetos en la línea
* indicador que determina si en la línea se definieron operadores o no
* un apuntador al primer nodo de una lista doblemente ligada (LDL)
* un apuntador al último nodo de la LDL

Cada nodo de la LDL es un objeto. El objeto está constituido por:

* la estructura que describe al tipo de objeto
* un indicador del tipo de objeto (T -> texto, V -> variable, E -> expresión, O -> operador)
* un apuntador al siguiente objeto
* un apuntador al objeto anterior

La definición de toda la estructura que maneja los objetos del editor se presenta líneas abajo. Dicha estructura es más evidente por medio de su representación gráfica, que se presenta en la figura 3.4.

```
typedef struct object {
   union {
        struct texto    text ;
        struct variable var ;
        struct funcion  expr ;
        struct operadpr op ;
   } nodo ;
   char   tip_obj ;          /* T, V, E, O */
   struct object *forw ;     /* siguiente objeto */
   struct object *back ;     /* objeto anterior  */
} OBJ, *OBJPTR ;


typedef struct control {
   UCHAR  num_nodos ;    /* número de nodos en un renglón */
   UCHAR  es_de_op ;     /* indica si hay operadores */
   OBJPTR obj ;          /* apuntador al primer objeto */
   OBJPTR last ;         /* apuntador al último objeto */
} CTRL ;

CTRL ctr[24] ;           /* estructura de control */
```

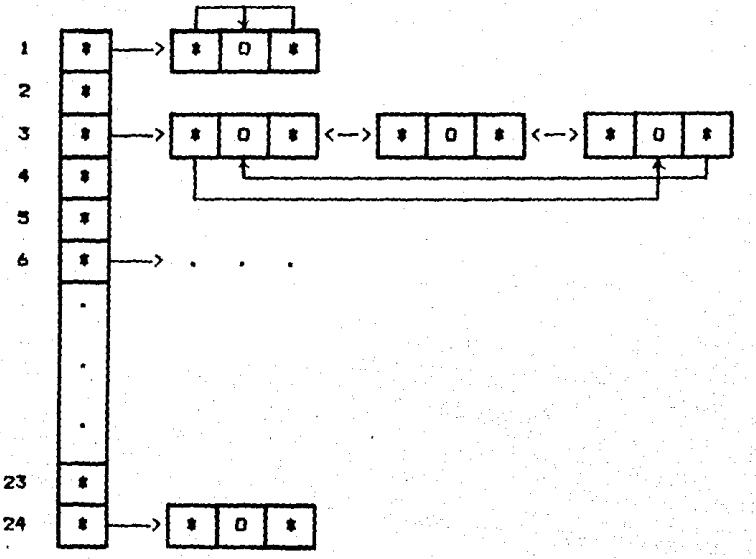


Figura 3.4 Estructura de control de los objetos

Texto

Como su nombre lo indica, hacer uso de este objeto permite especificar la información que describe todo un contexto determinado. Tal información puede ser: encabezados y texto asociado con una variable ("prompts"). La estructura de este tipo de objeto es:

```
struct texto {
     char    *cadena ;          /* texto deseado */
     struct ajespole comun* ; /*ajuste, estilo, pos, long*/
} ;
```

Variables

Por medio de ellas se especifica el campo a observar. Se cuenta con dos tipos de variables: de entrada y salida (E/S), y de salida (S). Las variables de entrada y salida corresponden únicamente al archivo de datos principal, mientras que las de salida corresponden a los archivos secundarios. La estructura de este tipo de objeto es la siguiente:

```
struct variable {
     char    *name ;            /* nombre del campo */
     struct ajespole comun ; /* ajuste, estilo, posi, long*/
     char    tipo ;             /* tipo de valores del campo */
     UCHAR   file ;             /* descriptor del archivo */
     UCHAR   deci ;           /* precisión (campos numéricos)*/
     float   lim_inf ;        /* límite inferior (numéricos) */
     float   lim_sup ;        /* límite superior (numéricos) */
     char    *men_err ;       /* mensaje de error */
} ;
```

---

*esta estructura es común a los cuatro tipos de objetos. Su descripción es la siguiente:

```
struct ajespole {
     char  ajuste ; /*Normal, izquierda, derecha, centrado*/
     char  estilo ; /* Normal, inverso, subrayado */
     UCHAR colum ;  /* posición inicial */
     UCHAR p_f ;    /* posición final */
     UCHAR longit ; /* longitud */
}
```

Expresiones

Con este tipo de objeto el usuario podrá hacer uso de expresiones aritméticas comúnes que involucren a algunos de los campos del archivo principal. La expresión se especifica en notación infija, luego se compila a postfija y se evalúa. La estructura de este tipo de objeto es la siguiente:

```
struct funcion {
     char    *infix ;          /* expresión infija */
     struct ajespole comun ; /* ajuste, estilo, posi, long*/
     char    tipo ;            /* lógica o numérica */
     UCHAR   valong ;          /* longitud del valor */
     UCHAR   deci ;            /* precisión */
) ;
```

Operadores

El uso de este objeto permite el cálculo de operaciones como totales, promedios, máximos, mínimos y contadores. Si la definición de la vista comprende a la cláusula de agrupación, los operadores que se hayan especificado serán calculados de acuerdo con el agrupamiento establecido. Si, aunado al elemento de agrupación, la definición de la vista contiene el elemento predicado, el cálculo comprenderá a las ocurrencias del archivo principal que satisfagan al predicado. La estructura de este tipo de objeto es la siguiente:

```
struct operador {
     char   *de ;  /* campo sobre el que actua el operador*/
     struct ejespole comun ; /* ajuste, estilo, posi, long*/
     char   tipo ; /* Contador, máx, min, promedio, total */
     UCHAR  deci ;           /* precisión */
     char   *txt_asoc ;      /* texto asociado al operador*/
     double valor ;          /* valor del operador */
} ;
```

Predicado

Por medio de un predicado se pretende observar, de entre todos los datos, solamente a aquéllos que la cumplan. La especificación del predicado consiste en una expresión infija cualquiera, cuyo resultado es de tipo booleano. El manejo del predicado se hace con la siguiente variable:

```
char  *condicion ;
```

Cláusula de agrupación

La inclusión de este elemento en la definición de la vista permite agrupar aquellas ocurrencias que coincidan en cierta información, en cuanto a su valor. La especificación del agrupamiento se hace indicando el campo (o la concatenación de campos) sobre el cual se desea efectuar la agrupación de las ocurrencias. El manejo de la cláusula de agrupación se realiza a través de la siguiente variable:

```
char  *agrupar ;
```

Se presenta a continuación, la representación binaria de todos los elementos previamente descritos.

BYTE	CONTENIDO	DESCRIPCION
0	8 bits (número)	número de cajas
1-2	16 bits (número)	longitud de la vista
3-14	12 bytes	vacio (uso posterior)
15	8 bits (número)	número de archivos
16-63	48 bytes	nombres de archivos
64-111	48 bytes	agrupación
112-160	48 bytes	predicado
161-n	xx bytes c/u	descripción de cajas
n+1	1 byte	06_H (fin de cajas)
n+2-n+25	24 bytes	num. de objetos/linea
n+26-n+27	16 bits (número)	longitud de objetos
n+28-x	80 bytes c/u	descrip. de objetos
x+1	1 byte	$0D_H$ (fin de objetos)
x+2	1 byte	$1A_H$ (fin de vista)

Descripción de cajas: <

BYTE	CONTENIDO	DESCRIPCION
0-3	4 byte	screen, no_op, x1, x2
4	1 byte	54_H (fin de caja)

Descripción de objetos: <

BYTE	CONTENIDO	DESCRIPCION
0	1 byte	tipo de objeto
1-61	61 bytes	texto,campo o expresión
62-66	5 bytes	atributos comunes

67	1 byte	tipo de la variable
68	1 byte	descriptor del archivo
69	1 byte	precisión numérica
70-73	4 bytes	límite inferior
74-77	4 bytes	límite superior

Para concluir, la figura 3.5 muestra la estructura de los elementos que conforman la vista por medio del método de Jackson.

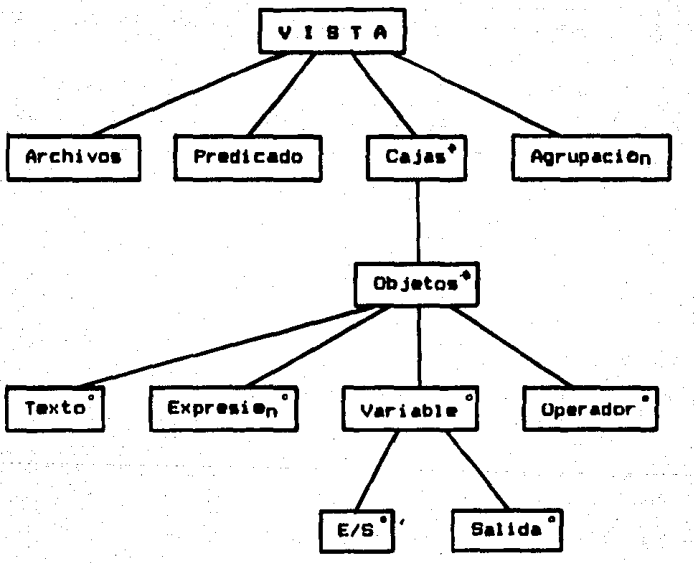


Figura 3.5 Estructura de la vista

Todos estos elementos que han sido descritos (tanto de índole geomètrica como lôgica) son almacenados en un archivo en disco -ver figura 3.6-, el cual serà la entrada para el segundo componente del sistema: el manejador de la pantalla.

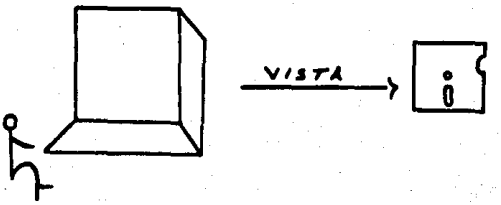


Figura 3.6 Uso del editor

3.2 Interfaz con el DBMS

Uno de los principales objetivos en el desarrollo del sistema fue su **portabilidad.** Esto es, se pretende que el sistema pueda trabajar conjuntamente con cualquier DBMS sin que ello implique hacer modificaciones de tipo estructural al sistema. ¿Cômo lograr tal propôsito? Basando el diseño de la interfaz en un conjunto de rutinas denominadas **primitivas.**

En el editor, la interfaz con el DBMS es requerida para dos cosas: la obtención del esquema de cada uno de los archivos que conforman la vista, y la determinación de si existe o no relación entre los archivos con los cuales se desea definir una vista (tal determinación se hace en base a

los esquemas obtenidos previamente).

Para la satisfacción de ambos requerimientos fue necesario controlar adecuadamente la información de cada archivo, siendo ésta la causa de la definición de la siguiente estructura:

```
typedef struct archivos {
   char      file[13] ;   /* nombre del archivo */
   char      key[128] ;   /* llave primaria */
   FILEDESC fd ;          /* descriptor del archivo */
   DICCPTR  ptr ;         /* apuntador al esquema */
   char      *buff_reg ;  /* instancia de un registro */
} ARCH ;
```

y de la siguiente variable:

```
ARCH  df[]    /* información de los archivos */
```

siendo el campo **ptr** -el apuntador al esquema del archivo- el medio por el cual se cumple con el primer requerimiento. Como puede apreciarse, el campo **ptr** es de tipo DICCPTR. La descripción de este tipo se muestra a continuación:

```
typedef struct header {
   UCHAR    version ;      /* versión del DBMSS */
   UCHAR    *fecha ;       /* apuntador a la fecha */
   ulong    numreg ;       /* número de registros */
   ushort   long_rec ;     /* longitud del registro */
   UCHAR    no_fields ;    /* número de campos */
   UCHAR    *f_d ;         /* apuntador a descrip. de campos*/
} DICC, *DICCPTR ;
```

lo cual significa que para cada archivo hay un apuntador a un área en memoria que contiene toda la información

relacionada con ese archivo en particular. Desde la perspectiva de la programación, estas estructuras permiten tener una gran flexibilidad en el manejo de los archivos, ya que lo único que requieren los módulos que necesitan información de alguno de ellos, es la dirección (el apuntador) de la memoria que contiene su información -el esquema, número de registros, etc.-. La primitiva que se encarga de interactuar con el DBMS para la obtención de la información de cada archivo de datos se denomina **forma_dic()**.

La observancia del segundo requerimiento se efectúa en la función **cheq_rel()**, la cual necesita recibir para su ejecución el nombre de los archivos a analizar. Tal información se encuentra en la variable **df[]**. En el siguiente punto se muestra el pseudocódigo de esta función.

3.3 Breve descripción de los módulos principales

Para tener una perspectiva más clara de la estructura básica del editor en cuanto a su desarrollo, se presenta el diagrama de la figura 3.7. Enseguida se hace una breve descripción del funcionamiento de cada uno de los módulos que, como hemos señalado, conforman la estructura básica del editor.

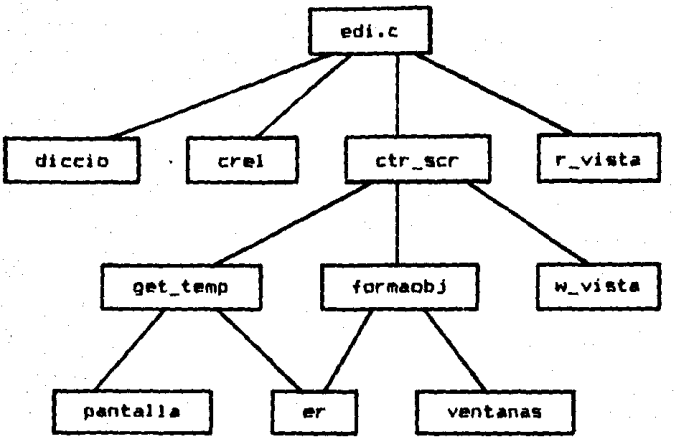


Figura 3.7 Diagrama jerÀrquico del editor

Edi.c

Empieza por obtener la base de datos y los archivos con los cuales se definirà la vista. Posteriormente inicializa unas variables globales, obtiene el esquema de cada archivo y verifica la relaciòn entre los archivos, para -si todo va bien hasta el momento-, invocar al mòdulo que controla la pantalla. Su pseudocòdigo es el siguiente:

```
main()
{
   obtener nombre de la vista ;
   if (no existe la vista)
       pedir los archivos de datos ;

   /* obtener el esquema de c/u de los archivos */
   init_diccio() ;

   /* verificar la relaciòn entre los archivos */
   cheq_rel() ;
```

```
    /* inicializar las estructuras de control */
    init_editor() ;

    /* control de la pantalla */
    ctr_scr() ;
}
```

Diccio.c

En este mòdulo se encuentra la primitiva forma_dic() que se encargarà de interactuar con el DBMS para accesar y llevar a memoria la informaciòn requerida por la estructura DICC para cada archivo. El pseudocòdigo de la primitiva forma_dic() es el siguiente:

```
/*  forma_dic
 *
 *    obtener informaciòn (esquema, etc.) de un archivo
 *
 *  Entrada:
 *
 *    fd       : descriptor del archivo
 *
 *  Salida:
 *
 *    apuntador al àrea de memoria que contiene toda la
 *    informaciòn del archivo
*/

DICCPTR  forma_dic(fd, fi)
FILEDESC fd ;
{
    DICCPTR  rtp ; /* apuntador al esquema */

    rtp = (DICC *)malloc(sizeof(DICC)) ;

    /* versiòn del DBMS */
    rtp->version = get_ver(fd) ;

    /* fecha de la ùltima actualizaciòn del archivo */
    rtp->fecha = last_up(fd) ;

    /* nùmero de registros */
    rtp->numreg = get_rec(fd) ;
```

```
    /* longitud del registro */
    rtp->long_rec = get_longrec(fd) ;

    /* nùmero de campos */
    rtp->no_fields = n_field(rtp->long_head) ;

    /* descripciòn de los campos */
    rtp->f_d = get_fields(fd, rtp->no_fields) ;
    return(rtp) ;
}
```

Crel.c

En este mòdulo se encuentra la funciòn cheq_rel() que se encargarà de verificar la relaciòn entre los archivos que se han declarado para la vista. El pseudocòdigo de esta funciòn es el siguiente:

```
/*  cheq_rel
 *
 *    verificar la relaciòn entre los archivos
*/

void cheq_rel()
{
    UCHAR  hay_error = NO ;

    for (i=1 ; i < no_df ; ++i) {
         hay_rel = rel_cheq(i) ;
         if (no hay_rel) {
             printf("\nEl archivo %s ", df[i].file) ;
             printf("no tiene relacion con el principal") ;
             hay_error = SI ;
         }
    }
    if (hay_error)
        printf("\nNo podemos seguir trabajando") ;
        exit(0) ;
    }
}

/*  rel_cheq
 *
 *    checar la relaciòn del principal con el secundario
 *
 *  Entrada
```

```
 *
 *    i              : descriptor del archivo
*/

UCHAR     rel_cheq(i)
FILEDESC  i ;
{
   /* ¿la llave del archivo secundario se encuentra en */
   /* el archivo principal? */
   if (sec_in_main(i))
       return(SI) ;

    else

    /* ¿la llave del archivo principal se encuentra en */
    /* el archivo secundario? */
       return(main_in_sec(i)) ;
}
```

R_vista.c

Leer del disco la vista -archivo.scr-, formar la estructura de control (ctr[]) y presentar al usuario la vista para que pueda editarla. Estas tres funciones son realizadas por los procedimientos r_vista(), formstruct() y muestra_vista(), respectivamente. A continuaciòn se muestra el pseudocòdigo de la funciòn r_vista():

```
/*  r_vista
 *
 *    leer del disco la vista
 *
 *  Entrada
 *
 *    arrobj: arreglo de direcciones a objetos que deberà
 *            contener las direcciones de los objetos
 *            requeridos, al leer la descripciòn de los
*             mismos
*/

void  r_vista(arrobj)
OBJPTR arrobj[] ;
{
    leer nùmero de cajas ;
    leer nombres de los archivos ;
```

```
    leer agrupación y predicado ;
    leer descripción de cajas ;
    leer número de objetos por línea ;
    leer descripción de objetos ; /* actualiza arrobj[] */
)
```

Ctr_scr.c

Este módulo tiene el control total de la pantalla, lo cual equivale a decir que es el encargado de la interfaz con el usuario. Es aquí donde el usuario podrá declarar el uso de las cajas, los objetos, el predicado y el agrupamiento, y almacenar la vista. Como se puede apreciar, aquí esta el núcleo del editor. El pseudocódigo de la rutina ctr_scr() es el siguiente:

```
/* ctr_scr
 *
 *   permitir al usuario definir la geometría, así como
 *   también la lógica de la vista
 *
*/

void  ctr_scr()
{
   UCHAR  continua = SI ;

   while (continua) {
       comando = getchar() ;
       switch (comando) {
       case flechas:
       case tabs:
       case espacio:
          situar el cursor en la línea y columna adecuada
          dependiendo de la posición en que se encuentre;

       case TEXTO:
       case VARIABLE:
       case EXPRESION:
       case OPERADOR:
          insertar_objeto(comando) ;

       case CAJA:
          obtener_caja() ;
```

```
        case GUARDAR:
           escribir_vista() ;

        case F1:
           ayuda() ;

        case F2:
           editar_objeto() ;

        case F3:
           siguiente_objeto(IZQUIERDA) ;

        case F4:
           siguiente_objeto(DERECHA) ;

        case F5:
           situar el cursor en el primer objeto de la línea

        case F6:
           situar el cursor en el último objeto de la línea

        case F7:
           obtener_agrupaciòn() ;

        case F8:
           obtener_predicado() ;

        case TERMINAR:
           continua = NO ;
        )
    )
)
```

Get_temp.c

Su objetivo es definir una nueva caja e indicar al usuario en què caja se encuentra, así como obtener dos de los elementos para la definiciòn de la vista: el agrupamiento y la condiciòn. Las funciones que conforman este mòdulo son:

```
    * get_TE()          /* obtener una nueva caja */

    * get_agrupacion()  /* obtener agrupamiento */

    * get_condicion()   /* obtener predicado */
```

Formaobj.c

Aquí se tiene el control de la inserción de los cuatro tipos de objetos. La definición de los objetos texto y variable se hace aquí mismo, mientras que la de los otros dos se efectúa en el módulo expr_op.c. A continuación se muestra el pseudocódigo de la función insert_obj().

```
/*  insert_obj
 *
 *    insertar un objeto del tipo deseado en la estructura
 *    de control (ctr[])
 *
 *  Entrada:
 *
 *    t_o: objeto deseado (Texto, Var., Eexpr., Oper.)
*/

void insert_obj(t_o)
char t_o ;
{
   OBJPTR  objeto ;

   /* validar la posición de inserción del objeto,
    * para evitar traslapes */
   if (no es valid_posi())
       return ;

   /* solicitar memoria para el objeto */
   objeto = (OBJ *)malloc(sizeof(OBJ)) ;

   /* pedir los datos correspondientes a cada tipo
    * de objeto */
   switch (t_o) {
   case TEXTO:
      pide_text(objeto) ;
   case VARIABLE:
      pide_var_io(objeto) ;
   case EXPRESION:
      pide_expresion(objeto) ;
   case OPERADOR:
      pide_operador(objeto) ;
   }
```

```
    /* meter el objeto en la estructura de control (ctr[]).
     * Especificamente, en la lista doblemente ligada
     * que corresponde a la linea en que se define el objeto
     * (ctr[linea].obj = objeto) */
    mete_objeto(objeto) ;
)
```

W_vista.c

Envia la defincion de la vista a disco en el formato adecuado. El pseudocòdigo de la funciòn w_vista() es el siguiente():

```
/*  w_vista
 *
 *    guardar la vista en un archivo
 *
*/

void  w_vista()
{
    escribir nùmero de cajas ;
    escribir nombres de los archivos ;
    escribir agrupaciòn y predicado ;
    escribir descripciòn de cajas ;
    escribir nùmero de objetos por linea ;
    escribir descripciòn de objetos ;
)
```

Pantalla.c

Control de las cajas para un manejo òptimo de la pantalla, a travès de rutinas que interactùan con otras desarrolladas en ensamblador.

Er.c

Es el analizador sintàctico de una expresiòn regular, que se compila de notaciòn infija a postfija. Interactùa con los

módulos get_temp.c y formaobj.c (a través de expr_op.c) en vista de que en get_temp.c se usa para compilar el predicado y en expr_op.c para compilar una expresión aritmética.

Ventanas.c

Presenta los archivos y los esquemas de los mismos a través de ventanas, para que el usuario pueda seleccionar el campo del archivo deseado. Las funciones de este módulo también hacen uso de las rutinas hechas en ensamblador.

4. DISEÑO DEL MANEJADOR DE LA PANTALLA

La interfaz del usuario es el programa.
-Dr. Alan Kay,
Apple Computer Corporation

La interfaz del usuario es la parte más crítica e importante de un programa de aplicación, ya que es lo único que el usuario puede ver -lo que le interesa- y por lo tanto juzgar. Tener presente lo anterior, fue fundamental en el diseño del manejador de la pantalla (MP).

El MP es el segundo componente del sistema. Mientras que la función del editor es definir la vista, la función del MP es permitir no sólo la observancia de los datos, sino también su actualización a través de la vista previamente definida. El objetivo de este capítulo es mostrar cómo realiza su función el manejador de la pantalla. Se presenta la estructura del MP enfatizando tanto las estructuras de los datos en que se fundamenta, así como la relación e independencia (acoplamiento y cohesión) entre sus módulos.

Posteriormente, se observa la interfaz con el diccionario de datos. Aquí nos interesa, además de obtener el esquema de cada uno de los archivos, conocer qué tipos de datos maneja el DBMS así como la forma en que los almacena. Asimismo, necesitamos unas primitivas que nos permitan extraer y

actualizar la información de la base de datos.

4.1 Estructura del manejador de la pantalla

El objetivo primordial del MP es consultar y editar la base de datos. La especificación de la información sobre la que actúa el MP está contenida en la definición de la vista. La figura 4.1 ilustra su funcionamiento de una manera simple y llana.

Figura 4.1 Funcionamiento del Manejador de la Pantalla

Antes de pasar a la descripción del MP, es importante señalar que todas las estructuras definidas para el diseño del editor (capítulo dos), también juegan un papel primordial en el desarrollo del MP. Tales estructuras son: TEs (manejo de cajas), CTRL (estructura de control), OBJ (descripción de objetos), ARCH (manejo de archivos), DICC (esquema del archivo).

Una vez que ha sido señalado el funcionamiento del MP, surgen dos preguntas básicas: ¿Cómo realiza su trabajo el

MP?, y ¿Cuàles son los mòdulos (y las relaciones entre ellos) que permiten el funcionamiento del MP? Para dar respuesta a la primer pregunta, se presenta a continuaciòn el pseudocòdigo del MP.

```
/*  man_scr
 *
 *    manejador de la pantalla
 *
 *  Entrada:
 *
 *    comm: comandos deseados
 *
*/

void  man_scr(comm)
char  *comm ;
{
    leer la vista ;
    verificar la existencia de la base y
              los archivos de datos ;
    obtener los esquemas de c/u de los archivos ;
    if (hay agrupamiento)
        ordenar(principal, claus_agrup) ;
    mostrar la vista ;
    modo_edit = (*comm = 'R') ? NO : SI ;
    if (mod_edit)
        ediciòn(comm) ;
    else
        reportes() ;
}
```

Como se puede apreciar en el pseudocòdigo, lo primero que hace el MP es determinar sobre què informaciòn va a trabajar. Para tal efecto, lee la vista, verifica la existencia de los archivos, y obtiene el esquema de cada uno de ellos. Posteriormente (si todo va bien hasta el momento), en caso de existir la clàusula de agrupaciòn en la

definición de la vista, se clasifica al archivo principal de acuerdo a la cláusula de agrupación. Por último, se muestra la vista en la pantalla y, de acuerdo a la cadena de comandos recibida, se determina si el manejador va a trabajar en modo de edición o de reportes.

En la figura 4.2 se muestran los módulos que conforman el MP. La función que realiza cada uno de ellos es la siguiente:

* diccio.c: obtener el esquema de cada uno de los archivos involucrados en la vista.
* edición.c: permitir la actualización de la base de datos.
* reportes.c: permitir únicamente la observancia de los datos.
* r_vista.c: leer y mostrar la vista
* índices.c: manejar los índices de los archivos.
* er.c: compilar y evaluar expresiones regulares.
* buffreg.c: contiene a la primitiva que se encarga de llenar el "buffer" de cada archivo.
* tipos.c: contiene las primitivas para poder manejar como cadenas los tipos de datos.
* desplaz.c: calcular el desplazamiento que corresponde al i-ésimo campo.

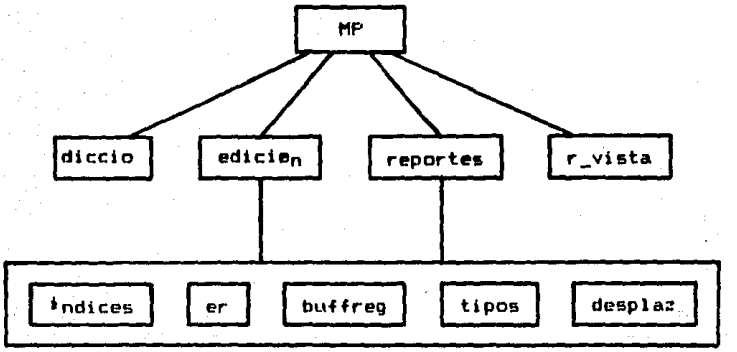


Figura 4.2 Diagrama jerÀrquico del MP.

La descripciòn de los mòdulos diccio.c y r_vista.c se hace en el capítulo tres, ya que también son requeridos por el editor para la definiciòn de la vista. De la explicaciòn de los mòdulos ediciòn.c y reportes.c, se encargan los siguientes puntos; sin embargo, es importante apreciar que ambos (ediciòn y reportes) se apoyan en un conjunto de mòdulos bàsicos que son fundamentales para su funcionamiento. El trabajo específico de cada uno de los mòdulos que conforman este conjunto bàsico, fue un factor determinante en cuanto al **acoplamiento** del sistema. El mòdulo índices.c se ocupa de manejar los índices[1] de los

[1]El objetivo del mòdulo se logra a travès de dos funciones: quicksort(), que clasifica a los registros por la llave que se le indique, y binsrch(), que realiza una bùsqueda binaria para localizar a los registros; por esta razòn, el nùmero màximo de registros en un archivo es de 50 elementos. Para observar una aplicaciòn màs "real" del sistema, lo ùnico que habría que hacer es reemplazar el mòdulo con un c-isam o un b-tree, con lo cual se demuestra la independencia (cohesiòn, de acuerdo a la Ingeniería de Software) del mòdulo.

archivos. El módulo er.c se encarga de compilar y evaluar expresiones.

En el módulo buffreg.c se encuentra la primitiva que se encarga de obtener la instancia del registro que se le indique. La función requiere dos parámetros: el descriptor del archivo y el número de registro que se desea observar. El módulo tipos.c contiene a las primitivas **val_ascii()** y **ascii_val()** las cuales permiten, dado un valor y su tipo, transformarlo en cadena de caracteres y viceversa, respectivamente. Por último, el módulo desplaz.c contine dos funciones: offset_fld() calcula el desplazamiento (en el registro) del i-ésimo campo, y la otra, iesimo_fld() retorna el número de campo que es la cadena que recibe (en caso de que la cadena recibida sea un campo). A continuación se muestra el pseudocódigo de la función desplaz().

```
/*  offset_fld
 *
 *    obtener el desplazamiento del i-ésimo campo
 *
 *  Entrada
 *
 *    fd       : descriptor del archivo
 *    numcampo: número de campo
*/

ushort    offset_fld(fd, numcampo)
FILEDESC  fd ;
UCHAR     numcampo ;
{
```

```
    ushort desplaz = 0 ;

    for (i=0 ; i < numcampo ; ++i) {
        long_fld = longitud del campo i del archivo fd ;
        desplaz += long_fld ;
    }
}
```

4.1.1 Reportes

Las funciones que realiza esta parte del MP son dos: obtener la información de la BD y presentarla al usuario en la forma que se indica en la vista.

Para la obtención de la información, el campo **buff_reg** de la estructura ARCH es fundamental (líneas abajo, la estructura ARCH es nuevamente mostrada). Como se puede apreciar, buff_reg es un apuntador a caracteres (o bytes).

```
typedef struct archivos {
   char      file[13] ;     /* nombre del archivo */
   char      key[128] ;     /* llave primaria */
   FILEDESC fd ;            /* descriptor del archivo */
   DICCPTR  ptr ;           /* apuntador al esquema */
   char     *buff_reg ;     /* instancia de un registro */
}
```

Para cada archivo de datos que forme parte de la definición de la vista, se solicita un área en memoria cuyo tamaño es determinado por la longitud del registro. La dirección de la memoria otorgada se encuentra en el campo buff_reg. Cuando se solicita la observancia de un registro, la primitiva **buffreg()** es la encargada de interactuar con la base de datos para poder depositar tal registro en el

espacio de memoria señalado por buff_reg. Una vez que se ha llevado a memoria la instancia del registro solicitado, es a partir de aquí -buff_reg- que se obtienen los valores de los campos que se declararon en la vista. El pseudocódigo de la función reportes() es el siguiente:

```
/*  reportes
 *
 *    el MP en modo de reportes
 *
 *  Entrada:
 *
 *    fd            : descriptor del archivo
*/

void  reportes(fd)
FILEDESC fd ;
{
   ulong  no_reg ;     /* número de registro a observar */
   ulong  cc_reg ;     /* contador de registros */
   char   var_val[256] ;  /* valor de un campo */

   while (~ EOF(fd)) {
       ++cc_reg ;
       no_reg = pide_indice() ;
       llena_buffs_regs(no_reg) ;
       if (hay_condicion)
           if (no cumple_condicion())
               continuar con otro registro ;
       /* mostrar c/u de las cajas con sus repectivos
        * objetos */
       muestra_TEs(fd, var_val) ;
   }
}
```

Tal y como lo ilustra el pseudocódigo de la función, el trabajo de reportes() es sencillo. Para cada registro del archivo principal se hacen tres cosas: obtener los registros de los archivos secundarios con los cuales está relacionado (de ello se encarga llena_buffs_regs()), verificar si cumple

la condición (en caso de que exista la condición en la definición de la vista) y mostrar cada una de las cajas con sus respectivos objetos "dinámicos" (variables, expresiones y operadores). La figura 4.3 presenta la jerarquía de los módulos para el funcionamiento de reportes().

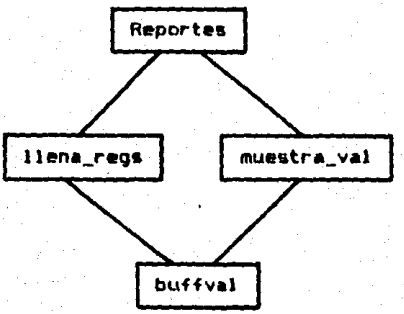


Figura 4.3 Diagrama jerárquico de reportes.

Se hace a continuación una breve descripción de cada uno de los módulos.

Llena_regs.c

Este módulos se encarga de obtener las instancias de los registros del archivo principal y secundarios. Las instancias de los secundarios son registros que están relacionados con la instancia del registro del archivo principal. El pseudocódigo de la función llena_buffs_regs() es el siguiente:

```
/*  llena_buffs_regs
 *
 *    obtiene la instancia de los registros de los archivos
 *    principal y secundarios
 *
 *  Entrada:
 *
 *    num_reg: número de registro del archivo principal
 *
*/

void  llena_buffs_regs(num_reg)
ulong num_reg ;
{
    /* se llena el buffer del principal por medio
       de la primitiva buffreg() */
    buffreg(fd_principal, num_reg) ;

    para cada archivo secundario {
        /* se obtiene la llave del secundario */
        val_llave_sec = dame_val_llave(a partir de la
           instancia del registro del archivo principal) ;

        /* se obtiene el número de registro del secundario*/
        num_reg_sec = pide_indice(fd_sec, val_llave_sec) ;

        /* se llena el buffer del secundario */
        buffreg(fd_sec, num_reg_sec) ;
    }
}
```

Muestra_val.c

La responsabilidad de este módulo es mostrar para cada una de las cajas, los valores correspondientes a cada uno de los objetos. El pseudocódigo de muestra_TEs() es el siguiente:

```
/*  muestra_TEs
 *
 *    mostrar los valores de los objetos de las cajas que
 *    conformen la definición de la vista.
 *
*/

void  muestra_TEs()
{
```

```
    OBJPTR objeto ;
    TEPTR  rtp = te ;  /* dirección del nodo cabeza */

    do {
       rtp = rtp->rte ; /* dirección de la primera caja */
       for (i=rtp->x1 ; i < rtp->x2 ; ++i) {
          obj_lin = ctr[i].num_nodos ;
          objeto = ctr[i].obj ;  /* ier. objeto/línea */
          if (rtp->screen = 'S')  /* manejo de la pantalla*/
              scrollbox() ;
          for (j=0 ; j < obj_lin ; ++j) {
             switch (objeto->tip_obj) {
             case TEXTO:
                break ;
             case ENTRADA:
             case SALIDA:
                dame_val_campo(var_name(objeto)) ;
                mostrar_valor(ren, col, var_val) ;
                break ;
             case EXPRESION:
                copy(exp_infix(objeto), infix) ;
                er(var_val) ;
                mostrar_valor(ren, col, var_val) ;
                break ;
             case OPERADOR:
                pos_cur(ren, col) ;  /* situar cursor */
                actualiza_operador(objeto) ;
                break ;
             }
             objeto = objeto->forw ;
          }
       }
       rtp = rtp->rte ;   /* siguiente caja */
    } while (rtp != te) ;
}
```

Buffval.c

El objetivo del módulo es obtener el valor de las llaves de los archivos así com el valor de los campos. Lo peculiar del módulo radica en que la obtención de los valores se realiza por medio del campo $buff_reg de la estructura ARCH, correspondiente a cada uno de los archivos involucrados en la vista. Este módulo consta de dos funciones: buff_keyval() y buff_campoval(); puesto que para obtener el valor de una

llave, se requiere de la obtención del valor de uno o varios campos, se muestra a continuación el pseudocódigo de buff_campoval().

```
/* buff_campoval
 *
 *   dar la dirección (en el parámetro "valor") del valor
 *   del campo deseado haciendo uso del campo *buff_reg de
 *   la estructura ARCH.
 *
 *  Entrada
 *
 *    fd    : descriptor del archivo
 *    campo: campo a obtener
 *    valor: dirección del valor del campo en buff_reg
 *
 *  Salida
 *
 *    lonfld: longitud del campo
 *
*/

UCHAR   buff_campoval(fd, campo, valor)
FILDESC fd ;
char    *campo ;
char   **valor ;
{
    ushort  desplaz ;
    UCHAR   long_campo ;
    UCHAR   num_campo ;

    num_campo = iesimo_fld(fd, campo) ;
    long_campo = long_fld(fd, num_campo) ;
    desplaz = offset_fld(fd, num_campo) ;

    /* dirección del valor del campo en buff_reg */
    *valor = df[fd].buff_reg + desplaz ;
    return(longfld) ;
}
```

4.1.2 Edición

La función de este módulo (edición.c) es permitir la actualización de la base de datos. En el capítulo dos (2.7)

se observaron los problemas que se presentan al actualizar la base de datos a través de vistas. Tales problemas son la causa de restringir la edición de la base de datos sólo al archivo principal de la vista; por esta misma razón tenemos dos tipos de variables: de entrada y salida (archivo principal), y de salida solamante (archivos secundarios).

Para que edición.c cumpla con su función, también hace uso de todas las estructuras que han sido definidas en el capítulo dos (TEs, CTRL, OBJ, ARCH y DICC). Sin embargo, así como el módulo reportes.c se fundamenta en el campo $buff_reg de la estructura ARCH, en este módulo (edición.c) existen dos variables que desempeñan un papel primordial: set_val y set_var. Ambas variables son apuntadores a caracteres y se comportan como cadenas de cadenas. En la variable set_val se almacenan todos los valores que el usuario digita, mientras que en la variable set_var se almacena el nombre de todas las variables de entrada y salida solamente. Sin embargo, ¿cuál es la causa que justifica el uso de tales variables? La respuesta está en que ahora necesitamos obtener los valores de cada variable a partir del conjunto de valores que el usuario especifica, y no a partir del buffer de cada archivo (campo $buffreg de la estructura ARCH). Por lo tanto, a cada cadena de set_var, le corresponde una cadena (que puede ser nula) de set_val; o dicho de otra manera, cada campo con su valor

correspondiente. El pseudocódigo de la función edición() es el siguiente:

```
/*  edición

 *    el manejador de pantalla en modo de edición
 *
 *  Entrada
 *
 *    fd  : descriptor del archivo
 *    comm: comandos deseados
*/

void     edicion(fd, comm)
FILEDESC fd ;
char     *comm ;
{
   set_val = (char *)malloc() ;
   do {
      get_valores(set_val) ;   /* leer los valores */

      /* preguntar por el movimiento que se desea hacer */
      if ((comando = get_comm(comm)) == TERMINA)
           salir del ciclo ;

      else { /* preparar el tipo de movimiento deseado */
           prep_edit(fd, comando, set_val) ;

           /* limpiar de la pantalla los valores de las
            * variables */
           screen_only_var() ;
      }
   } while (comando != TERMINA) ;
}
```

De acuerdo al pseudocódigo presentado, se puede apreciar que edicion() empieza por obtener memoria para poder alojar los valores que el usuario especifique; la dirección de la memoria otorgada se encuentra en la variable set_val. La segunda instrucción es un ciclo en el cual se van a realizar dos cosas: obtener los valores por parte del usuario

(get_valores()) y ejecutar el comando (obtenido por get_comm()) que indique el usuario. La función prep_edit() se encarga de preparar y verificar la ejecución del comando. Cada vez que se ejecuta un comando, se "limpia" (borrar) en la pantalla el valor correspondiente a cada una de las variables. La figura 4.4 muestra la jerarquía de los módulos para el funcionamiento de edicion().

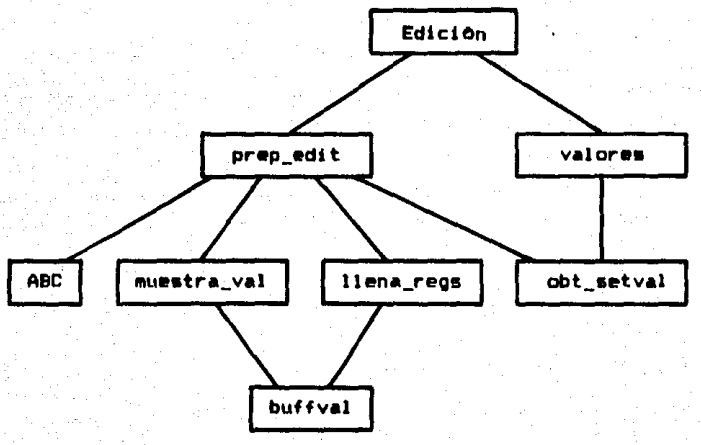


Figura 4.4 Diagrama jerárquico de edición

Se hace a continuación una breve descripción de cada uno de los modulos.

Valores.c

Este módulo es el encargado de obtener los valores que el usuario especifica al interactuar con la pantalla. Dichos valores corresponden a las variables de entrada y salida. Además de realizar lo anterior, el módulo también se ocupa de mostrar los valores de los otros objetos (variables de salida, expresiones y operadores) que dependen del valor de las variables de entrada y salida. El pseudocódigo de la función get_valores() se muestra a continuación.

```
/*  get_valores
 *
 *    obtener los valores de las variables de E/S,
 *    actualizando a los demás objetos.
*/

void get_valores()
{
    OBJPTR objeto ;
    TEPTR  rtp = te ;  /* dirección del nodo cabeza */
    char   valor[] ;   /* valor a obtener */

    do {
       rtp = rtp->rte ; /* dirección de la primera caja */
       for (i=rtp->x1 ; i < rtp->x2 ; ++i) {
          obj_lin = ctr[i].num_nodos ;
          objeto = ctr[i].obj ;  /* 1er. objeto/línea */
          if (rtp->screen = 'S')  /* manejo de la pantalla*/
              scrollbox() ;
          for (j=0 ; j < obj_lin ; ++j) {
             switch (objeto->tip_obj) {
             case TEXTO:
                break ;
             case ENTRADA:
                get_E_var(objeto, valor) ;
                copiar(valor, set_val) ;
                break ;
             case SALIDA:
                muestra_S_var(objeto, valor) ;
                break ;
```

```
            case EXPRESION:
               muestra_expresion(objeto, valor) ;
               break ;
            case OPERADOR:
               pos_cur(ren, col) ;  /* situar cursor */
               actualiza_operador(objeto) ;
               break ;
            }
            objeto = objeto->forw ;
         }
      }
      rtp = rtp->rte ;    /* siguiente caja */
   } while (rtp != te) ;
}
```

Prep_edit.c

El objetivo del módulo es preparar y verificar que todo marche correctamente para la actualización de la base de datos. La función principal del módulo es prep_edit(). Su trabajo se resume en obtener el número de registro a partir de la variable set_val, y posteriormente invocar a la función que corresponda al comando recibido. El pseudocódigo de la función es el siguiente:

```
/*  prep_edit
 *
 *    preparar la actualización de la base de datos
 *
 *  Entrada
 *
 *    fd      : descriptor del archivo
 *    comm    : comando
 *    set_val: valores de los datos
*/
*
void     prep_edit(fd, comm, set_val)
FILEDESC fd ;
char     comm ;
char    *set_val ;
{
```

```
    char  val_llave[256] ;

    num_reg = busca_llave(fd, set_val, val_llave) ;
    switch (comm) {
    case ALTAS:
        if (num_reg)
            mensaje("llave repetida") ;
        else
            altas(fd, set_val) ;
        break ;
    case BAJAS:
        llena_buffs_regs(num_reg) ;
        muestra_TEs(fd) ;
        if (confirma eliminaciòn)
            bajas(fd, num_reg) ;
        break ;
    case CAMBIOS:
        llena_buffs_regs(num_reg) ;
        muestra_TEs(fd) ;
        get_valores() ;
        if (confirma modificaciòn)
            cambios(fd, num_reg, set_val) ;
        break ;
    case CONSULTA:
        llena_buffs_regs(num_reg) ;
        muestra_TEs(fd) ;
        break ;
    }
}
```

Obt_setval.c

La responsabilidad de este mòdulo es obtener el valor de la llave a partir del conjunto de valores que proporciona el usuario (y que se encuentran en la variable set_val). Una vez obtenida la llave (en caso de haber sido digitada por el usuario), se busca en el índice y se retorna el número de registro correspondiente a la llave. El pseudocòdigo de la funciòn busca_llave() es el siguiente:

```
/*  busca_llave
 *
 *  Entrada
 *
 *     fd        : descriptor del archivo
 *     set_val   : conjunto de valores
 *     val_llave: aquí se depositará valor de la llave
 */

ulong     busca_llave(fd, set_val, val_llave)
FILEDESC fd ;
char      *set_val ;
char      *val_llave ;
{  ulong  reg ;

   /* obtiene la llave a partir de set_val */
   if ((set_llaveval(set_val, df[fd].key, val_llave)) ==0)
        return(0) ;

   /* busca el valor de la llave en el índice */
   reg = busca_indice(fd, val_llave) ;
   return(reg) ;
}
```

Llena_regs.c

Este módulo se encarga de obtener las instancias de los registros principal y secundarios. El pseudocódigo de la función principal de este módulo (llena_buffs_regs()) se puede observar en la descripción del módulo reportes.c

Muestra_val.c

La responsabilidad de este módulo es mostrar los valores de todos los objetos que conforman la vista. El pseudocódigo de la función principal de este módulo (muestra_TEs()) se puede observar en la descripción del módulo reportes.c

Buffval.c

El objetivo del módulo es obtener el valor de las llaves de los archivos así como el valor de los campos. El pseudocódigo de la función principal de este módulo (buff_campoval()) se puede observar en la descripción del módulo reportes.c

ABC.c

Este módulo contiene a las primitivas altas(), bajas() y cambios(). El desarrollo de tales funciones depende del DBMS que se use, razón por la cual, su descripción es presentada en el siguiente capítulo.

4.2 Interfaz con el DBMS

Como se señaló al inicio de este capítulo, el MP no sólo requiere interactuar con el diccionario de datos del DBMS para la obtención de los esquemas de cada uno de los archivos, sino también conocer qué tipos de datos maneja el DBMS y cómo actualizar y obtener la información de la base de datos.

Para el logro de los requerimientos arriba señalados, necesitamos de dos estructuras de datos para depositar en ellas cierta información, y por otra parte, recurrir a las primitivas. Las primitivas son funciones que se encargarán de llevar la información a las estructuras, así como también

de permitir el manejo adecuado de los datos (tanto para su presentación como para su actualización).

Las estructuras de datos requeridas son ARCH y DICC (su descripción se encuentra en el capitulo dos). En la estructura ARCH se encuentra información de cada uno de los archivos involucrados en la vista; sus campos contienen la llave del archivo, el apuntador al esquema del archivo y una área en memoria ("buffer") que permite almacenar la instancia de un registro. Por otra parte, la estructura DICC contiene el esquema para cada uno de los archivos.

La primitiva que se encarga de obtener el esquema de los archivos es forma_dic(). Para que deposite el esquema en la estructura DICC, recibe el descriptor del archivo cuyo esquema se desea obtener. Por lo que toca a las primitivas para el manejo adecuado de los datos, se desarrollaron las siguientes funciones:

* llena_buff_reg: llenar el "buffer" de cada archivo.
* val_ascii: dado un valor y su tipo, transformarlo en cadena de caracteres
* ascii_val: dado una cadena de caracteres y su tipo, tranformarla en el valor adecuado.
* altas: permitir añadir un registro
* bajas: permitir eliminar un registro
* cambios: permitir modificar un registro

La descripción de estas primitivas se presenta en el siguiente capítulo, ya que es ahí donde se observa la implantación del sistema con dos manejadores de información (Informix y dBASE) de uso común.

5

CASOS DE ESTUDIO

En vista de que el trabajo aquí expuesto lo podemos clasificar como un trabajo teórico-práctico, es necesario observar su comportamiento real. Para tal efecto, se ha seleccionado a dos manejadores de datos de amplio uso hoy en día: Informix y dBASE. En ambos casos, lo que principalmente nos interesa es obtener el esquema de un archivo, los índices primarios, los tipos de datos (tanto para su adecuada presentación en la pantalla, como para el manejo de los valores nulos) y la forma de accesar la información.

5.1 Informix

Para observar el comportamiento del sistema con Informix, se hace uso del lenguaje ESQL (Embedded Structured Query Language). Este lenguaje permite hacer uso de estatutos o instrucciones de SQL en el código fuente de un lenguaje huésped de programación como C o COBOL (C en nuestro caso). En vista de que la determinación de la información que va a manejar el sistema solamente es conocida en tiempo de ejecución, las instrucciones que necesitamos son aquellas involucradas con el manejo dinámico de los datos. Por lo tanto, antes de proceder a explicar como se desarrolla cada una de las funciones requeridas por el sistema, haremos una breve descripción de las instrucciones de ESQL que fueron utilizadas.

Instrucciones para el manejo de un cursor

Un **cursor** es un apuntador a un renglòn (tupla) dentro de un conjunto de renglones. El renglòn apuntado se denomina "renglòn actual". Desde la perspectiva de un programa en C, un cursor es una estructura (_SQCURSOR) que mantiene toda la informaciòn necesaria para procesar un estatuto SQL. Algunos de los campos controlan la siguiente informaciòn:

* nombre del cursor
* tipo y longitud de las columnas (atributos)
* tipo y longitud de las variables huèsped
* un apuntador al renglòn actual

Los cursores utilizados para procesar estatutos SELECT, se denominan select-cursor, mientras que los que procesan estatutos INSERT se denominan insert-cursor. Las operaciones que se pueden realizar con un select-cursor son las siguientes:

* DECLARE <-- definir un cursor
* OPEN <-- inicializa el SELECT (verifica validez)
* FECTH <-- obtener una tupla
* CLOSE <-- desactivar el cursor

Las operaciones que se puden realizar con un insert-cursor, ademàs de las anteriores, son:

* PUT <---- llevar al buffer una tupla
* FLUSH <---- escribir el buffer

Instrucciones y áreas de datos dinámicas

Las instrucciones dinámicas son utilizadas para poder ejecutar cualesquier instrucción de SQL. A continuación se muestra la sintaxis de cada una de estas instrucciones.

PREPARE stmt **FROM** ("string" | str_ptr)

De acuerdo a su sintaxis, se observa que PREPARE sirve para formar una instrucción (stmt) SQL a partir de una cadena; por lo tanto, verifica la sintáxis, semántica y contexto de la instrucción que se pretende formar.

DESCRIBE stmt **INTO** da_ptr

Esta instrucción obtiene información acerca de los datos que va a procesar la instrucción (stmt) previamente preparada. Específicamente, DESCRIBE asigna espacio para la estructura SQLDA y retorna un apuntador a ella (SQLDA describe las columnas a accesar cuando un SELECT es ejecutado). La dirección del espacio asignado se encuentra en da_ptr.

DECLARE nombre_cursor [**SCROLL**] **CURSOR FOR** stmt

Define un cursor (apuntador a una tupla) para un estatuto preparado.

EXECUTE stmt [**USING** (lista_host | **DESCRIPTOR** ida_ptr)]

Esta instrucción sirve para ejecutar cualquier instrucción diferente de SELECT o INSERT.

Area dinàmica de datos

El àrea dinàmica de datos es un espacio en memoria que describe los datos que se manejan al ejecutar un estatuto previamente preparado. El àrea consiste de tres partes bàsicas:

* una àrea en memoria para alojar a los datos actuales
* un conjunto de estructuras sqlvar_struct (SVs) que describen el àrea
* una estructura sqlda que describe esa colecciòn.

Para trabajar con una àrea dinàmica de datos, se debe declarar un apuntador a la estructura sqlda:

```
struct sqlda  *da_ptr ;
```

La definiciòn de las estructuras sqlvar_struct y sqlda se presenta abajo, mientras que su representaciòn gràfica se muestra en la figura 5.1.

```
struct sqlvar_struct {
    short  sqltype ;    /* tipo de variable */
    short  sqllen ;     /* longitud en bytes */
    char   *sqldata ;   /* apuntador al valor */
    short  *sqlind ;    /* apuntador al indicador de var. */
    char   *sqlname ;   /* nombre de la variable */
    char   *sqlformat ; /*reservado */
) ;

struct sqlda {
    short  sqld ;    /*nùmero de estructuras sqlvar_struct */
    struct sqlvar_struct *sqlvar ;  /* apuntador a SVs */
) ;
```

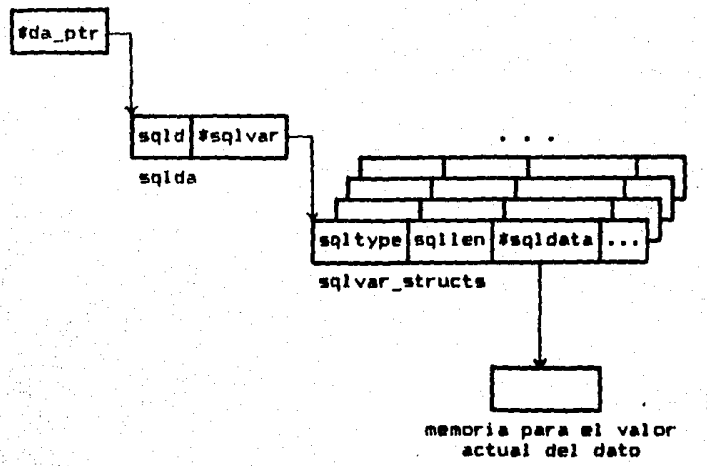


Figura 5.1 Area dinàmica de datos

Una vez que se ha hecho una breve descripciòn de las instrucciones que aparecerân en las funciones requeridas por el sistema, se presenta a continuaciòn la implantaciòn de cada una de las funciones.

Esquema de un archivo

Informix usa un archivo grande (directorio) del sistema operativo para alojar la base de datos. Todas las relaciones (tablas) son almacenadas en este archivo; de esta forma, Informix se encarga de controlar el archivo. Para obtener el esquema de una tabla, habrá que preparar un estatuto de SQL. La instrucciòn que se prepara es:

```
SELECT * FROM nombre_tabla
```

A continuación se muestra el código de las funciones que se ocupan de llevar el esquema de una tabla, a la estructura DICC (la primitiva **forma_dic()** es la que invoca a la función prepara() con el estatuto correspondiente a la tabla deseada).

```
/*  prepara
 *
 *  obtener la estructura SQLDA la cual contiene un conjunto
 *  de estructuras SQLVAR_STRUCT
 *
 *  Entrada
 *
 *     query           :instrucción a preparar
 *
 *  Salida
 *
 *     descripcion     :dirección de la estructura SQLDA
*/

struct sqlda *prepara(query)
char    *query[] ;
{
   struct sqlda *description ;

   /* prepara la instrucción */
   $prepare query_object from $query ;
   if (Status != 0)
       merror("Error durante la preparación", Status) ;

   /* obtener la dirección de la estructura sqlda */
   $describe query_object into descripción ;
   if (Status != 0)
       merror("Error durante la descripción", Status) ;
   return(description) ;
}


/*  get_numflds
 *
 *     obtener el número de campos en el file.
 *
 *  Entrada :
 *
 *     descrip                 :descriptor de sqlda
*/
```

```
UCHAR  get_numflds(descrip)
struct sqlda *descrip ;
{
   return((UCHAR)descrip->sqld) ;
}


/*  get_fields
 *
 *    obtener el esquema de un archivo
 *
 *  Entrada :
 *
 *    descrip                 :descriptor de sqlda
 *    n                       :número de campos
 *
 *  Salida :
 *
 *    apuntador al arreglo descriptor de todos los campos
*/

char   *get_fields(descrip, n)
struct sqlda *descrip ;
UCHAR  n ;
{
   struct sqlvar_struct *col ;
   void   field_name(), field_type_len() ;
   UCHAR  i ;
   char   *ter ;     /* apuntador al área del esquema */

   ter = (char *)calloc((n * 32), sizeof(char)) ;
   for (col = descrip->sqlvar, i=0 ;
        i < n ; col++, i++) {
        char  *rtp ;

        rtp = ter + (32 * i) ;
        field_name(col, rtp) ;          /* nombre */
        rtp += 11 ;
        field_type_len(col, rtp) ;      /* tipo y longitud */
   }
   return(ter) ;
}


/*  field_name
 *
 *    obtener el nombre del campo.
 *
 *  Entrada :
 *
 *    col          :estructura que describe un campo
 *    buf          :aqui se deposita el nombre del campo
*/
```

```
void   field_name(col, buf)
struct sqlvar_struct *col ;
char   *buf ;
{
   void  copiar() ;

   copiar(col->sqlname, buf) ;
}


/*  field_type_len
 *
 *    obtener el tipo y la longitud del campo.
 *
 *  Entrada :
 *
 *    col          ;estructura que describe un campo
 *    buf          ;aqui se coloca la longitud
*/

void   field_type_len(col, buf)
struct sqlvar_struct *col ;
char   *buf ;
{
   /* tamaño en bytes de un tipo de dato SQL. */
   col->sqllen = rtypmsize(col->sqltype) ;
   *buf = col->sqltype ;    /* tipo */
   buf += 5 ;
   *buf = col->sqllen ;     /* longitud */
}
```

Indices primarios

Informix clasifica sus índices como únicos y duplicados. Para uso del sistema, el índice único (índice que contiene la llave primaria del archivo) deberá tener el mismo nombre que el archivo de datos. De esta manera, la obtención de la llave primaria es simple, ya que solo se invoca a la función -que se ocupa de ello y es de la librería de ESQL- con dos parámetros: el nombre del archivo y una cadena en la cual se colocará la llave (df[].file y df[].key, respectivamente).

Tipos de datos

Informix maneja nueve tipos de datos (por el momento, el sistema trabaja con seis). Para cada uno de ellos tiene un tipo de dato equivalente en C. La siguiente tabla ilustra la correspondencia entre los tipos de datos de SQL y los de C.

tipo de dato SQL		tipo de dato C	
declaración	#define	declaración	#define
CHAR(n)	SQLCHAR	char [n]	CFIXCHARTYPE
SMALLINT	SQLSMINT	short int	CSHORTTYPE
INTEGER	SQLINT	long int	CLONGTYPE
SERIAL	SQLSERIAL	long int	CLONGTYPE
SMALLFLOAT	SQLSMFLOAT	float	CFLOATTYPE
FLOAT	SQLFLOAT	double	CDOUBLETYPE
DATE	SQLDATE	long int	CLONGTYPE

Figura 5.2 Correspondencia entre los tipos de datos SQL y C

De acuerdo a la tabla (figura 5.2), se puede apreciar el almacenamiento para cada tipo de dato. Por ejemplo, el tipo de dato SQLDATE es almacenado como un entero largo. La descripción de las primitvas **val_ascii()** y **asciival()** es mostrada abajo.

```
/*  val_ascii
 *
 *  dar el valor en ASCII (en una cadena de caracteres
 *  imprimibles) de acuerdo al tipo del dato
 *
```

```
 *  Entrada
 *
 *     pval : apuntador al valor fuente
 *     tipo : tipo del valor
 *     size : tamaño del valor
 *     valor: valor destino en ASCII
*/

void    val_ascii(pval, tipo, size, valor)
char    *pval ;
UCHAR   tipo ;
UCHAR   size ;
char    *valor ;
{
   switch (tipo) {
   case SQLCHAR:
      copyn(pval, valor, size) ;
      break ;
   case SQLSMINT:
      /* obtener valor numérico */
      numi = *((short int *)pval) ;
      /* convertir el valor a cadena */
      ritos(numi, valor) ;
   case SQLINT:
   case SQLSERIAL:
      numl = *((long int *)pval) ; /* valor numérico */
      rltos(numl, valor) ;  /* convertir el valor a cadena*/
   case SQLSMFLOAT:
   case SQLFLOAT:
      numd = *((double *)pval)) ;  /* valor numérico */
      rdtos(numd, valor) ;  /*convertir el valor a cadena*/
   case SQLDATE:
      /* obtener el número de día */
      numdia = *((long int *)pval) ;
      /* convertir el número de día a cadena */
      datestr(numdia, valor) ;
      break ;
   }
}


/*  ascii_val
 *
 *  dada una cadena de caracteres, obtener el valor
 *  correspondiente al tipo de dato indicado
 *
 *  Entrada
 *
 *     pval : cadena de caracteres
 *     tipo : tipo del valor
 *     size : tamaño del valor
 *     valor: valor destino
*/
```

```
void    ascii_val(pval, tipo, size, valor)
char    *pval ;
UCHAR   tipo ;
UCHAR   size ;
char    *valor ;
{
   switch (tipo) {
   case SQLCHAR:
      copiar(pval, valor, size) ;
      break ;
   case SQLSMINT:
      /* convertir la cadena en entero */
      rstoi(pval, (short int *)valor) ;
   case SQLINT:
   case SQLSERIAL:
      /* converit la cadena en un entero largo */
      rstol(pval, (long int *)valor) ;
   case SQLSMFLOAT:
   case SQLFLOAT:
      /* convertir la cadena en un número real */
      rstod(pval, (double *)valor) ;
   case CDATE:
      /* dirección para colocar el número de día */
      numdia = (long int *)valor ;
      /* convertir la cadena a número de día */
      strdate(pval, numdia) ;
      break ;
   }
}
```

En lo que corresponde al manejo de los valores nulos, Informix hace uso de indicadores de variables. Para poder determinar si una variable (campo) tiene un valor nulo, deberá asociarse un indicador a cada variable. En nuestro caso, el campo sqlind de la estructura sqlvar_struct representa el indicador asociado a la variable (sqlname). Valores negativos en el campo sqlind significan que el valor de la variable al que corresponde el indicador, es nulo.

Acceso a los datos

Las rutinas que se encargan de obtener, insertar, eliminar y modificar las tuplas de una relación son las primitivas buff_reg(), altas(), bajas() y cambios(), y se apoyan en la función prepara(). Ello es así en vista de que lo único que se va a necesitar es preparar las instrucciones SELECT, INSERT, DELETE y UPDATE respectivamente. La descripción de la primitiva buffreg() es la siguiente:

```
/*  buffreg
 *
 *  llenar buff_reg[] con el registro indicado.
*/

void      buffreg(fd, no_reg)
FILEDESC fd ;
ulong     no_reg ;
{
   struct sqlda *prepara() ;
   void   set_bin() ;
   char   query[] ;
   struct sqlda *descrip ;

   /* forma la instrucción */
   formq("SELECT * FROM df[fd].file WHERE ROWID =",
         no_reg, query);

   /* se prepara la instrucción */
   descrip = prepara(query) ;

   /* para cada atributo, se asocia un lugar en memoria
    * en el cual se aloja el valor. En nuestro caso, el
    * espacio en memoria es df[fd].buff_reg */
   setbind(fd, descrip) ;

   /* define un cursor (apuntador) para el SELECT */
   $declare q_cursor cursor for query_object ;

   $open q_cursor ;

   /* se obtiene la tupla */
   $fetch q_cursor using descriptor descrip ;
}
```

```
/*  setbin
 *
 *    para cada atributo, asociar un lugar en memoria en el
 *    cual se aloja el valor
 *
 *  Entrada:
 *
 *    fd    : descriptor del archivo
 *    psqlda: apuntador a la estructura sqlda
*/

void     setbind(fd, psqlda)
FILEDESC fd ;
struct sqlda *psqlda ;
{
   ushort offset_fld() ;
   ushort desplaz ;
   struct sqlvar_struct *col ;
   char   *base = df[fd].buff_reg ;
   int    i ;

   for (col = psqlda->sqlvar, i = 0 ;
        i < psqlda->sqld ;
        col++, i++) )
      /* desplazamiento del i-ésimo campo */
      desplaz = offset_fld(fd, i+1) ;
      /* posición en df[fd].buff_reg */
      col->sqldata = base + desplaz ;
   }
}
```

5.2 dBASE III PLUS

El amplio uso de dBASE se debe básicamente al auge de las computadoras personales. Como señalamos al inicio de este trabajo, es inegable que actualmente la mayoría de las organizaciones (públicas o privadas) disponen de una computadora personal que les auxilie en el manejo de la información; además, si el volumen de la información no es de grandes dimensiones, dBASE es un sistema adecuado para su manejo. A continuación se hace la descripción de cada una de las características de dBASE que nos interesan.

Esquema de un archivo

dBASE almacena cada relación (tabla) en un archivo separado. Asimismo, las tuplas de una relación son almacenadas como registros de longitud fija. En base a lo anterior, se aprecia que las relaciones son "mapeadas" a una estructura de archivos simple. La estructura de un archivo de datos de dBASE se muestra en la figura 5.3.

Encabezado ("header")

BYTE	CONTENIDO	DESCRIPCION
0	1 byte	$version_n$
1-3	3 bytes	fecha ultima modif.
4-7	32 bits (número)	número de registros
8-9	16 bits (número)	longitud del "header"
10-11	16 bits (número)	longitud del registro
12-31	20 bytes	bytes reservados
32-n	32 bytes c/u	arreglo de campos
n+1	1 byte	0_{DH} (fin de campos)

Descripción del arreglo de campos

BYTE	CONTENIDO	DESCRIPCION
0-10	11 bytes	nombre del campo
11	1 byte	tipo (C, N, L, D o M)
12-15	32 bits (número)	dirección del dato
16	1 byte	longitud del campo
17	1 byte	decimales
18-31	14 bytes	bytes reservados

Figura 5.3 Estructura de un archivo de dBASE

De acuerdo a la figura 5.3, se puede apreciar que el esquema del archivo se encuentra en los primeros bytes (byte 32). Para obtener la descripción del esquema se llena un buffer de 512 bytes (tantas veces como sea necesario) el cual es controlado por la función buffile(), y a partir de allí se llevan los datos a la estructura DICC del archivo correspondiente. El pseudocódigo para la obtención del esquema es el siguiente:

```
/*  get_fields
 *
 *    obtener el esquema de u archivo.
 *
 *  Entrada :
 *
 *    fd                        :descriptor del archivo
 *    n                         :número de campos
 *
 *  Salida :
 *
 *    apuntador al arreglo descriptor de todos los campos
*/

char      *get_fields(fd, n)
FILEDESC  fd ;
UCHAR     n ;
{
   void   field_name(), field_type_len() ;
   UCHAR  i ;
   char   *ter ;  /* apuntador al área del esquema */

   ter = (char *)calloc((n * 32), sizeof(char)) ;
   for (i=0 ; i<n ; ++i) {
        char  *rtp ;

        rtp = ter + (32 * i) ;
        field_name(fd, rtp) ;      /* nombre */
        rtp += 11 ;
        field_type_len(fd, rtp) ;     /* tipo */
   }
   return(ter) ;
}
```

```
/*  field_name
 *
 *    obtener el nombre del campo.
 *
 *  Entrada :
 *
 *    fd                        :descriptor del archivo.
 *    buf                       :aqui depositaremos el nombre
*/

void      field_name(fd, buf)
FILEDESC  fd ;
char      *buf ;
{
   char  *buffile() ;
   void  copiar() ;
   char  *ter ;

   /* tomar del buffer[] 11 bytes */
   ter = buffile(fd, 11) ;
   copiar(ter, buf) ;
}


/*  field_type_len
 *
 *    obtener el tipo y la longitud del campo.
 *
 *  Entrada :
 *
 *    fd                        :descriptor del file
 *    buf                       :aqui se coloca la longitud
*/

void      field_type_len(fd, buf)
FILEDESC  fd ;
char      *buf ;
{
   char  *buffile() ;
   char  *ter ;

   /* tomar del buffer[] un byte */
   ter = buffile(fd, 1) ;
   *buf = *ter ;    /* tipo */
   buf += 5 ;
   ter = buffile(fd, 1) ;
   *buf = *ter ;    /* longitud */
}
```

Indices primarios

La observancia de los índices primarios tiene como objetivo conocer la llave primaria -que puede ser un atributo simple o compuesto- de cada archivo de datos. En el caso de dBASE, los índices los maneja en un archivo con extensión ".NDX" y la llave se encuentra a partir del byte 24 del archivo. La llave es una cadena y es obtenida a través de la función getbytes(fd, posi, str, tam), donde fd es el descriptor del archivo, posi es la posición a partir de la cual se van a obtener los bytes, str es la cadena donde se van a depositar (el parámetro actual sería df[fd].key) y, tam es el número de bytes a leer.

Tipos de datos

Los tipos de datos de dBASE son cinco: caracter, numérico, lógico, "memo"[1] y fecha. El almacenamiento de estos tipos de datos se realiza en formato ASCII, siendo la descripción de cada uno de ellos la siguiente:

- **Caracter:** caracteres ASCII
- **Numérico:** - . 0 1 2 3 4 5 6 7 8 9
- **Lógico:** ? (indefinido) Y y N n T t F f
- **Fecha:** ocho dígitos en formato YYYYMMDD

[1]El campo "memo" es usado para hacer descripciones amplias de la tupla o registro correspondiente. El uso de este campo no es considerado en el sistema.

La descripción de las primitivas **val_ascii()** y **ascii_val()** se muestra a continuación.

```
/* tipos de datos de dBASE */
#define dBLOGICO     'L'
#define dBNUMERICO   'N'
#define dBCHAR       'C'
#define dBDATE       'D'

/*  val_ascii
 *
 *  dar el valor en ASCII (en una cadena de caracteres
 *  imprimibles) de acuerdo al tipo del dato
 *
 *  Entrada
 *
 *    pval : apuntador al valor fuente
 *    tipo : tipo del valor
 *    size : tamaño del valor
 *    valor: valor destino en ASCII
*/

void   val_ascii(pval, tipo, size, valor)
char   *pval ;
UCHAR  tipo ;
UCHAR  size ;
char   *valor ;
{
   switch (tipo) {
   case dBLOGICO:
   case dBNUMERICO:
   case dBCHAR:
      copyn(pval, valor, size) ;
      break ;
   case dBDATE:
      copyn(pval + 6, valor, 2) ;        /* día */
      *(valor + 2) = '/' ;
      copyn(pval + 4, valor + 3, 2) ;    /* mes */
      *(valor + 5) = '/' ;
      copyn(pval + 2, valor + 6, 2) ;    /* año */
      break ;
   }
}

/*  ascii_val
 *
 *  dada una cadena de caracteres, obtener el valor
 *  correspondiente al tipo de dato indicado
 *
 *  Entrada
 *
 *    pval : cadena de caracteres
```

```
 *     tipo : tipo del valor
 *     size : tamaño del valor
 *     valor: valor destino
*/

void    ascii_val(pval, tipo, size, valor)
char    *pval ;
UCHAR   tipo ;
UCHAR   size ;
char    *valor ;
{
   switch (tipo) {
   case dBLOGICO:
   case dBNUMERICO:
   case dBCHAR:
      copiar(pval, valor, size) ;
      break ;
   case dBDATE:
      copiar("19", valor, 2) ;
      copiar(pval + 6, valor + 2, 2) ;    /* año */
      copiar(pval + 3, valor + 4, 2) ;    /* mes */
      copiar(pval, valor + 6, 2) ;        /* dia */
      break ;
   }
}
```

En lo que corresponde al manejo de los valores nulos, éstos son representados como cadenas de espacios (sin importar el tipo de dato de que se trate).

Acceso a los datos

Las rutinas que se encargan de obtener, insertar, eliminar y modificar las tuplas de una relación son la primitivas buffreg(), altas(), bajas() y cambios(), respectivamente. La descripción de cada una de ellas es la siguiente.

```
/*  buffreg
 *
 *  llenar buff_reg[] con el registro indicado.
*/

void      buffreg(fd, no_reg)
FILEDESC fd ;
```

```
ulong     no_reg ;
{
   void   getbytes() ;
   ulong  goto_noreg() ;
   ulong  desplaz ;
   ushort longrec ;

   desplaz = goto_noreg(fd, no_reg) ;
   longrec = df[fd].ptr->long_rec ;
   getbytes(fd, desplaz, df[fd],buffreg, longrec) ;
}


/*  goto_noreg
 *
 *    posicionarse en el registro noreg del file fd
 *
 *  Entrada :
 *
 *    fd                        :descriptor del file
 *    noreg                     :número de registro deseado
 *
 *  Salida :
 *
 *    posi                :posición en el file (de tipo long)
*/

ulong     goto_noreg(fd, noreg)
FILEDESC fd ;
ulong     noreg ;
{
   ushort longenc ;
   ulong  posi ;

   longenc = longitud del encabezado ; /* byte 8 */
   posi = longenc + (df[fd].ptr->long_rec) * (noreg-1) ;
   return(posi) ;
}


/*  altas
 *
 *    añade un registro a un archivo
 *
 *  Entrada
 *
 *    fd : descriptor del archivo
 *    str: conjunto de valores
*/

void      altas(fd, str)
FILEDESC fd ;
char     *str ;
{
```

```
    /* longitud del registro */
    reclong = df[fd].ptr->long_rec ;

    /* pedir RAM para la tupla que se va a insertar */
    buff = (char *)malloc(reclong) ;

    /* preparar el registro (buff) de acuerdo a los valores
     * recibidos (str). Esta función hace uso de ascii_val()
     * ya que los valores que tenemos están en ASCII y
     * deberán ser convertidos al tipo que corresponda */
    prep_reg(fd, str, buff) ;

    /* inserta la tupla (buff) en el archivo */
    putbytes(fd, -1, buff, reclong) ;
}



/*  bajas
 *
 *    eliminación virtual de un registro (el registro es
 *    marcado)
 *
 *  Entrada
 *
 *    fd     : descriptor del archivo
 *    numreg: número de registro a eliminar
 */

void     bajas(fd, numreg)
FILEDESC fd ;
ulong    numreg ;
{
    /* posición del registro a modificar */
    desplaz = goto_noreg(fd, numreg)

    /* marca la tupla con un asterisco */
    putbytes(fd, desplaz, "*", 1) ;
}



/*  cambios
 *
 *    actualiza un registro de un archivo de acuerdo al
 *    conjunto de valores recibido.
 *
 *  Entrada
 *
 *    fd     : descriptor del archivo
 *    numreg: número de registro a actualizar
 *    str    : conjunto de valores
*/
```

```
void      cambios(fd, numreg, str)
FILEDESC fd ;
ulong     numreg ;
char     *str ;
{
    /* longitud del registro */
    reclong = df[fd].ptr->long_rec ;

    /* preparar el registro (df[fd].buff_reg) de acuerdo a
     * los valores recibidos (str). Esta función hace uso
     * de ascii_val() ya que los valores que tenemos están
     * en ASCII y deberán ser convertidos al tipo que
     * corresponda */
    prep_reg(fd, str, df[fd].buff_reg) ;

    /* posición del registro a modificar */
    desplaz = goto_noreg(fd, numreg)

    /* reescribe la tupla (buff_reg) en el archivo */
    putbytes(fd, desplaz, df[fd].buff_reg, reclong) ;
}
```

6

Conclusiones

Nosotros sabemos sòlo en tanto que hacemos.

-Novalis.

Toda aplicaciòn requiere de un anàlisis de la tarea o funciòn que se desee realizar. Lo presentado en los capìtulos anteriores es el anàlisis y desarrollo de un sistema cuyo objetivo principal es permitir que el usuario pueda observar y actualizar una base de datos de una manera sencilla.

Para lograr tal objetivo, se escogiò una alternativa media entre una herramienta poderosa como un lenguaje (o extensiones) para el manejo de los datos y otras herramientas cuya orientaciòn es geomètrica, esto es, su ùnico objetivo es el de elaborar plantillas (las cuales son incorporadas en un programa).

Para escoger tal alternativa, el factor determinante fue el usuario. De esta forma, el sistema pretende ser usado lo mismo por una persona experta en computaciòn, que por otra que carezca de conocimientos computacionales (por ejemplo, programar). Dedicar el sistema a esta gama de usuarios, enfatiza la importancia que tiene la interfaz del usuario en el desarrollo de herramientas. Una buena interfaz del

usuario requiere un anàlisis amplio de la funciòn a realizar, lo cual significa que no sòlo factores tècnicos forman parte del anàlisis. Los factores que tienen que ver con la interacciòn del usuario para con el sistema adquieren cada dia mayor importancia.

Apèndice A

Manual del Usuario

A continuación, se presenta una descripciòn del uso del sistema. En vista de que el sistema tiene dos componentes principales, se harà la descripciòn de cada uno ellos. Iniciamos con el editor y concluimos con el manejador de la pantalla.

Editor

Para el uso del editor se tienen dos casos: crear la vista y editar una vista. Si se desea crear la vista, habrà que especificar el nombre de la vista, la base de datos, y los archivos involucrados. Por otra parte, si lo que se desea es actualizar una vista, lo ònico que necesita el editor es el nombre de la vista a modificar. El modo de uso del editor es el siguiente:

edit [vista] [base de datos] [archivos ..]

Por lo que toca a la definiciòn de cada uno de los elementos que conforman la vista, primero se describe el factor geomètrico (pantalla) y posteriormente los factores lògicos.

Manejo de la pantalla

Inicialmente, el usuario tiene una sòla caja (ver figura A.1), cuya dimensiòn es el nùmero de lìneas de la pantalla

(24). Al iniciar, por "default", el manejo de la pantalla es fijo. Esta estrategia indica que los objetos que sean definidos en la vista deberán aparecer siempre en el mismo lugar. La otra opción que ofrece el sistema para el manejo de la pantalla, es definir una caja móvil, lo cual significa que los objetos irán apareciendo en la siguiente línea disponible, haciendo una especie de "enrrollamiento" por la parte superior de la caja (obsérvese la figura 3.1).

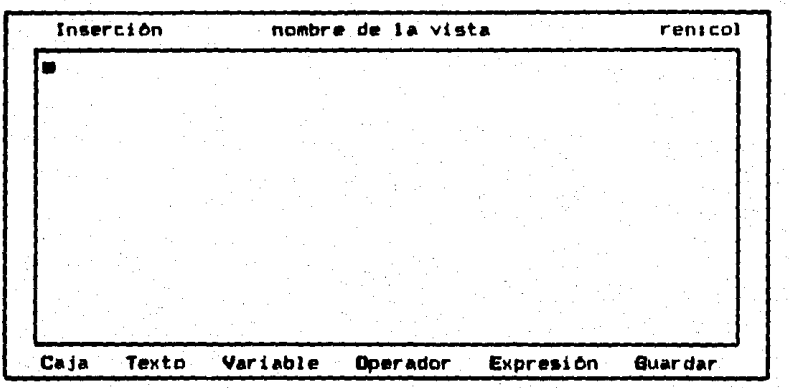


Figura A.1 Estado inicial del editor

Si se desea otra caja -además de la inicial-, ésta se obtiene digitando la tecla indicada para ello (C), con lo que inmediatamente se dibuja el límite inferior de la caja anterior y el límite superior de la caja nueva. Obviamente, el trazo de la caja se hace en la posición en que se

encuentra el cursor. Asimismo, las dimensiones de una caja podràn modificarse, pero siempre dentro de un marco de 24 lineas.

Archivos principal y secundarios

La especificaciòn de los archivos se hace solamente cuando se crea la vista y para tal efecto hay dos opciones: indicarlos en la linea de comandos o responder a las preguntas hechas por el sistema en su fase inicial. Una vez que el usuario ha señalado los archivos de datos a considerar en la vista, el sistema determinarà la relaciòn existente entre los mismos. En caso de no existir tal relaciòn, habrà una indicaciòn de terminaciòn.

Objetos

Texto

Cuando el usuario desea anexar un objeto de tipo texto, lo indica digitando una "T"; enseguida se solicita el texto. Una vez que se ha digitado, se le preguntarà por los atributos comunes.

Variable

Este tipo de objeto se declara digitando una "V", con lo que aparece una ventana que muestra los archivos que definen la vista y que permite seleccionar el archivo deseado; luego aparecerà el esquema del archivo seleccionado en otra ventana, en donde se podrà escoger -con las flechas del

teclado- el campo deseado. Una vez que se ha indicado el campo, se preguntará por los atributos comunes.

Expresión

La incorporación de este tipo de objeto en la vista se hace al digitar la "E"; posteriormente aparece el "prompt" que solicita la expresión, lo cual deberá hacerse en notación infija. Por último, se preguntará por los atributos comunes.

Operador

Para añadir este tipo de objeto, el usuario se sitúa en el lugar deseado y oprime la tecla "O"; a continuación se hacen dos preguntas: ¿Cuál operador se desea? y ¿sobre qué variable será calculado el operador? Una vez que se han respondido, se procede al establecimiento de los atributos comunes.

Condición

La condición es una expresión que será falsa o verdadera. El usuario la establece oprimiendo primeramente la tecla que permite incluir la expresión: (F7). Esta se introduce en la forma usual, esto es, en notación infija. Como se vio en el capítulo 3, por medio de la condición podrán observarse los registros (ocurrencias) que la satisfagan.

Cláusula de agrupamiento

El usuario especifica el campo por medio del cual desea agrupar los registros del archivo principal. Este elemento

es muy importante para el cálculo de los operadores. Como en el caso anterior, el usuario oprime primero la tecla indicada (**F8**) para establecer el agrupamiento, e inmediatamente después el sistema pregunta por el atributo -simple o compuesto- en que desea hacer la agrupación.

Por ultimo, para salir del editor existen dos opciones: abandonarlo sin grabar en disco la vista, o, antes de salir del editor, almacenar en disco la definición de la vista. Si al salir del editor se desea almacenar en disco la vista, se oprime la tecla "**S**", caso contrario, una "**N**".

Manejador de la pantalla (MP)

Para el uso del MP se tienen dos casos: modo de actualización (edición) y modo de consulta (reporte). Para trabajar con el MP habrá que especificar la vista y el modo de trabajo deseado, de aquí que el uso del MP sea el siguiente:

mp vista modo-uso

donde el modo de uso de edición se indica con la palabra edit o las letras **A** (altas), **B** (bajas) y/o **C** (cambios); mientras que el modo de uso de reporte se indica con la palabra repo o la letra **R**.

Modo de edición

Para trabajar en este modo, el usuario lo primero que deberá hacer es proporcionar los valores de las variables que pueden ser alteradas (variables de entrada-salida). El cursor se irá colocando únicamente en aquellos campos en los cuales podrán asignarse valores. Si la vista contiene variables de salida o expresiones, sus valores aparecerán conforme se vayan proporcionando los valores de las variables de las cuales dependen. Una vez que todos los valores de la vista han sido establecidos y mostrados, el usuario señala el tipo de movimiento que desea efectuar (A, B, C). Inmediatamente, el MP se encarga de ejecutar la operación deseada. Si en la ejecución de la operación se observa una anomalía, el MP indica al usuario (mediante un mensaje adecuado) el tipo de anomalía observada. En caso de que la ejecución de la operación sea exitosa, el MP presenta la plantilla (forma) vacía para continuar con la tarea de edición de la DB.

Modo de reporte

Este modo de operación es utilizado cuando el único objetivo es la observancia de los datos. De acuerdo a lo expuesto en el capítulo dos sobre **vistas** (2.7), podemos decir que las formas aquí empleadas se denominan formas (vistas) "read-only". Cuando el MP es invocado bajo este modo de operación, inmediatamente empieza a mostrar todos los datos que deban

de aparecer de acuerdo con la cláusula de condición, así como también en el orden correcto de acuerdo con la cláusula de agrupación. En este último caso, se calculan adecuadamente los operadores que hayan sido definidos en la vista. Cuando un grupo de datos termina y otro está por empezar, se le indica al usuario, de manera que éste pueda tener una mejor apreciación de los datos. Por último, si la forma tiene una caja móvil, cada nuevo dato (tuplo) aparece por la parte inferior; cuando la caja está llena, al aparecer un nuevo dato, desaparece el dato que se encuentra en la primera línea de la caja.

Bibliografía

[Bass 1985] Leonard J. Bass, "A Generalized User Interface for Applications Programs", Communications of the ACM, Volumen 28, Número 6, (junio 1985), páginas 617-627.

[Bell 1987] Doug Bell, Ian Morrey y John Pugh, Software Engineering: A Programming Approach, Primera edición, Prentice-Hall International, (1987).

[Campbell 1987] Joe Campbell, Crafting C Tools for the IBM PCs, Primera edición, Prentice-Hall, (1987).

[Cosmadakis y Papadimitriou 1984] Stavros S. Cosmadakis y Christos H. Papadimitriou, "Updates of Relational Views", Journal of the ACM, Volumen 31, Número 4, (Octubre 1984), páginas 742-760.

[Date 1986] C.J.Date, An Introduction to Database Systems, Volumen 1, Cuarta edición, Addison-Wesley, Reading, Massachusetts, (1986).

[Dayal y Bernstein 1982] U. Dayal y Philip A. Bernstein, "On the Correct Translation of Update Operations on Relational Views", ACM Transactions on Database Systems, Volumen 8, Número 3, (septiembre 1982), páginas 381-416.

[Fisher 1988] Alan S. Fisher, CASE: Using Software Development Tools, Primera edición, John Wiley & Sons, (1988).

[Kernighan y Plauger 1981] B.W. Kernighan y P.J. Plauger, Software tools in Pascal, Addison-Wesley, (1981).

[Korth y Silberschatz 1986] Henry F. Korth y Abraham Silberschatz, Database System Concepts, Primera edición, McGraw-Hill Book Company, (1986).

[Rettig 1985] Luis Castro, Jay Hansen y Tom Rettig, advanced programmer's GUIDE, Ashton-Tate, (1985).

[Stevens 1987] Al Stevens, C Database Development, Primera edición, Management Information Source, Inc., (1987)

[Tremblay y Sorenson 1985] Jean-Paul Tremblay y Paul G. Sorenson, The Theory and Practice of Compiler Writing, Primera edición, McGraw-Hill, Inc., (1985)

[Ullman 1982] J.D.Ullman, Principles of Database Systems, Segunda edición, Computer Science Press, Rockville, Maryland, (1982).