UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
"A R A G O N"

"DISEÑO DE LA BASE DE DATOS PARA LA INTERFAZ HOMBRE - MAQUINA DEL PROYECTO SISTEMA DE CONTROL DISTRIBUIDO PARA LA CCC GOMEZ - PALACIO, DURANGO"

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

DANTE APULEYO SALAZAR ARENAS

ENEP ARAGON

San Juan de Aragón, Edo. de México 1993

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

Indice de Figuras

Capítulo 1

Capítulo 2

Capítulo 3

INTRODUCCION

La presente década se anuncia para nuestro pais como una época de apertura no solo en el aspecto económico a los mercados internacionales sino además, en los renglones cultural y científico; bajo este marco, la ingeniería en México debe responder ofreciendo tecnología con un alto nivel de competitividad tanto en proyectos nacionales como en el extranjero. Al mismo tiempo, los profesionistas del área de la computación afrontan un momento singularmente difícil en la historia del software caracterizado por problemas como costos de desarrollo elevados, código no reutilizable, documentación inadecuada, incumplimiento de fechas de entrega, etc.

Generalidades

Este trabajo de tesis documenta el proceso de diseño y las características de la base de datos para la IHM (*Interfaz Hombre-Máquina*) del proyecto a cargo del IIE (*Instituto de Investigaciones Eléctricas*) para el desarrollo del sistema de control distribuido para la central de ciclo combinado de Gómez Palacio, Durango. El punto más relevante del diseño de esta base es que en respuesta a la crisis del software mencionada en el párrafo anterior, la especificación de la misma fué hecha a través de un *lenguaje-esquema* que posteriormente es *traducido* de manera automática en el código necesario para establecer el ambiente donde los datos se almacenarán. Aunque el desarrollo de la herramienta traductora de *lenguaje-esquema* se llevó acabo al mismo tiempo que la recopilación y análisis de los requerimientos de la base, su enfoque es global, pudiendo aplicarse en otros proyectos donde la información deba manipularse en tiempo real.

En las dos últimas décadas los temas sobre bases de datos han pasado a formar una verdadera disciplina académica de la ingeniería en computación, muestra de ello es la abundante literatura al respecto donde se exponen técnicas y experiencias en el análisis, diseño e implementación y la aparición de herramientas que simplifican y formalizan las diferentes etapas del desarrollo de una base de datos, siendo notable el desplazamiento del peso que tradicionalmente tenía la fase de codificación considerada como la más importante, larga y costosa de un proyecto a fases previas y posteriores como lo son el análisis, diseño, prueba y mantenimiento. Cabe señalar que estos cambios aunque difíciles de aceptar en ambientes de trabajo donde la costumbre es plasmar las ideas directamente sobre el código, han mostrado ser arma eficaz contra la crisis del software, aumentando la eficiencia horas/hombre durante el desarrollo del producto original y más aun durante la etapa de mantenimiento.

La idea general del texto sigue el enfoque mencionado, presenta un robusto marco teórico general y un profundo análisis del problema en cuestión que culminan en la especificación de la base de datos; la codificación de la misma queda simplemente como un procedimiento de traducción de esquema a código. La lectura del texto, tratado con estas características, es adecuada tanto para la consulta del investigador directamente relacionado con el proyecto como para todo aquel que se enfrenta a un problema de diseño de base de datos, especialmente cuando se trata de aplicaciones en tiempo real.

Antecedentes

Como antecedentes de este proyecto, el IIE ya había desarrollado otro SCD (*Sistema de Control Distribuido*) para la Central Termoeléctrica de Dos Bocas, Veracruz pero su base de datos estaba implantada directamente en código y la documentación de la misma que estaba en proceso de elaboración no había crecido a la par del diseño; sin embargo, como los servicios que el SCD debía prestar en la CCC (*Central de Ciclo Combinado*) de Gómez Palacio y las características operativas de ambas plantas son semejantes, se tomó como punto de partida la base de datos para la IHM de Dos Bocas, trabajando en la redefinición, eliminación e incorporación de nuevas entidades, atributos, índices y relaciones.

Objetivos

Dentro de los objetivos generales propuestos para el diseño de esta base se había contemplado inicialmente: amplia y continua documentación, diagramación mediante hipergráficas y codificación en lenguaje C con control de concurrencia para ambiente VAXELN. Posteriormente, reconociendo por un lado la fuerte posibilidad de obtener el desarrollo de otros proyectos de SCD similares y atendiendo a que la transferencia de tecnología desde el punto de vista del hardware (la línea SAC) ya es un hecho, se decidió iniciar la consolidación del software como un producto; con esta idea en mente, se extendió el alcance del diseño de la base, haciendo obligatoria la creación de la herramienta traductora de esquemas, la biblioteca de rutinas para el manejo de la base de datos y la codificación en C con alto grado de portabilidad hacia diferentes ambientes.

Contenido

El presente trabajo está dividido en tres partes, cada una correspondiendo a un capítulo. En el primero de ellos que está subdividido en cinco secciones, damos el marco teórico que rodea el tema de bases de datos sin considerar específicamente los aspectos del problema real; empezamos tratando aspectos generales de sistemas y conceptos relacionados con bases de datos incluyendo una breve semblanza sobre la evolución de los sistemas de BD (Base de Datos). En la segunda sección, considerando la problemática que implica un diseño integral desde hardware hasta software hablaremos sobre los medios de almacenamiento exponiendo la historia, características y tendencias de los mismos. Subiendo un nivel en cuanto almacenamiento de información, tocaremos el tema de archivos de datos mencionando tipos, características y estructuras de datos en general. La cuarta sección se ocupa del modelado, ahí aplicaremos la diagramación por hipergráficas, proponiendo una simbología especial. Para finalizar este primer capítulo veremos

como mapear el modelo conceptual a cada uno de los tres enfoques más conocidos: jerárquico, de red y relacional.

En el segundo capítulo se aborda el planteamiento y análisis del problema; en la primera sección, que es en gran medida resultado del contacto directo con la situación en planta, consideraremos brevemente el proceso de generación y daremos el diagrama del SCD, se mencionarán los tipos de variables que entran en juego y los algoritmos que se les debe aplicar. Pasaremos después a la proposición y descripción detallada de entidades y atributos, incluyendo para finalizar el capítulo algunas estadísticas cuantitativas sobre entidades con lo cual, habremos reunido todos los elementos necesarios para el diseño de la base.

En el tercer capítulo se conjugan los dos anteriores; empezaremos proponiendo una arquitectura general para la BD sin tratar aspectos teóricos sobre diseño, luego tomando los resultados del análisis de requerimientos que desembocaron en atributos y entidades elaboraremos y normalizaremos su modelo conceptual. Entrando a la parte medular del capítulo describiremos el procedimiento a seguir para hacer un buen uso de la herramienta traductora de esquemas, estudiaremos también las estructuras descriptoras de la base y como declararlas a través del *lenguaje-esquema* por medio de la sintaxis correcta, ejemplificándolo con el propio esquema de la base para la IHM de la central de Gómez Palacio. En la segunda sección del capítulo expondremos el propósito y modo de uso de las rutinas compatibles con los descriptores para el manejo de la BD, ordenándolas de acuerdo a su objetivo y proporcionando varios ejemplos. Ya para finalizar se hará un análisis evaluativo sobre el consumo real de memoria y el tiempo de respuesta de las operaciones a efectuar sobre la BD.

Los apéndices contienen algunos listados; en el apéndice A encontraremos el esquema de la BD, archivo de texto que es leído por la herramienta traductora la cual a su vez, produce otros tres archivos con código fuente en lenguaje C; el listado de uno de ellos, el archivo **BD_DESC.H** en el que se declaran las estructuras descriptoras de la BD se reprodujo en el apéndice B debido a su importancia. En el apéndice C listamos el archivo **BD_ERR.H** que contiene directivas del tipo '#define' para sustituir durante la compilación los nombres de los errores que las rutinas del manejador de la base detectan por valores enteros.

Justificación

El hecho de aplicar una técnica de cuarta generación (T4G) para la codificación de la base de datos no obligó a seguir un planteamiento teórico ni de análisis de requerimientos especial, es decir que apoyándonos en la exposición hecha en los dos primeros capítulos podemos seguir el camino de codificación convencional o alimentar al traductor de esquemas y emplear la biblioteca de funciones para el manejo de la base de datos.

Elaborar una herramienta traductora de esquemas no fue desición sencilla, se discutieron los problemas que podría llevar consigo: en primer lugar, la aplicación de T4G es relativamente reciente y limitada a sistemas de información comerciales siendo innovador su uso en productos de ingeniería para control de procesos al menos en nuestro pais además, el usuario del producto final (CFE) no había proporcionado la especificación de requerimientos completa y definitiva por

lo que eran de esperarse cambios y nuevas peticiones (lo cual de hecho sucedió), también consideramos las críticas hechas a T4G por que implican un mayor esfuerzo en el planteamiento de la estrategia de diseño, el código fuente que generan es con frecuencia ineficiente y por si fuera poco, obligan al usuario a "aprender" otro lenguaje para interaccionar con la herramienta y conseguir el objetivo deseado.

No obstante, dadas las dimensiones y alcance del proyecto el costo de los inconvenientes era ampliamente superado por los resultados esperados lo cual se manifestó por la facilidad con que es posible modificar la estructura de la base, sin perder consistencia, mediante la simple modificación del esquema y regeneración del código. Además las características del código fuente producido garantizan un buen nivel de velocidad de respuesta, indispensable en cualquier aplicación de tiempo real ya que mantienen la información en memoria principal y el acceso a la misma puede ser directo o, aleatorio a través de índices de estructura predefinida y optimizada o bien mediante algoritmos construidos por el programador.

Referencias

En la elaboración de este trabajo fueron consultadas varias fuentes bibliográficas tanto de tonalidad teórica como práctica, asi mismo se consideraron las ideas expuestas en reuniones y entrevistas especialmente en la parte de análisis de requerimientos. La bibliografía ha sido manejada por capítulo y cuando existe algún punto de singular interés se hace referencia mediante notas al pie de página a otras partes del texto o a otras obras donde pueda ampliarse la información.

CAPITULO 1

ASPECTOS GENERALES SOBRE BASES DE DATOS

1.1. DEFINICION Y ALCANCES DE LOS SISTEMAS DE BASE DE DATOS

1.1.1. Sistemas en general

Ya de tiempo atrás han existido sistemas de información, pero la necesidad de controlar y procesar datos en cantidades cada vez mayores en diferentes áreas ha originado la derivación de una rama de la ingeniería encargada específicamente del diseño, análisis e implantación de sistemas. Para exponer objetivamente el diseño de una base de datos revisaremos brevemente algunos conceptos relacionados con sistemas.

Podemos decir que un sistema, en general, *es un conjunto de procedimientos interrelacionados cuya agrupación permite llegar a un objetivo específico*[1]. Estos *procedimientos* detallan paso a paso, que, quien, cuando y como se deben hacer las cosas. Los elementos conformadores de procedimientos los llamamos *componentes* que si varían poco o nada son *parámetros* del sistema, por el contrario cuando están sujetos a cambio se les llama *variables*.

El enfoque de sistemas aplicado a la ingeniería y en general a cualquier actividad compleja proporciona dos herramientas muy poderosas, *modularidad* y *abstracción*. Se dice que hay modularidad cuando el sistema es dividido en subsistemas los cuales a su vez pueden dividirse hasta llegar a módulos compuestos por procedimientos únicamente. Por otro lado, con la

[1] Fitzgerald, J., *Fundamentos de Análisis de Sistemas*, p23

abstracción elegimos el nivel de detalle deseado. Podemos distinguir una clasificación de los sistemas en cuanto a su forma de operación:

Sistemas Abiertos: Aquellos que no tienen "realimentación", es decir, no se autocontrolan.

Sistemas Cerrados: Los que automáticamente se controlan, modificando su funcionamiento a partir de datos generados por ellos mismos.

Antes de iniciar el diseño de cualquier sistema es indispensable:

1) Definir claramente el problema en cuestión.

2) Describir detalladamente el sistema existente.

3) Determinar los requerimientos del nuevo sistema.

El proceso de diseño inicia con la elaboración de un modelo conceptual que en principio solo existe en el pensamiento y es descrito gráfica o textualmente, a este modelo lo llamamos *sistema conceptual* pues contiene datos relacionados formando una idea o concepto. La implantación y puesta en marcha de tal sistema lo hace evolucionar a la forma de *sistema empírico*.

En particular un sistema de cómputo debe:

1) Proporcionar a las personas indicadas la información adecuada a un costo razonable.

2) Mejorar la disponibilidad y veracidad de la información comparada con la obtenida mediante métodos convencionales.

3) Dejar abierta la posibilidad de manejar volúmenes de información mayores.

4) Facilidad de adaptación a cambios en el tipo de los datos.

5) En general, incrementar productividad y reducir costos.

Los sistemas de cómputo por su flexibilidad[2] pueden dividirse en dos clases, aunque en la práctica presenten características de ambas:

1) *Sistemas de información:* Proporcionan herramientas para construir algoritmos de procesamiento de datos a partir de procedimientos básicos. Esa flexibilidad los hace adecuados para responder a actividades de tipo ejecutivo.

---

[2]Martin, James, *Principles of Data Base Management*, p65

2) *Sistemas de operaciones:* Las prestaciones que ofrecen ya están definidas y programadas; sobre los anteriores tienen la ventaja de responder rápidamente, por tanto, se aplican en sistemas supervisorios.

Por otro lado, un sistema de tiempo real (algunos autores[3] incluyen dentro de esta categoría sistemas como los de terminales de usuario), lo consideraremos como aquel conectado a sensores; termopares, rastreadores ópticos, aparatos de medida y transductores en general, los cuales generan un flujo de datos no estructurados semejante al de los organismos vivientes; estos datos se procesan y producen salidas que alteran características del medio como temperatura, posición de válvulas, etc. Con esto queda cerrado un lazo de realimentación comparable con la coordinación entre la mano y el ojo.

Otra característica frecuentemente encontrada en los sistemas de tiempo real es que demandan tiempos de respuesta menores, atendiendo en un instante dado a más de una entrada; la concurrencia suele ser un requisito fundamental. De ahi que un sistema de cómputo para control en tiempo real deba diseñarse metódicamente.

1.1.2. Sistemas operativos

El usuario e incluso el programador se ven liberados de codificaciones largas y tediosas gracias a la existencia de herramientas que proporcionan un acceso básico por medio de un administrador de archivos presente en los sistemas operativos, el cual, se encarga de transformar direcciones lógicas solicitadas en físicas. De igual forma lenguajes de alto nivel proveen rutinas de Entrada/Salida para garantizar transportabilidad del código gracias a su independencia respecto al hardware.

Los sistemas de base de datos se han establecido en todo tipo de sistemas operativos, para lograr resultados óptimos, es importante conocer los servicios y filosofía de funcionamiento del ambiente de desarrollo. Por ello mencionamos a grandes razgos las características de algunos sistemas operativos y en particular las del *VMS* y del *VAXELN*, que fueron los ambientes de trabajo para el diseño y pruebas de la base de datos para la interfaz hombre-máquina de la central de ciclo combinado de Gómez Palacio.

En la Figura 1.1 encontramos gráficas de recursos vs. tiempo de las filosofías de operación de sistemas mas comunes.

El **procesamiento por lote**, metodología actualmente restringida a contadas aplicaciones asigna todos los recursos del computador a cada programa, corriendo uno a uno; esto se refleja en un promedio de recursos aprovechados muy bajo.

Por medio de **multiprogramación** los recursos se utilizan mas razonablemente, en este ambiente si una tarea libera un recurso requerido por otra en estado de espera, esta última

---

[3]Martin, James, *Principles of Data Base Management*, p27

empezará a correr. Asi los recursos del sistema en conjunto son mejor aprovechados aunque particularmente cada tarea demore un poco más en realizarse.

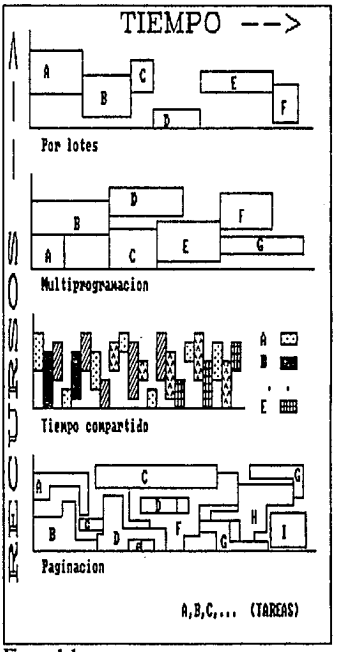


Figura 1.1
Recursos vs Tiempo

En **tiempo compartido** las tareas son seccionadas, repartiéndose el tiempo del procesador. Este método resulta eficaz si las tareas son cortas, siendo punto crítico la administración de la memoria principal especialmente al atender a varios usuarios. La velocidad de respuesta depende de varios factores tales como: tiempo asignado a cada procedimiento, prioridad, tiempo para cambiar la sección activa y número de usuarios en el sistema.

La **paginación**, también llamada *sistema de almacenamiento virtual* es una técnica todavía mas eficaz que la anterior; se basa en divisiones posteriores de la memoria en *páginas* que son espacios de memoria de unos cuantos miles de bytes cuyo propósito es mejorar la compartición de recursos.

VMS

El VMS (Virtual Memory System) ofrece paginación; todas las referencias de programas generadas por el CPU son direcciones virtuales que deben ser traducidas a direcciones físicas; esta filosofía permite ejecutar aplicaciones que ocupan mas espacio de memoria que el disponible en un momento dado. El mecanismo es el siguiente: ante una petición de lectura a disco el microcódigo de la VAX busca en la tabla de páginas cargadas en memoria principal si existe alguna asociada con la dirección del dato solicitado, en caso afirmativo reporta los dígitos de orden superior asociados con la página o de lo contrario, genera una interrupción de página faltante remitiendo al *paginador* la obligación de disponer espacio para una página nueva.

VMS proporciona otra herramienta que salva la limitante en cuanto a espacio en memoria principal conocido como "swapper" (intercambiador). Si aun después que el paginador limita el número de páginas disponibles en memoria para cada proceso (aumentando asi la cantidad posible de procesos activos) no hubiera capacidad en memoria para iniciar otra computación prioritaria, el "swapper" mueve aquellas con menor prioridad a disco hasta disponer del espacio necesario. El diseño de un sistema de base de datos debe tomar en cuenta tamaño y fronteras de las páginas con el proposito de alinear los bloques de acuerdo a esos parámetros, a modo que solo una página se cargue por cada solicitud de bloque (en la sección 1.3 se describen el manejo de bloques y la paginación con más detalle).

Además de proporcionar herramientas para manejar archivos dando independencia al programa respecto al hardware empleado, los sistemas operativos para multiusuario como el VMS, tienen un manejador cronológico de colas para los procesos que solicitan acceso a algún recurso del sistema, la atención a tales solicitudes es controlada mediante determinadas *políticas de acceso* que comúnmente se pueden modificar.

Las políticas de acceso a disco deben ajustarse en relación a los requerimientos de lectura de la base seleccionando aquella que se refleje en un tiempo de retraso mínimo, para lo cual el diseñador aprovecha la capacidad de transferencia del dispositivo, inclinándose en muchos casos por una política de *menor tiempo de servicio primero (SSTF)*; estas políticas se analizarán al detalle en la sección 1.2.

VAXELN

El objetivo principal del VAXELN es servir de soporte a aplicaciones en tiempo real. Con los sistemas operativos comunes comparte la característica de tener un programa (el *Kernel*) que coordina los demás y administra los recursos de la computadora sin embargo, no es objetivo de VAXELN facilitar la interacción entre la computadora y el(los) usuario(s) sino entre esta y una aplicación.

El proceso de desarrollo de una aplicación implica el uso de un nodo en el que corra VMS para edición, compilación, ligado con bibliotecas de VAXELN y construcción de un "sistema", el cual es descargado via red Ethernet hacia otro nodo donde ejecutamos el comando BOOT.

El VAXELN es especial para aplicaciones en tiempo real porque permite diferentes niveles de concurrencia: multitarea, multiprogramación y multiproceso. La aplicación se puede dividir en programas (jobs) y estos a su vez en procesos; los jobs tienen niveles de prioridad de 0 a 31 y los procesos de 0 a 15; los procesos en todo momento se encuentran en uno de los siguientes estados:

RUN
El proceso tiene control del procesador y esta corriendo actualmente.

READY
El proceso no esta corriendo actualmente pero está listo para ejecutarse tan pronto como el "scheduler" se lo permita.

WAIT
El proceso espera a que ciertas condiciones se cumplan como un lapso de tiempo, uno o más eventos o un mensaje de otro proceso.

SUSPEND
El proceso no se está ejecutando y para sacarlo de ese estado es necesario llamar explícitamente al procedimiento RESUME con lo cual podrá pasar a READY.

Ahora bien, las condiciones para pasar de READY a RUN y de ahi a WAIT son: Un proceso pasa de READY (que es el estado inicial de todos los procesos) a RUN cuando su nivel de prioridad es el más alto entre todos los que estan en READY. Un proceso en el estado RUN pasa a WAIT cuando en su algoritmo se hace una llamada a los procedimientos WAIT_ANY o WAIT_ALL.

Para coordinar la ejecución de los procesos y establecer medios de comunicación entre ellos VAXELN pone a disposición del programador diferentes objetos: Semáforos, Puertos, Mensajes, Eventos, etc.

1.1.3. Conceptos relacionados con sistemas de BD

Los componentes principales de un sistema de base de datos (*DBMS*, "Data Base Management System") son: hardware (el medio de almacenamiento), software (el DBMS y los programas desarrollados) y la información en si[4] contenida en una *base de datos*, la cual siendo el corazón del sistema, conviene distinguirla claramente.:

[4]Tsai, Alice,*Sistemas de Base de Datos*, p5

Definición de base de datos

Antes de la aparición del término *base de datos* a principios de 1970[5], se hablaba únicamente de archivos y conjuntos de datos, ahora el concepto de base de datos es tan polifacético que podemos definirlo desde varios puntos de vista. A nivel macro como herramienta de la informática, se le puede considerar parte integrante del proceso administrativo o de control de una empresa, mientras que a nivel micro, diríamos que se trata de un conjunto de bits almacenados en determinado hardware mediante la tecnología computacional. En un punto intermedio, una base de datos se implanta mediante archivos, registros y relaciones entre ellos representando el conjunto de datos estructurados que describen un sistema real.

En el ámbito de sistemas computarizados prácticamente todos cuentan con un almacén de información ya sea en medios primarios o secundarios, almacén que *estructurado* adecuadamante puede llegar a la categoría de base de datos. Es en esta área donde el concepto de base de datos hace referencia a ciertas unidades de información mutuamente relacionadas y organizadas lógicamente con el fin de facilitar su manipulación, dichas unidades o datos pueden ser información sobre una empresa, institución, proceso, o en general cualquier problema específico. De modo que diseño y mantenimiento de bases de datos son de vital importancia para el procesamiento de información.

Hablando prácticamente, base de datos es una colección de archivos integrados, coordinados y relacionados entre sí;[6] resultado de combinar información de diferentes fuentes vinculadas con aquello que deseamos modelar, controlar o monitorear mediante un sistema de cómputo.

Tratándose de bases de datos muy grandes deben añadirse como características deseables el acceso rápido, robustez en la estructura de datos, seguridad en la actualización y economía de almacenamiento.

Algunos autores[7] incluyen dentro de la definición de base de datos al mismo hardware del computador en el que son almacenados los datos e incluso, los programas que sirven para procesar la información. Reconsiderando esa posición, en cuanto al hardware observamos que cuando la base se debe diseñar para un equipo de cómputo específico, mas que formar parte de la base en si, estaremos ante una limitante para el desarrollo de la misma que en todo caso podría considerarse como parte del sistema de base de datos en conjunto; por otro lado si el equipo será adquirido en función de los requerimientos de diseño entonces la decisión de compra resultará de un análisis de las necesidades. En lo tocante a los programas utilizados para manipular la base, formarán mas bien parte del administrador de la base y no de esta.

---

[5]Martin, James, *Principles of Data Base Management*, p49

[6]Wiederhold, Gio, *Diseño de Bases de Datos*, p412

[7]Cfr. Wiederhold, op. cit., p1

Funciones de los sistemas de base de datos

Se considera que un DBMS debe integrar una colección de archivos que conforman la base de datos al ofrecer varias rutinas para realizar las tareas o *funciones* mas comunes sobre bases de datos, que son:

1) Proporcionar herramientas para abrir archivos asi como para crear y modificar la estructura de datos.

2) Mapear automáticamente referencias lógicas a direcciones de hardware, estableciendo independencia respecto a este punto.

3) Garantizar integridad y seguridad de la información.

4) Llevar un registro o historia de las operaciones realizadas sobre los archivos.

De acuerdo a lo anterior, en la Figura 1.2 mostramos como el DBMS sirve de enlace entre las peticiones o programas del usuario y el acceso físico a la base.

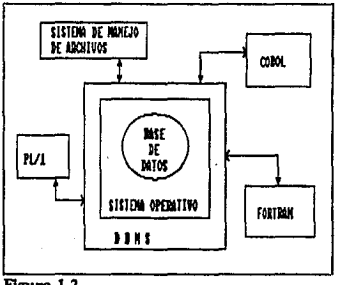


Figura 1.2
Ubicación del DBMS respecto a otros elementos del software

Caracterfsticas de los sistemas de base de datos

Podemos hablar en general de las *caracterfsticas* de los DBMS usados en sistemas de base de datos ya sea que hayan sido desarrollados para mainframe o ambiente de computadoras personales pues estos últimos resultan de transportar algunos conceptos aplicados en los primeros.

Estructura de datos: Las estructuras de base de datos mas socorridas son jerárquica, de red y relacional, (la relacional se ha empleado en ambientes personales principalmente).

Esquemas: Representan el escenario global de la base de datos y, dependiendo de su complejidad, o si ha de ser accesada por varios usuarios puede ser necesario recurrir a subesquemas

Lenguaje de consulta: ("Query Language") Es un lenguaje del tipo intérprete de alto nivel con el cual los usuarios manipulan generalmente de modo iterativo la base de datos, algunos DBMS no cuentan con esta característica especialmente si corren en ambientes mainframe.

Independencia de datos: Aspecto esencial que marca la diferencia entre un DBMS y un FMS (*File Management System* - Sistema Administrador de Archivos), su objetivo es conseguir que ante reestructuraciones en la definición de los datos dentro de archivos (por ejemplo, la adición de un campo) no sea necesario reescribir parte del código con que se manipula la información y al contrario, que los cambios al código puedan adaptarse fácilmente a la estructura de datos[8].

Compartición de datos: En ambientes de red la información debe ser accesada por diferentes usuarios que solicitan diversos enfoques, aplicar este concepto reduce duplicidad de información al mínimo.

Estructura de archivos: En la sección 1.3 estudiaremos al detalle las organizaciones de archivos pero por el momento, es importante remarcar que un DBMS implementa las bases de datos recurriendo a diferentes organizaciones de archivos al mismo tiempo, en *organizaciones de alto nivel*, permitiendo asi consultas sobre claves múltiples.

Seguridad: En ambientes multiusuario es común que el acceso o modificación de cierta información este restringida para algunos usuarios, ejemplo típico son los archivos de nómina.

Integridad: Siempre existe de manera potencial la posibilidad de daño a la estructura o a la información misma, especialmente debido a la compartición de datos por varias personas ya que el peligro de interferencias por transacciones activadas que afectan a una misma sección de la base aumenta. Los DBMS completos ofrecen procedimientos que además de garantizar la integridad recuperan en cierta medida la base cuando ha sido dañada. Las averías en bases de datos deben tomarse más seriamente que las alteraciones a un sistema convencional de archivos, pues en el primer caso es de esperarse reacciones en múltiples lugares.

Granularidad: La granularidad[9] de un manejador de base de datos se manifiesta por su capacidad de hacer referencia directa a elementos o campos de un registro. A mayor finura en esta característica, más eficiente el funcionamiento.

---

[8]James, Martin, *Principles of Data Base Management*, p45

[9]Campderrich B., *Técnicas de Bases de Datos*, p27

Los sistemas de base de datos como todo sistema tienen un ciclo de vida:

1) Análisis de los requerimientos.

2) Diseño de la base de datos.

3) Construcción de la base (posible transformación de la base anterior si ya existía una).

4) Transformación o creación de las aplicaciones para manipular la base.

5) Integración de aplicaciones, evaluación del sistema.

6) Crecimiento, mantenimiento.

En estos puntos no consideramos la muerte del sistema lo cual no quiere decir que un sistema de información deba ser permanente sino que aprovechando los elementos presentes evoluciona, reestructurándose a partir del primer paso cuando surgen cambios de importancia; de este modo no hay desperdicio del trabajo hecho.

1.1.4. Evolución de los sistemas de BD

El manejo de grandes cantidades de datos por medio de computadora ha sido un objetivo por alcanzar desde los primeros equipos fabricados, ejemplo de esto son las máquinas tabuladoras construidas por Hollerit usadas para reunir y procesar la información correspondiente al censo de 1900 en Estados Unidos.

Inicialmente debido a las características de los medios de almacenamiento el procesamiento secuencial era muy usado hasta que a mediados de los años sesenta, con el surgimiento de discos magnéticos con capacidad de acceso aleatorio, los manejadores de base de datos fueron adecuados mejorando sus prestaciones al basarse en estructuras de archivos mas complejas.

Todavía a finales de los años sesenta, el manejo de datos se hacía por medio de codificaciones en algún lenguaje de alto nivel pero el crecimiento no solamente en cantidad de datos sino en requerimientos de los usuarios hizo notorias varias deficiencias.

Con el desarrollo de sistemas de manejo de archivos se intentaron resolver las siguientes:

1) Aunque hay gran distancia entre lenguaje ensamblador y los lenguajes de alto nivel, la codificación de rutinas que conceptualmente son sencillas significaba trabajo de semanas e incluso de meses con la intervención de varios programadores, reflejándose esta circunstancia en costos y tiempos de proyecto.

2) Poca flexibilidad ante cambios: Resultaba difícil hacerlos y frecuentemente solo el programador del código original era capaz de efectuarlos.

3) Respuesta lenta a consultas originada por dificultad en el traslado del modelo conceptual de la base de datos a modelo físico, esto debido a carencia de herramientas poderosas dentro del mismo lenguaje.

A pesar de existir lenguajes de alto nivel como COBOL, FORTRAN, PL/1, etc., la creciente necesidad de desarrollo de aplicaciones en las que el punto central son bases de datos hizo indispensable diseñar paquetes con herramientas para el manejo de archivos (*FMS: Sistemas de Manejo de Archivos*), estos sistemas eran muy primitivos pues solo permitían el acceso a un archivo a la vez y a partir de una clave primaria.

Entre otros, algunos de los problemas[10] encontrados al usar sistemas de manejo de archivos y que se intentaron resolver con los DBMS fueron:

1) *Redundancia.* Cada módulo que conforma una aplicación normalmente requiere de diferentes conjuntos de campos, con un FMS la solución es repetir estos campos en tantos archivos como sea necesario desaprovechándo espacio de almacenamiento y poniendo en riesgo la consistencia de la información.

2) *Integridad de la información.* Debido a la redundancia, actualizar todos los archivos involucrados resultaba difícil y muchas veces surgían incongruencias. Otro factor en la falta de integridad no relacionado con el manejo de archivos se presentaba por seguridad insuficiente en los datos almacenados y por una verificación deficiente de la vigencia de estos al realizar cambios.

3) *Compartición de datos.* Como ya se mencionó los sistemas de manejo de archivos no permiten el acceso simultáneo a varios archivos siendo por tanto problemático accesar a información de dos o mas de ellos.

4) *Rapidez.* Los accesos a sistemas de archivos no interrelacionados son muy lentos, a tal grado que búsquedas manuales pueden ser más efectivas.

Es importante establecer la diferencia entre un sistema de base de datos usado para administrar un conjunto de archivos conformando una base de datos y un sistema de manejo de archivos; con los FMS se lleva a cabo recuperación de registros en función de una llave en tiempos de ejecución menores que los ofrecidos por un lenguaje de alto nivel convencional ya que los algoritmos de clasificación, mezcla y generación de reportes son optimizados por el FMS, pero estas utilerías presentan el inconveniente de solo poder abrir un archivo a la vez y accesarlo únicamente mediante los métodos básicos, sin presencia de ligamentos físicos o lógicos con otros archivos. Aunque un archivo controlado por FMS puede tener diversos usos al igual que en una base de datos, la diferencia a nivel conceptual entre el sistema de manejo de archivos y el de base de datos radica en que el primero solo podrá satisfacer un enfoque.

---

[10]Tsai, Alice, *Sistemas de Bases de Datos*, p198

A falta de relaciones entre archivos, los FMS presentan diversos problemas: cuando deseamos accesar campos de diferentes archivos debemos crear uno nuevo llegando a ser común tener un archivo por programa, por ello, la coordinación general de recursos tanto de hardware como de software se descuida, el desarrollo de nuevas aplicaciones pierde agilidad y la probabilidad de errores por inconsistencia en la información crece paralelamente con la redundancia de datos.

Con la aparición de bases de datos y la consecuente centralización de la información, la necesidad de compartir datos por varios usuarios localizados en diferentes lugares geográficos, deterioró el sentido de propiedad de la información perdiéndose al mismo tiempo la responsabilidad sobre su validez. Un diseño adecuado para sistemas con áreas restringidas contempla dentro de la misma estructura de la base las codificaciones y autorizaciones adecuadas a cada sector o clase de usuario.

Las bases de datos distribuidas surgen como opción interesante, dan al usuario transparencia en el acceso y sensación de propiedad permitiéndole en un momento dado entrada al sistema principal. Se clasifican en tres tipos:

1) *Distribuida particionada.* Es el tipo de distribución más conocido, la base se fracciona en porciones ubicadas en diferentes regiones, cada una de ellas manejadas por DBMS locales.

2) *Multiplicada y distribuida.* Con este método el tiempo de respuesta se reduce pero los problemas de actualización, redundancia e integridad aumentan de modo importante al igual que los costos debido a la redundancia extrema. El concepto es aplicable si las actualizaciones a la base son simultáneas en cada nodo como sucede cuando provienen de un medio externo que es el caso de los sistemas de adquisición de datos.

3) *Distribuida combinando réplicas y particiones.* Este método conjuga los dos anteriores y actualmente es el más usado; en principio la base se fragmenta, pero si una sección es solicitada frecuentemente en más de un lugar se permite la redundancia.

Con celeridad aumenta la cantidad de sistemas basados en computadoras personales o bien combinaciones de mainframe con terminales inteligentes PC. Aunque los DBMS desarrollados para microcomputadoras son operacionalmente semejantes a los de mainframe ya que su creación fué inspirada en estos, difieren considerablemente en cuanto a tamaño, complejidad y prestaciones sobre el manejo de recursos.

En general, los sistemas distribuidos mejoran el tiempo de respuesta al estar la información físicamente en el lugar que se requiere, reducen costos de comunicación, proporcionan a los usuarios un mejor control sobre sus datos y a diferencia de un sistema centralizado no dependen de un solo servidor el cual cuando falla provoca paralización completa.

El concepto de distribución ha tocado no solo la arquitectura del hardware, el diseño de software con el crecimiento en variedad y complejidad de las operaciones sobre bases de datos ha llevado a una separación formal de funciones, aplicando el término de "módulo" a un conjunto

pequeño pero integrado de programas. Esta metodología es ampliamente usada en proyectos de desarrollo de interfaces hombre-máquina donde es común la intervención de varias personas.

Los avances tecnológicos deben aplicarse razonadamente para no caer en errores propios de los habitantes de la isla de Lupata[11], donde sus habitantes por el simple hecho de "modernizarse" estaban dispuestos a todo; a veces se implanta un sistema de cómputo innecesario, por ejemplo en aplicaciones donde la información únicamente es consultada, sin cambios. En tales circunstancias suele funcionar tan eficientemente y a un costo menor sistemas de información impresa o microfilmada.

1.1.5. Manejo de la información en una BD

En secciones anteriores mencionamos la ventaja que representa el contar con rutinas de acceso generalizadas que liberan al programador de la codificación en lenguaje máquina, el programador, partiendo de estas rutinas y apoyándose en las prestaciones del sistema operativo codifica en lenguaje de alto nivel *secciones* de programas separadas entre si por solicitudes al sistema operativo, como la detención del proceso, petición de datos a un archivo, etc.

Las *secciones* se suman para formar *procesos* que son las unidades básicas de software que el sistema operativo maneja, dichas porciones de código pueden requerir diferentes recursos y ejecutarse en paralelo. Por último, la reunión de procesos nos da el concepto de *computaciones*; operaciones básicas que desde el punto de vista de una base de datos se clasifican en cinco grupos:

1) *Elaboración y mantenimiento de la estructura de datos.* Este paso suele ser el más costoso, implica además de la codificación del esquema, interpretación y captura de datos.

2) *Actualización de los datos.* Computación que comprende adición, modificación y eliminación de registros obsoletos.

3) *Recuperación de la información.* Realizada con llaves de búsqueda o bien seleccionando registros cuyos atributos cumplan ciertas condiciones.

4) *Procesamiento y "digestión" de la información.* Tratándose de bases de datos extensas donde los datos están ampliamente difundidos la reducción de información y su subsecuente presentación en gráficas y reportes útiles en la toma de desiciones así como en el análisis de tendencias es fundamental. Frecuentemente será necesario el acceso a casi toda la base para obtener los resultados requeridos.

5) *Mantenimiento integral de la base.* Ya sea por el valor intrínseco de la información a almacenar en una base de datos o por la complejidad en las relaciones de los datos debemos poner especial cuidado en la integridad y consistencia de la información. Si este punto se

[11] *Lupata* es un lugar imaginado por J. Swift

deja a un lado, el esfuerzo realizado para sustituir un sistema de información manual efectivamente probado será inútil.

1.2. MEDIOS DE ALMACENAMIENTO

1.2.1. Historia

El desarrollo de las bases de datos está íntimamente ligado a los avances en la tecnología de los medios de almacenamiento, en general, para cualquier aplicación sería ideal que sus necesidades de almacenamiento fueran satisfechas usando memoria principal (de acceso aleatorio) y que además, esta fuese no-volátil y con capacidad de re-escritura; ya que esto por el momento no es posible, el uso de equipo para almacenamiento externo permite por medio de manejo de archivos, memoria virtual y "schedulers" extender la capacidad potencial de memoria buscando ante todo, balance entre la velocidad de ejecución del CPU y el tiempo de trasferencia Entrada/Salida del dispositivo.

Ya a finales del siglo pasado la información era guardada mediante códigos especiales perforados en cintas y tarjetas de papel (como la tarjeta Hollerit) para posteriormente recuperarla con dispositivos ópticos o electro-mecánicos. Fue hasta el principio de la década de los 50's cuando cintas magnéticas digitales de media pulgada hicieron su aparición, de esos tiempos a la actualidad la densidad de datos en medios magnéticos de cinta ha aumentado 100 veces por lo menos.

El acceso aleatorio inició con equipos cuyo tiempo de acceso era de hasta varios segundos, tales dispositivos a pesar de contar con "pilas" de unos 20 discos solo tenían un par de cabezas desplazándose de arriba a abajo sobre un brazo; los avances en esta tecnología han sido en dos direcciones principalmente; primero, en cuanto al tiempo de acceso, mejorado con paquetes de disco de ensamble integral y segundo, en cuanto a la densidad de almacenamiento que ha aumentado en más de un factor de 250.

Es tan rápida la carrera tecnológica en el área de almacenamiento que nos vemos obligados a tomar en cuenta la "duración económica" de los equipos, tiempo correspondiente a la vida en el mercado de un dispositivo específico. Muestra de ello es la constante búsqueda de nuevas opciones de almacenamiento como memorias de burbuja magnética, dispositivos ópticos, sistemas neuronales, y el notable mejoramiento en costos y capacidad de almacenamiento en memoria principal permitiendo que ciertas aplicaciones hagan uso extensivo de archivos virtuales.

1.2.2. Descripción general de los sistemas de almacenamiento

Cintas magnéticas

Las cintas magnéticas son bandas plásticas cubiertas de óxido de hierro magnetizable que fácilmente llegan a medir 740 metros, a lo ancho se graban nueve o siete canales, uno de los cuales contiene bits de paridad mientras que los restantes en conjunto forman el flujo de caracteres. De ahí que la densidad lineal de bits y bytes por pulgada sean las mismas con valores variando entre 800 y 6250 bits por pulgada.

Se ha intentado reducir el tiempo de acceso (comúnmente del orden de cuatro minutos para leer 30 megabytes) tomando ideas originalmente desarrolladas para el registro de señales de video, en la Figura 1.3 podemos observar la disposición de las pistas en el registro longitudinal típico y en el helicoidal donde además de la información en una de las orillas de la cinta existe una pista de sincronización. Aplicando esta técnica, la capacidad alcanzada es de hasta 5600 megabytes en un mismo carrete y tiempos de búsqueda de bloque promedio inferiores a 30 segundos.

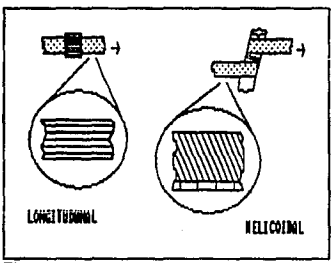


Figura 1.3
Registros longitudinal y helicoidal

En las cintas magnéticas, físicamente, los archivos son organizados en bloques consecutivos (almacenanados contiguamente). Por esta restricción resulta impráctica otra forma de organización de archivo que no sea la secuencial.

Tomando en cuenta velocidad de acceso, secuencialidad de la información, fácil manejo y tamaño compacto, las cintas magnéticas actualmente se emplean para almacenar grandes cantidades de datos principalmente en calidad de copias de respaldo.

Discos magnéticos

Los medios de almacenamiento permanente para una base de datos están monopolizados por discos magnéticos ya que a un bajo costo proporcionan tiempos reducidos en el acceso directo a datos. Los discos se organizan en pistas o anillos concéntricos los cuales a su vez, están subdivididos en unidades mínimas de almacenamiento y recuperación de información (bloques o sectores). A pesar de no poder hacer referencia directa a cada byte de información, el hecho de accesar a cualquier sector en el disco sin restricciones de acceso previo a otros como sucede con las cintas magnéticas, los clasifica como dispositivos de acceso directo.

Para mostrar el avance que ha tenido esta tecnología basta con hacer notar que la cantidad de información que se puede almacenar en dispositivos de tamaño semejante aumentó aproximadamente 100 veces en la decáda de los 70's. Otro aspecto impulsado por la necesidad de transportar e intercambiar datos es el nacimiento de los discos flexibles, los cuales se recubren de un material llamado *Mylar* y envuelven en empaques plásticos con las perforaciones necesarias para hacerlos girar y determinar su posición.

Los discos magnéticos fijos (llamados así porque el disco no puede removerse de la unidad correspondiente) ya sea sencillos o en paquete se cubren con óxido ferromagnético, semejante al depositado en cintas. Los paquetes de discos se forman apilando varios de ellos en un mismo eje, dejando el espacio necesario entre cada uno para un brazo móvil con una cabeza lectora/grabadora en el extremo de este modo, hay casi tantas cabezas como superficies de disco (la superficie inferior y superior, "las tapas" comúnmente carecen de cabezas) pero solo una de ellas transfiere información en un momento dado.

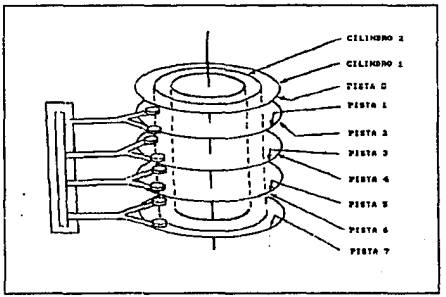


Figura 1.4
Paquete de discos

En un intento por reducir los tiempos de búsqueda, modificando la geometría del disco, se diseñaron dispositivos en forma de tambor con una cabeza por pista quedando así cancelado el movimiento de cabezas sobre brazos, desafortunadamente, el número de pistas es menor y por tanto la capacidad del dispositivo también.

Antes de analizar al detalle los parámetros técnicos de funcionamiento, conviene estudiar la estructura física y logica de los discos.:

Las pistas cuyo ancho comúnmente es de 10 milésimas de pulgadas, mientras más próximas al centro del disco tendrán menor diámetro pero a pesar de ello para facilitar el acceso, todas tienen la misma capacidad de almacenaje así, aunque tengan diferentes densidades de bytes por pulgada, la velocidad de bytes por segundo que pasan bajo la cabeza es la misma simplificando servomecanismos y garantizando el buen funcionamiento del lector de discos. En paquetes de discos se le llama cilindro al conjunto de pistas con igual radio pero que pertenecen a diferentes superficies de grabación.

Las pistas a su vez se dividen como ya habíamos mencionado en unidades de transferencia básicas las cuales si son de longitud variable se les denomina *bloques* (usualmente el tamaño es definible por software), y *sectores* cuando el largo es fijo. La cantidad de bytes por sector o bloque es generamente baja, 512 es un valor típico. Los archivos surgen de la concatenación entre sectores y por ello es importante su organización; la mas sencilla es la *contigua* que como su nombre lo indica, 9almacena los archivos en "grupos" de sectores físicamente cercanos; la *ligada* donde en cada sector existe un "apuntador al siguiente" y por *tabla de mapeo* (FAT: File Allocation Table), organización que encadena los sectores de cada archivo por medio de un mapa de bits mantenido en un sector del mismo disco con el que es posible determinar áreas ocupadas y disponibles de manera que la creación o borrado de archivos implica consulta y actualización del mapa.

En caso de contar con varios paquetes de unidades de disco se llegan a requerir los siguientes parámetros para una dirección física:

A) Identificación del dispositivo físico
B) Cilindro
C) Superficie
D) Sector o bloque según el caso
E) Registro(s) de interés
F) Campo(s) o caracter(es)

Nótese como hasta el inciso (D) está implícito movimiento físico de mecanismos, cada petición debe especificar claramente un solo elemento de dirección que recupere un bloque, la unidad básica de transferencia, mientras que en los incisos siguientes se plantea el acceso al buffer presente en memoria principal. Bien podríamos usar directamente este tipo de direccionamiento en nuestras peticiones de información pero además de resultar de difícil manejo, tomemos en cuenta que cada dispositivo tendrá su propia cantidad de cilindros, superficies, bloques, etc. o

simplemente, si el archivo que originalmente contenía los datos es copiado a otro disco es muy probable que la información sea ubicada en diferente lugar.

Ya que la petición de un solo caracter nos obliga a la recuperación de todo el sector donde se encuentra, es recomendable guardar los registros lógicos en múltiplos de sectores o tratándose de registros pequeños deberán guardarse registros completos en cada unidad de transferencia (mas adelante hablaremos sobre el "factor de bloqueo") por eso; los dispositivos cuyo acceso es a través de bloques de longitud variable son preferidos pues aprovechan mejor el espacio disponible.

Almacenamiento en memoria RAM

Como ya mencionamos, resulta preferible cuando el hardware lo permite, mantener los datos correspondientes a un sistema en memoria principal; buscando aumentar las posibilidades de esta opción, existen constantes investigaciones sobre dispositivos acoplados por carga, registros de corrimiento y circuitos MOS, los cuales compiten contra medios magnéticos de almacenamiento para ciertas aplicaciones. Es de esperar que en un futuro sus principales inconvenientes como costo, volatilidad y tamaño sean resueltos.

1.2.3. Transferencia de la información

Físicamente, en los discos magnéticos la unidad de información mínima a la que tenemos acceso debido a las limitaciones del hardware se denomina sector, cuyo largo queda determinado al "inicializar" o "formatear" el disco, pero desde la perspectiva de los procesos o transacciones la unidad de información que nos interesa es *el bloque* (determinado por software), que puede estar formado por uno o varios sectores y que será el conjunto de bytes de tamaño fijo que efectivamente es transferido.

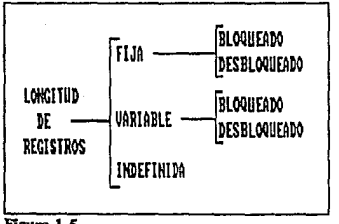


Figura 1.5
Relación entre registros y bloques

Bloque se puede definir como *"La unidad de información de intercambio entre dispositivos de almacenamiento y un área de trabajo (buffer) en memoria principal"*. prácticamente toda

aplicación hace uso de buffers para almacenar temporalmente información ayudando a sincronizar la disparidad entre las velocidades procesador / unidad controladora de disco.

Las relaciones que podemos establecer entre registros y bloques se indican en la Figura 1.5; entendiéndose por *"bloquear registros"* a la asignación de un número específico de registros físicos a recuperar en cada operación de lectura/escritura. Determinar el tamaño de los bloques no es una tarea sencilla, se deben tomar en cuenta varios factores para garantizar el desempeño del sistema, si eligimos un tamaño grande de bloque al hacer una petición serán arrastrados datos innecesarios, por el contrario, un tamaño insuficiente es identificado por la generación de varias solicitudes de acceso para obtener los datos correspondientes a una tarea.

Buffers

Buffer es el área que recibe un bloque y que comúnmente corresponde a una página[12] en memoria principal de donde los procesos toman, actualizan y agregan datos, su objetivo principal es optimizar el desempeño o utilización de los dispositivos de almacenamiento secundario atendiendo lo más pronto posible las demandas del CPU.

El abrir muchos archivos o trabajar en ambientes multiusuario ocasiona consumo de grandes cantidades de memoria, por ejemplo, un mainframe atendiendo peticiones de red[13] puede llegar a tener abiertos unos 100 archivos diferentes y si hay dos buffers de 2048 bytes por archivo, se estarán consumiendo 400 Kbytes, únicamente en buffers,

Acerca de la optimización en el manejo de buffers en ambientes multiusuario, antes de asignar un nuevo buffer, el sistema operativo revisa si alguien pidió ya el mismo bloque de información, de ser así, simplemente se compartirá el buffer. Y ya que son prioritarios los buffers para sostener el procesamiento, deben disponerse dinámicamente nuevas áreas de memoria o bien en multiprogramación por página, retirar páginas de procesos de baja prioridad. Por estas razones, si contamos con la posibilidad de modificar el tamaño de los buffers debe ajustarse no solamente para que exista un número de registros entero por buffer sino también para que haya una relación semejante entre tamaño de página, buffer y bloque.

Bloqueo

La decisión del método de bloqueo dependerá en mucho de las necesidades de actualización y crecimiento (inserción o agregado) del archivo, debe considerarse también si los registros son de longitud constante o variable. Para ello tomaremos los siguientes parámetros:.

1) *Densidad de carga.* Cuando se espera considerable cantidad de inserciones o posible crecimiento de registros de tamaño variable, es recomendable no utilizar en su

---

[12] Campderrich B., *Técnicas de Bases de Datos*, p149

[13] Gio Wiederhold, *Diseño de Bases de Datos*, p66

totalidad el espacio de cada bloque para asi poder agregar información nueva. Será hasta llenar el bloque cuando haga falta ligar uno nuevo al archivo. La relación entre longitud total y espacio reservado para inserciones es conocida como densidad de carga.

2) *Localidad.* Resulta claro que la obtención de una secuencia de registros será más rápida si estos están físicamente cerca o como se dice, si la estructura de datos es "conglomerada".

Una de las actividades más comunes es la lectura completa de un archivo para procesarlo íntegramente a primera vista, la mejor organización es la contigua pero como el tiempo de procesamiento no es nulo, mientras un bloque es leído y procesado para accesar el siguiente sería necesario dar una revolución de disco extra. El método para evitar esta situación consiste en que los bloques contiguos no se ubiquen físicamente uno seguido de otro sino "salteados" (interleave) tal como lo muestra la Figura 1.6.

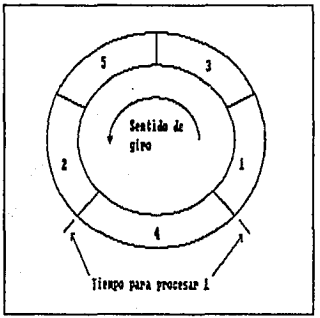


Figura 1.6
Interleave 1:3

Otro método de optimización controlado a nivel sistema operativo se implementa en el manejador cronológico estableciendo una política de manejo diferente a la que aparentemente es más justa; es decir, la de "primero en entrar - primero en salir" por la de "menor tiempo de servicio primero" (SSTF - Shortest Service Time First) atendiendo asi los accesos a disco no en estricto orden prioritario o cronológico sino con la adición de un criterio de acceso a los bloques conforme vayan pasando bajo la cabeza lectora.

Finalmente, el acceso puede mejorarse todavía "escalonando" el inicio de las pistas, (este método requiere de cierto control por hardware) para evitar el retraso ocasionado por el movimiento del brazo en la búsqueda de una nueva pista. O bien renumerando los sectores de

diferentes pistas a modo que el primero de una pista cualquiera se encuentre desplazado respecto al anterior tal como se muestra en la Figura 1.7.

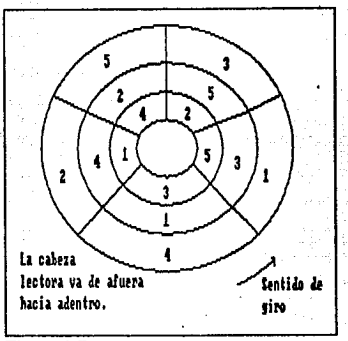


Figura 1.7
Renumeración de sectores

1.2.4. Parámetros técnicos

En general, podemos hablar de dos medidas de eficiencia: de acceso y de almacenamiento. La **eficiencia de acceso** será mayor en cuanto exista menor cantidad de accesos físicos por solicitud de un registro lógico específico, la calculamos con una simple división; por ejemplo[14]: si para encontrar un registro el controlador de dispositivo hace dos accesos, entonces la eficiencia será de 1/2 = 0.5

La **eficiencia de almacenamiento** (conocida también como densidad de carga) es la relación entre la cantidad de bytes necesarios para guardar cierta información y la empleada realmente, medida a su vez en bytes. Recordemos que además del dato en si, se almacenan algunas tablas, mapas, información de control en general y en ocasiones existe espacio libre para inserciones.

El diseño completo de una base de datos no puede dejar al aire el desempeño del medio de almacenamiento que será usado, un estudio a fondo además de considerar las medidas de eficiencia arriba mencionadas deberá tomar en cuenta los siguientes parámetros:

[14]Atre, Shakuntala, *Técnicas de Bases de Datos*, p201

Tiempo de Localización
Latencia
Razón de Transferencia
Factor de Bloqueo

Tiempo de acceso aleatorio por localización

El retraso promedio que implica llegar a un punto del que se va a extraer información específica, partiendo de un lugar indeterminado se le llama en conjunto *Tiempo de acceso aleatorio*, sin embargo se pueden reconocer englobados dentro de este dos tiempos: el de *localización*, que se refiere al retraso debido al posicionamiento del conjunto brazo-cabeza sobre la pista adecuada y el de *latencia* que abordaremos más adelante.

La Figura 1.8 muestra una gráfica típica de tiempo de búsqueda contra cilindros recorridos, para las consideraciones prácticas lo que usamos es un tiempo promedio, que en el caso de las unidades de disco modelo RZ23 usadas en las VAX es de 25ms. En todo caso, estos tiempos suelen ser menores, especialmente si hubo cuidado de asignar bloques contiguos para todos los archivos; por otro lado, cuando se espera apertura y acceso a dos archivos por una misma aplicación, es buena idea dejarlos en diferentes unidades de disco si contamos con mas de una.

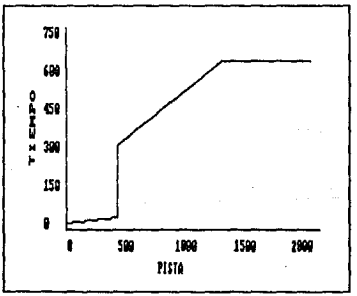


Figura 1.8
Tiempo de búsqueda vs Cilindros recorridos

Tiempo de acceso aleatorio por latencia

El otro tiempo englobado en el acceso aleatorio hace referencia, en el caso de los discos magnéticos, al retraso debido a la rotación hasta poder leer los datos de ahi que se le conozca también como *latencia rotacional*. El retraso dependerá del tamaño de los sectores y de la razón de transferencia de datos.

Dependiendo del dispositivo y de la cantidad de marcadores de bloque para ubicar cierta información puede llegar a ser necesario moverse al punto inicial de toda la pista, pero si existe un medio para detectar el inicio de bloques, en promedio un acceso aleatorio tendrá que pasar la mitad de la pista antes de llegar al bloque; en otras palabras, el tiempo de latencia rotacional *tr* será:

$$tr = 0.5*60 / R \qquad [s]$$

Donde 'R' es la velocidad del disco en revoluciones por minuto y *tr* se da en segundos. Para el caso de los controladores de disco usados, la velocidad según hojas de datos del fabricante es de 8.4ms lo que concuerda con la velocidad especificada de 3575 rpm:

$$30 / 3575 = 8.39 \times 10^{-3}$$

Razón de transferencia

Ya ubicada la cabeza en el lugar adecuado, sigue el tiempo realmente necesario para leer o escribir la información que por lo general es igual para ambas actividades. En discos magnéticos se obtiene a partir de la velocidad de rotación y de la densidad de bytes en el medio de almacenamiento, expresándose en bytes por milisegundo o en Kbytes sobre segundo. Las unidades de disco de la VAX tienen una razón de transferencia muy alta: 1250 Kbytes/s

Factor de bloqueo

Prácticamente la evaluación del desempeño del acceso a un archivo se mide en registros y no en bloques de ahí que el ajuste realizado entre registros y bloques conocido como factor de bloqueo (*Bfr*) sea un parámetro decisivo; su valor indica la cantidad de registros por bloque. Diferentes métodos de bloqueo se aplican a diferentes necesidades, a continuación mencionamos los tres considerados básicos.

Bloqueo Fijo. Se asignan registros completos de tamaño fijo a cada bloque de modo que cualquier espacio no ocupado queda desperdiciado. El tamaño de bloque es una limitante en el diseño.

Bloqueo Variable Espaciado. Para archivos con registros de longitud variable, los bloques se llenan con secciones de registros con objeto de aprovechar espacio, utilizando el apuntador del siguiente bloque para terminar la lectura de registros truncados; la eficiencia de este método se ve disminuida tanto cuanto sea la razón de los bloques completos a los truncados. El tamaño de bloque se ajusta de acuerdo al buffer disponible.

Bloqueo Variable No Espaciado. A fin de evitar doble acceso para una sola petición de usuario únicamente se guardan registros completos en cada bloque, desperdiciándose por tanto gran parte del espacio si los registros son de longitud variable. El bloque se ajusta a múltiplos del tamaño promedio de los registros.

1.2.5. Tendencias

Los avances en hardware se reflejan en mejores opciones para el diseño de software; en el ambiente específico de desarrollo (VMS), existe facilidad de paginación (direccionamientos virtuales en un espacio de memoria real mucho menor del supuesto). Además contamos con grandes cantidades de memoria RAM (16 Megabytes) pudiendo mantener ahí la información mas frecuentemente solicitada. Este medio ambiente de trabajo ya fué descrito con anterioridad y representa una opción segura y probada dentro de las tecnologías recientes.

Usar tecnología del campo de la óptica en lugar de magnetismo para almacenamiento, ofrece la ventaja de una sustancial reducción en el área requerida por bit, puesta en evidencia desde los primeros intentos basados en procedimientos ya empleados en microfilmado mediante técnicas fotográficas, gradualmente la extensión en el uso de la tecnología láser en diferentes aplicaciones llegó hasta el área de computación; desde procesadores ópticos (en desarrollo) hasta discos para lectura y escritura con luz láser (existen ya varios modelos comerciales). Y muy a tiempo, pues a pesar de que con cada década aumenta la densidad de información en discos magnéticos (gracias a la solución de problemas de posicionamiento básicamente), se está llegando al punto teórico máximo debido a las limitaciones físicas en cuanto al tamaño del área magnetizable.

Los primeros discos ópticos comerciales proporcionaron gran capacidad de almacenaje pero presentaban la desventaja de permitir únicamente lectura. La primera y única vez que se realiza la escritura en tales discos provoca pequeñas quemaduras en su superficie; aplicación posterior de luz láser de menor intensidad identifica la presencia de las marcas. Otros dispositivos mas recientes, con facilidad de re-escritura, usan las propiedades fotomagnéticas de algunos materiales consiguiendo reducir el área magnetizable al aplicar concentrados haces de luz emitida por diodos láser.

Por otro lado, hasta hace algunos años hubiera sido imposible pensar en almacenar una base de datos en su totalidad empleando semiconductores; pero ahora es una alternativa muy viable, incluso para ciertas aplicaciones pueden implementarse por hardware operaciones rutinarias extrayendo de los circuitos integrados directamente resultados, es decir, los datos ya procesados.

Un ejemplo son los dispositivos de almacenamiento asociativo que, a diferencia de la memoria de acceso aleatorio común en la que damos la dirección de los datos buscados, efectúan la recuperación de registros que cumplen ciertas condiciones mediante operaciones de hardware en paralelo comparando uno o varios campos llave con el argumento de búsqueda, en otras palabras, no se indica donde se encuentra la información sino lo que se busca. Desafortunadamente este hardware "dedicado" aunque mejora el desempeño, reduce flexibilidad y capacidad. El hardware asociativo por medio de grandes arreglos de dispositivos semiconductores que conforman el equivalente de un bloque, invariablemente correspondiendo a un registro, efectúa comparaciones entre estos y "campos de coincidencia"; al realizarse la búqueda como tarea propia del dispositivo, la CPU está disponible para otros procesos. Resulta claro de la descripción anterior que el diseño a la medida del hardware obliga a su uso exclusivo en la aplicación para la que fue hecho.

Un tipo de elementos para almacenamiento que tomó popularidad por un tiempo, ideados en los laboratorios Bell son las *memorias de burbuja magnética* que llegaron a trascender comercialmente con dispositivos de hasta 128 Kbits por chip, ofreciendo tiempos de acceso promedio de 4ms, no volatilidad y gran resistencia a vibraciones, pero no consiguieron suficiente aceptación en el mercado.

Debido a la existencia de sistemas completamente dedicados al manejo de bases de datos se ha tenido que diseñar no solamente software sino incluso hardware específico que ofrezca operaciones como la actualización de índices o búsquedas lo cual disminuye tiempo de CPU consumido. El equipo usado con esos fines puede ser una computadora independiente o un dispositivo adicional inteligente, tal es el caso de ciertos controladores de disco que cuentan con procesadores periféricos propios, encargados de obtener los bloques de datos que se van a leer, verificar la transmisión correcta y por último, informar al procesador central sobre el término de la actividad, el cual no tiene que "conocer" el lugar donde residen realmente los datos, ya que la asignación de archivos puede dejarse completamente al cargo del subprocesador.

Concluyendo, nuevas tecnologías han influido e influirán los enfoques dados al almacenamiento de información que dia con dia debe responder a las exigencias que impone el manejo de cantidades crecientes de información, tal vez en un futuro con la reducción de costos de almacenamiento en memoria principal, el almacenamiento secundario se ocupe únicamente para respaldo.

1.3. ARCHIVOS

Ya que una base de datos es finalmente un conjunto de datos relacionados cuyo almacenamiento se logra por medio de archivos, se puede decir que la organización de archivos es piedra angular de las bases de datos. Por tanto, es necesario un estudio de los tipos de archivo tanto para el diseño de una base como para su implantación.

1.3.1. Conceptos generales

Para estandarizar el lenguaje utilizado, antes de entrar de lleno en lo que son las características de los tipos de archivo, se darán definiciones y conceptos generales sobre sus elementos conformadores:

Campo. En algunos textos se considera que "campo" es la unidad de información más pequeña a la cual se puede hacer referencia en un programa de computadora. Mas correcto sería decir que los campos contienen *unidades dato elementales* las cuales describen de alguna manera un objeto o evento. Y esto aun con ciertas reservas pues a pesar de que en la mayoría de los casos cada campo contiene una estructura de datos básica, suele suceder que un campo es procesado para extraer dos o más datos.

Cuando es necesario identificar de manera única cada registro de un archivo (que es lo usual), deberá seleccionarse un campo que cumpla con el requisito de correspondencia biunívoca, escogiéndolo de entre varios posibles candidatos o *claves aspirantes*, a este campo se le llama *clave primaria*.

Registro. Es la reunión de campos-dato relacionados entre sí o más específicamente, una estructura lineal que agrupa elementos dato. Podemos hablar de dos clases de registros: lógicos y físicos, un registro lógico pone de manifiesto la percepción del usuario o programador sobre una *entidad*, mientras que el registro físico aparte de la información correspondiente al registro lógico almacena datos referentes a su ubicación y organización interna.

Los archivos controlados por FMS suelen tener una estructura sencilla con campos de longitud fija y orden predeterminado, pero en algunas bases de datos esa estructura no es adecuada de ahí que otras disposiciones hayan sido ideadas, de entre ellas, se identifican cuatro formas básicas de organización para la representación de campos[15]:

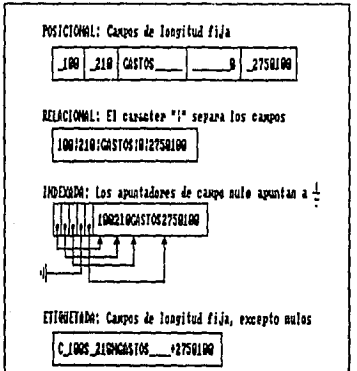


Figura 1.9
Organización de los campos dentro de registros

[15] Tsai, Alice, *Sistemas de Base de Datos*, p23

1) Posicional: Se trata del método más sencillo al que recurren comúnmente los sistemas de manejo de archivos, ciertamente esta estructura es fácil de manipular ya que se dejan espacios de longitud fija para cada campo pero por la misma razón, si varios de ellos no se llenan en su totalidad con información es decir, si la densidad es baja, habrá considerable desperdicio de espacio.

2) Relacional: En este caso se permiten registros de longitud variable y a modo de separación entre cada uno se agrega a los últimos bytes un *delimitador*, representado por un caracter de control cuyo uso estará prohibido en cualquier otro sitio de la base.

3) Indexada: Por medio de apuntadores que consisten en desplazamientos relativos en bytes queda indicado el inicio de cada campo almacenado.

4) Etiquetada: Util cuando algunos campos no solamente se llenan a medias sino incluso están vacios.

Archivo de Datos. La reunión de de registros similares produce archivos, los cuales exceptuando el caso de archivos virtuales, se conservan en dispositivos de almacenamiento secundario. Cuando estos registros son de igual tamaño y tienen los mismos campos, se dice que el archivo es *homogéneo* o *uniforme*. De un modo gráfico los archivos homogéneos son representados como "tablas" en las que cada renglón correponde a una entidad y cada columna a un atributo.

Conceptualmente es común que los usuarios se hagan a la idea de que un archivo es una larga secuencia de registros uno tras del otro. Pero para nuestros fines complementando la definición de archivo, se considerará de ahora en adelante que además de contener registros semejantes, un archivo debe presentar cierta "estructura u organización" consistente. Ya se ha hablado de *estructura de datos* y es tiempo de establecer que tal concepto se refiere en general a una colección de nodos relacionados íntimamente entre si, los cuales son piezas básicas en el almacenamiento de información.

Dejando a un lado el tipo de información almacenada en los archivos, por el grado de aprovechamiento de espacio se clasifican en: denso, disperso y redundante.

Los sistemas de archivos casi siempre cuentan con archivos auxiliares aparte de los encargados de contener la información en si, ejemplo de esto son los archivos índice formados por parejas de anotaciones de un valor de atributo y su correspondiente apuntador a algún registro del archivo indexado.

1.3.2. Estructuras de datos

Mencionamos que una estructura de datos es una colección de nodos o unidades de información básicas relacionados entre si. No intentaremos mostrar la amplia gama de tipos de estructuras de datos, se tocarán únicamente las más significativas para las diferentes organizaciones de archivo

ya que precisamente es dentro de los archivos, donde los datos deben tener cierta organización que los identifique y sea el medio para recuperarlos a memoria principal.

Arreglos

Un arreglo es una estructura de datos con un número de nodos definido, el arreglo en conjunto está asociado con un nombre y cada elemento tiene un índice numérico formado por al menos un entero. Los elementos del arreglo pueden guardarse de manera contigua, vectorizada o ligada:

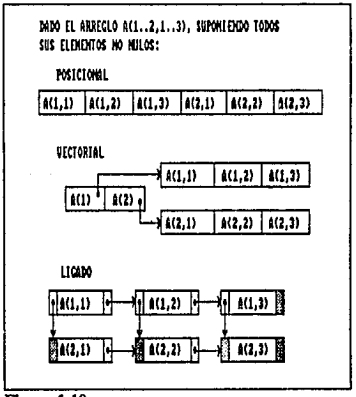


Figura 1.10
Formas de almacenamiento de los elementos de un arreglo

Como podemos observar, en organizaciones contiguas (que son las más comunes) la función de mapeo 1 a 1 es fácil de obtener, por ejemplo, en lenguaje C donde el primer elemento de un arreglo está definido por el índice (0,0,...,0), un arreglo "A" formado por x1 * x2 * x3 * ... * xn elementos de longitud L (en bytes) y que se empieza a almacenar en la dirección "d" tendrá la siguiente función de mapeo:

```
(i1,i2,i3,...,in)=
     d + L*(i1*x2*x3*...*xn +
            i2*x3*x4*...*xn +
            .
            .
            i(n-1)*xn       +
            i(n)                )
```

La anterior representación de arreglos no resuelve correctamente algunos problemas; por ejemplo, si el manejo de los datos requiere intercambio de renglones o columnas la forma vectorizada es la indicada, por otro lado, si la cantidad de información a almacenar es muy grande y se espera alto porcentaje de valores por omisión, una estructura ligada ofrece a cambio de tiempo consumido en búsqueda lineal economía de espacio, tómese como ejemplo la vectorización del arreglo mostrado en la Figura 1.11:

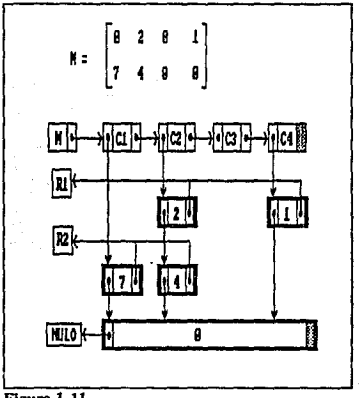


Figura 1.11
Vectorización de un arreglo

El algoritmo de recuperación del elemento M[i,j] para esta estructura suponiendo que el índice del primer elemento sea (1,1) es el siguiente:

```
x= M
HACER col = 1 HASTA j
  x= LIGA_A_COL(*x)
FIN
MIENTRAS (*LIGA_A_REN(*x) <> i) Y (*LIGA_A_REN(*x) <> NULO)
  x= LIGA_A_SIG(*x)
FIN
VALOR= *x
```

Pilas

La pila es una estructura de datos lineal con la política de último en entrar - primero en salir y hablando de sistemas de archivos su equivalente son los apilos; en ambos, las ligas entre elementos exceptuando la del último no pueden destruirse (mas adelante se estudiarán al detalle y veremos que existen diferencias en la eliminación de nodos).

Listas circulares

Se trata de una estructura de datos ligada y lineal en la que el siguiente elemento al último es el primero, la representación gráfica tradicional es la indicada en la figura 1.13:

En la sección 1.3.4 trataremos los archivos multianillo en los cuales se aplica directamente esta estructura de datos, en tales archivos la inserción y eliminación de elementos está permitida en cualquier punto del anillo mediante modificación de apuntadores.

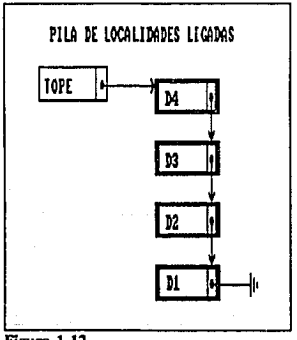


Figura 1.12
Pila

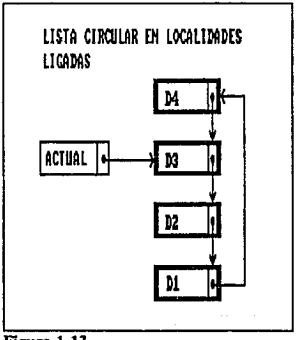


Figura 1.13
Lista circular

Arboles

Los árboles son un tipo de estructura no lineal, compuesta, y que en el ámbito de las bases de datos han encontrado aplicación directa en archivos indexados; la gráfica típica correspondiente a un árbol es la siguiente:

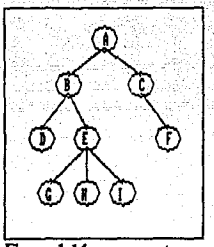


Figura 1.14
Arbol

Para que una estructura múltiplemente ligada tenga la categoría de árbol, se deben cumplir ciertos requisitos; apareciendo al mismo tiempo algunos conceptos relacionados con sus elementos:

* El número de nodos es igual a la cantidad de ligas mas uno.

* A todos los nodos del árbol excepto al primero llega una liga, el nodo que no es apuntado por otro elemento se le llama nodo raiz.

* Entre dos nodos no puede haber más de una liga.

* Todo nodo que apunta a otro se dice "padre" de este, el apuntado a su vez se le llama "hijo".

* Se dice que dos nodos son "hermanos" cuando ambos son apuntados por un tercero.

Internamente, las representaciones de árboles en computadora se implementan de varias formas, una de ellas es con *matrices de adyacencia* que son arreglos en los que existe un renglón por cada nodo "padre" y en ellos se marcan con uno lógico las columnas correspondientes a sus hijos; o bien mediante vectores, representación más compacta que aprovecha la circunstancia de que un nodo solo puede tener un padre; de este modo, en el vector habrá tantos elementos como nodos y cada elemento contendrá el nombre del nodo padre o un apuntador al mismo tal como se indica en la Figura 1.15:

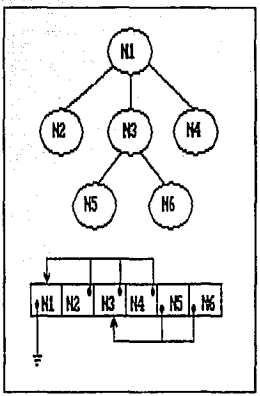


Figura 1.15
Representación de un árbol mediante un vector

Posteriormente veremos algunas clases de índice ideadas tomando el modelo de determinados tipos de árbol; como ejemplo tenemos los índices balanceados que resultan de aplicar estructuras tipo árbol balanceado:

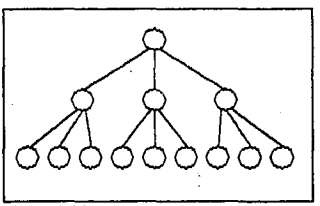


Figura 1.16
Arbol balanceado de orden 3

De la Figura 1.16 resulta evidente la característica que es necesario agregar a las condiciones vistas anteriormente que definen a un árbol, formalmente se pude decir que:

"Un árbol balanceado de orden -n- tendrá -n- cantidad de hijos por nodo padre".[16]

Otro tipo de árbol, que agrega flexibilidad a la estructura anterior es el árbol-B, donde la regla adicional se modifica a:

"Los nodos padre de un árbol-B de orden -n- pueden tener entre (n - 1) / 2 y (n - 1) nodos hijo, a excepción del nodo raíz, el cual como mínimo puede tener un hijo".

En las organizaciones de archivos indexados que se van a someter a frecuentes inserciones, los árboles-B resultan más eficientes ya que después de una reorganización de archivo los nodos se pueden dejar con el número de ligas mínimo.

1.3.3. Parámetros de desempeño

Para tener una guía práctica en la elección del tipo de archivos que conformarán una base de datos es necesario además del conocimiento conceptual y empírico de su estructura, contar con un método evaluativo. Por ello consideramos a continuación seis medidas de desempeño,[17] que contemplan las operaciones básicas dentro de una base: inserción, agregado, modificación y eliminación, tomando en cuenta los parámetros relacionados con el hardware de la sección 1.2.4 con los que están estrechamente ligados. Con fines comparativos, los comentarios que sobre dichos parámetros se hagan al tocar las diferentes organizaciones de archivos, supondrán la existencia de un solo buffer por archivo de igual tamaño a los bloques.

TAMAÑO DE REGISTRO: Este parámetro compara el grado de aprovechamiento de espacio proporcionado por las diferentes organizaciones, se debe tomar en cuenta aun cuando la cantidad de espacio en disco no represente problema ya que en general es mejor manejar archivos y paquetes de información pequeños y compactos.

RECUPERACION DE REGISTROS: Como ya se había visto en la parte dedicada a los parámetros de hardware, el tiempo necesario promedio para recuperar un registro se compone en primer lugar de la localización del lugar físico donde está la información y posteriormente la lectura propiamente dicha. Recuperar un registro suele requerir mas de una acceso ya que para encontrar el bloque que corresponde a una llave determinada, invariablemente se deben leer tablas o mapas de localización.

RECUPERACION DEL SIGUIENTE REGISTRO: A diferencia de la localización de un registro en la que está implícita la relación asociativa atributo-llave, una operación de lectura o escritura lineal encierra cierta dependencia estructural. El acceso lineal se puede implementar por medio de apuntadores o índices. Resulta claro que mantener una "localidad" cercana de registros sucesores reduce considerablemente el esfuerzo de recuperación.

---

[16]Cfr. Tsai, Alice, *Sistemas de Base de Datos*, p50

[17]Wiederhold, Gio, *Diseño de Bases de Datos*, p89

AGREGADO DE REGISTROS: Aumentar la cantidad de registros puede significar en el caso de inserciones la reescritura de toda o gran parte de una base, en otras circunstancias cuando el acceso serial queda garantizado por una clave primaria y los registros se agregan al final del archivo, el parámetro de obtención del siguiente registro se ve afectado al perderse la localidad.

ACTUALIZACION DE REGISTROS: Al menos dos operaciones primarias se engloban en la actualización, primero, lectura del bloque que contenga la información deseada y segundo reescritura del mismo; pero cuando los registros son de longitud variable puede ser imposible guardar el registro modificado en su antigua ubicación a menos que se reescriba casi por completo la base, siendo entonces necesario marcar el registro como inválido (borrado), abrir y ligar un bloque nuevo.

REORGANIZACION DE ARCHIVOS: El último parámetro a considerar se refiere a operaciones periódicas de lectura y escritura debido a la inserción o borrado de registros, cambios de estructura, etc. Actividades cuyo objetivo es mantener en un nivel aceptable los otros parámetros.

1.3.4. Tipos de archivo

Aunque aplicando diferentes criterios la variedad de archivos es grande, se pueden reconocer seis tipos fundamentales: Apilo, Secuencial, Secuencial indexado, Indexado, Directo y Anillo.

Sin embargo, la gran variedad de requerimientos que debe cumplir una base de datos hace que el diseño óptimo no se consiga con estos tipos "puros" mas que en contados casos, incluso en la práctica, para cada situación, debemos tomar en cuenta el desempeño general del sistema específico donde se instalará; o si va a ser adquirido, elegir el más adecuado.

Archivos apilo

Debido a su estructura lineal, presenta varias semejanzas con las pilas ("stacks"), se trata de un archivo en el que los registros son almacenados en el orden que llegan, pudiendo ser estos de longitud variable e incluso de diferente estructura en cuanto a sus campos. Este tipo de archivo tiene como ventajas la alta densidad y facilidad en el agregado de nuevos registros (lo cual es simplemente al final del archivo), a costa de un bajo desempeño general de las demás operaciones dinámicas sobre el archivo como modificación, búsqueda y borrado.

Desempeño

1. Tamaño de registros: La estructura organizativa de los campos dentro de registros es del tipo etiquetado (ver la secc. 1.3.1), estructura que es un arma de doble filo ya que por un lado no es necesario especificar los campos inexistentes, pero por el otro, se desperdicia espacio en el almacenamiento de las etiquetas. De este modo cuando la información es heterogénea (diferente longitud de registros) o dispersa (ver secc. 1.3.1 para un ejemplo de archivo disperso) la densidad del archivo será bastante buena.

Para calcular el tamaño de registros analizamos el tipo de información que se espera recibir; o bien, recurrimos a métodos de muestreo para obtener los siguientes valores promedio:

```
C...  Número de campos por registro
N...  Longitud del nombre de los atributos (etiqueta)
I...  Longitud de los datos en si.
```

Con estos valores y considerando un caracter especial indicador de fin de etiqueta, otro de fin de información y otro más de fin de registro, se llega a la siguiente fórmula para la longitud promedio de los registros (R):

```
R= C*(N + I + 2) + 1
```

2. Recuperación de registros: El tiempo necesario para la recuperación, aun suponiendo que la probabilidad de ocurrencia es homogénea a lo largo del archivo es bastante alto, en el mejor de los casos bastará con recuperar el primer bloque (si el registro está ubicado al principio del archivo) y en el peor tendremos que leer todo el archivo. Resulta pues, que en promedio el tiempo de recuperación (t_r), despreciando el ocupado en operaciones del procesador, equivale a la lectura de la mitad del archivo. Es importante para esta operación el que los bloques sean contiguos; pero aun asi, con objeto de evitar una búsqueda por cada petición, conviene acumular varias de ellas en lote (batch) a fin de aprovechar el proceso de mejor modo.

3. Recuperación del siguiente registro: Si el orden de los registros no es cronólogico, este parámetro tendrá el mismo valor que el anterior; es decir, $t_s = t_r$. Pero suponiendo que el siguiente es efectivamente aquel que se dió de alta después, el tiempo promedio será B_n/B_f, donde B_n corresponde al tiempo de transferencia de un bloque nuevo.

4. Agregado de registros: Ya que los nuevos registros se adicionan al final del archivo, basta con posicionarse en el último bloque, leerlo a un buffer, adicionar el registro, volver a escribirlo y si hubo derrama, ligar un bloque nuevo. Agregar registros es una de las actividades más sencillas y rápidas en un archivo apilo.

5. Actualizacion de registros: Las modificaciones en archivos apilo se realizan lentamente debido a la estructura propia del archivo, primeramente debemos localizar el registro, marcarlo como inválido con un caracter especial (a estas marcas se les conoce como *epitafios*) y agregar el registro modificado al final del archivo; obviamente la serialidad cronológica se pierde y la optimización del espacio requiere de procesamiento adicional.

6. Reorganizacion de archivos: Después de varias modificaciones o eliminaciones habrá acumulación considerable de registros marcados como inválidos, para eliminar esta "basura" basta con copiar el archivo sin tomar en cuenta los registros marcados con epitafios.

Aplicaciones

Pocas son las aplicaciones en las que es recomendable recurrir a archivos apilo, solo en circunstancias donde la información no es fácil de clasificar o en investigaciones donde la naturaleza del material recolectado no se ha especificado plenamente.

Archivos secuenciales

Los registros de un archivo secuencial deben tener uno o varios campos que los identifiquen de manera única funcionando como clave primaria, de acuerdo con la cual se ordenan ascendente o descendentemente y es asi como físicamente se almacenan. Aunque ideados para hardware de cinta magnética donde el acceso era exclusivamente secuencial, con la evolución de los dispositivos de almacenamiento se han mezclado procedimientos de acceso aleatorio como veremos más adelante.

Los archivos secuenciales además del ordenamiento antes indicado, presentan otra diferencia estructural respecto a los tipo apilo: cada registro presenta la misma estructura en cuanto a atributos, consecuentemente hay un considerable ahorro de espacio al no tener que guardar en forma de pareja los atributos con sus etiquetas; solo una vez en la cabecera del archivo se almacena la estructura de los registros y si estos son de tamaño fijo, por posición identificamos cada campo; gráficamente la disposición es comparable con la de una tabla.

Desempeño

1. Tamaño de registros: Calcular el espacio ocupado por -n- registros es tarea sencilla pues todos tienen la misma longitud (R); despreciando la cabecera del archivo donde estará la estructura de los registros, el tamaño del archivo será: nR.

2. Recuperación de registros: El método tradicional establecido desde que las cintas magnéticas eran el principal medio de almacenamiento secundario, consiste en la búsqueda serial; por eso, en general, la eficiencia de acceso es semejante a la de los archivos apilo ya que la recuperación de un registro aleatorio puede llevarnos a cualquier extremo, es decir, encontrarlo en el primer bloque solicitado o en el peor de los casos hasta el último. De esta manera, el promedio será igual a la lectura de la mitad del archivo.

Contar con dispositivos de acceso directo y hacer uso de las facilidades que proporcionan en cuanto a manejo de los registros reduce el tiempo de recuperación si aplicamos otro método como búqueda binaria, o un método combinado como el de "pruebas" donde mediante una función de mapeo hacemos una estimación sobre el lugar donde se encuentra el registro.

3. Recuperación del siguiente registro: En un archivo secuencial gracias a la forma ordenada de almacenamiento el "siguiente registro" estará muy probablemente en el mismo bloque del anterior, y en esos casos el esfuerzo de recuperación es despreciable; siendo B_fr el número de registros por bloque, la probabilidad de que el siguiente registro no esté en el mismo bloque es de

$1/B_{fr}$ en cuyo caso, el tiempo requerido será igual al de la adquisición de un bloque nuevo (t_{bn}) y por tanto el tiempo promedio es t_{bn}/B_{fr}.

4. Agregado de registros: Dado que los registros se deben manetener ordenados no es posible agregarlos al final del archivo, deben colocarse en un punto específico de acuerdo con su llave. El trabajo para efectuar tal operación es considerable, primero debemos buscar el punto adecuado para el nuevo registro y luego mediante ciclos alternados de lectura/escritura tan largos como los buffers lo permitan, reescribir todo el archivo con el nuevo registro ya insertado.

Resulta claro que hacer el procedimiento anterior para cada inserción es incosteable; el problema se resuelve nuevamente recurriendo al procesamiento por lote, guardando las inserciones en un archivo simple como el de tipo apilo al que llamamos *archivo de bitácora de transacciones* hasta acumular una cantidad considerable de registros o bien, que por requerimiento de los usuarios sea obligatorio actualizar el archivo secuencial. Así la actividad de inserción es aparentemente tan rápida y sencilla como en el caso de archivos apilo aunque la actualización "real" requiera de un esfuerzo mayor.

5. Actualización de registros: Tomemos en cuenta que existen dos tipos de modificación a los registros: alterando el o los campos que forman la clave primaria o respetándolos. En el primer caso lo más indicado es escribir directamente las modificaciones en el archivo secuencial, ya que la posición del registro no cambiará si permanece con la misma llave y el tiempo de actualización apenas se incrementa en lo correspondiente a la razón de transferencia de un bloque respecto al tiempo de localización; pero cuando la clave se modifica lo mejor es dejar al archivo secuencial tal cual y anotar en el archivo de transacciones dos registros, uno a eliminar (el original) y otro para insertar (el modificado), de manera que hasta la reorganización se consideren los cambios. Nuevamente, anotando las modificaciones en un archivo independiente simplificamos el trabajo inmediato dejando la actualización real para el momento de la reorganización del archivo.

6. Reorganización de archivo: De los apartados anteriores se desprende directamente que para la reorganización se tomarán tanto el archivo secuencial original como el archivo bitácora uniéndolos de forma coordinada hasta obtener uno nuevo. El procedimiento consta de dos fases, lectura del archivo bitácora y ordenación del mismo en memoria principal, y segundo, ciclos de lectura del archivo secuencial original alternados con escritura al nuevo. Aprovechando la tarea es buena idea asegurar la consistencia de la información, verificando el orden de claves primarias en el archivo original.

Suponiendo que no se permita a la bitácora crecer más allá del 5% del tamaño del archivo original y que el tiempo de lectura de todo el archivo sea t_l y el de escritura t_e, despreciando el tiempo consumido en ordenación, el proceso durará aproximadamente $1.05*(t_l + t_e)$.

Observaciones

Los registros de archivos secuenciales al encontrarse físicamente uno tras de otro, no facilitan la modificación de su estructura, obligando a dejar algunas "columnas" o campos vacios para futuros

cambios mismos que si acaban con las columnas libres, nos obligarán a reescribir todo el archivo para dejar el espacio necesario.

Otra limitante de la organización secuencial es la presencia de una llave lo cual exige reclasificación de archivos cuando se desea accesar la información de otra forma, o a generar nuevos archivos cuando se van a combinar datos provenientes de dos cuya clave primaria es diferente.

Aplicaciones

Tradicionalmente este tipo de archivos se usaron ampliamente y siguen usándose en el procesamiento comercial de datos. Si la información de una base es procesada cada cierto número de días, difícilmente otro tipo de organización de archivos proporcionará un mejor funcionamiento. Además esta organización suele usarse como medio estándar de transferencia entre tipos diferentes de computadoras o dispositivos.

Archivos secuenciales indexados

Estructura general

Aunque existen varias formas de implementar el acceso indexado, el método básico es a través de archivos secuenciales indexados que resultan de agregar un *índice* a los archivos secuenciales normales con lo cual, mejora notablemente el desempeño del acceso sin deteriorar las ventajas de secuencialidad que proporcionan, también se contempla la inclusión de una o varias *áreas de derrama* para manejar las adiciones y modificaciones al archivo.

La estructura de estos archivos dejando a un lado el área de derrama es muy semejante a la de un directorio telefónico; cuando queremos encontrar el número de una compañía, primero consultamos el índice buscando el número de página en donde está y luego revisamos cada nombre hasta hallar el deseado.

Los registros de archivos secuenciales identificados por su clave primaria o principal, además de ser de longitud fija, agregan en su estructura un apuntador cuya utilidad se explicará más adelante; en cuanto al archivo índice consta de anotaciones de llaves correspondientes al primer registro de cada bloque y un apuntador al bloque.

Antes de considerar la forma de acceso observemos como en este tipo de organización ya existe cierto grado de redundancia por consiguiente, las operaciones de escritura deben mantener la consistencia en las relaciones entre índice, archivo principal y área de derrama. Ahora el archivo índice debe mantenerse ordenado, tarea que es bien sencilla pues su tamaño será siempre mucho menor al del archivo principal, no solo por que la cantidad de registros disminuye a $1/B_{fi}$ sino además por que cada registro carece de atributos que no formen parte de la clave primaria.

De las consideraciones anteriores, concluimos que una organización secuencial indexada entrega su mejor rendimiento cuando es implantada en dispositivos de acceso directo, ya que

además del tradicional procesado secuencial la existencia de un índice permite el tratamiento aleatorio. Para la lectura de todos los registros, se carga el índice en memoria con objeto de recuperar uno a uno los bloques, viajando secuencialmente por los registros hasta encontrar un apuntador al área de derrama en cuyo caso la lectura debe continuar de ahí hasta un apuntador nulo que es la indicación de regreso al archivo principal.

Indexado en multinivel

En ocasiones, cuando el archivo secuencial es de dimensiones considerables o si el factor de bloqueo es bajo o bien, debido a llaves muy largas, el índice resultante no puede almacenarse en su totalidad en memoria principal y suele disponerse en archivos varios niveles de índice haciendo que cada uno apunte al siguiente hasta llegar a la información en sí, construyendo una estructura denominada *multi-indexado*. El inconveniente que representa la considerable disminución en cuanto a eficiencia de acceso, es superado con una buena organización de archivos en dispositivos de acceso directo como discos de alta velocidad.

La cantidad de niveles que se pueden formar en archivos de gran tamaño llega a producir estructuras tipo árbol, cuyas reglas de formación desembocan en diferentes características de desempeño. Dos ejemplos son los *árboles balanceados* que son fundamento del estándar *ISAM* y los *árboles-B* del *VSAM*, ambos desarrollados por IBM.

Organizaciones ISAM y VSAM

La organización ISAM (*Indexed Sequential Access Method*) se caracteriza por una indexación expresada en base a pistas y cilindros de disco, por tanto, debe implantarse a nivel sistema operativo a modo de proporcionar independencia de acceso a lenguajes de alto nivel y manejadores de base de datos; desde la creación de archivos es posible indicar el grado de densidad inicial por bloque tanto para el archivo de datos como para el índice lo cual viene dado en función directa del crecimiento esperado y, que bien aplicado disminuye en mucho el uso de áreas de sobreflujo[18].

La rapidez en el acceso a los archivos se centra en que cada pista corresponde a un bloque y al llenarse una de ellas, tomamos la pista correspondiente de otra superficie en el mismo cilindro de modo que el tiempo de localización es en muchos casos nulo. Es hasta que no hay mas cupo en el mismo cilindro que usamos el siguiente contiguo. El desempeño del archivo ya sea a la lectura o procesamiento se ve sensiblemente disminuido con el crecimiento del área de derrama, problema resuelto al efectuar reorganizaciones periódicas.

El estándar de archivos VSAM (*Virtual Sequential Access Method*) fue ideado por IBM para conjugar diferentes organizaciones de archivos, una de las ventajas obtenidas es la mejoría en el grado de independencia ya que los registros no son direccionados por hardware (como en el caso de ISAM) sino por el desplazamiento relativo a partir del origen del archivo, además, el tamaño

[18] Vid sección 1.3.2. *Arboles*

de bloque (llamado *intervalo de control*) ahora lo define el usuario sin importar las características físicas del hardware.

Ya que no es parte de los objetivos describir al detalle una estructura de archivos específica sino dar una visión general, no se profundizará más en lo tocante a estas organizaciones[19].

Desempeño

1. Tamaño de registro: El tamaño de los registros físicos ya sean de longitud fija o variable es igual al de los registros lógicos mas un byte para posibles marcas de eliminación (epitafios). Para obtener el tamaño total de la estructura de archivos (sin tomar en cuenta cabecera de archivos) hace falta considerar los índices y áreas de derrama. Suponiendo un solo nivel de índices, la siguiente ecuación da la longitud:

$$\text{tamaño} = (nR + (c + ap)(n/B_{fr}))\ (1 + E)$$

donde:

n... Número de registros
R... Longitud {promedio} de registro
E... Porcentaje de espacio libre
c... Tamaño de la clave primaria
ap.. Tamaño de los apuntadores
B_{fr}.. Factor de bloqueo

2. Recuperación de registros: Ya sea que las áreas de derrama se hallen vacías como sucede después de una reorganización, o en el otro extremo, próximas a su máxima capacidad, el tiempo de acceso a un registro aleatorio puede ser *tan* reducido como el necesario para la localización y transferencia de cualquier bloque, siempre y cuando la estructura de índice en su totalidad sea mantenida en memoria principal; de lo contrario, habrá que leer tantos bloques adicionales como niveles de índice se dejen en disco.

3. Recuperación del siguiente registro: Después de la obtención de un registro, tres situaciones se pueden presentar: primero que el siguiente registro esté en el bloque recién recuperado, que se localize en otro bloque dentro de la misma pista o cilindro o bien, que se encuentre en otro cilindro. Dependiendo de esto, los tiempos de acceso serán de cero, razón de transferencia por tamaño de bloque + tiempo de latencia, o tiempo total de acceso a bloque, respectivamente.

4. Agregado de registros: Las inserciones pueden presentarse de dos formas pero en ambas el índice tiene que consultarse para determinar el lugar que le corresponde al nuevo registro y leer el bloque respectivo, a partir de ahi, en el caso sencillo cuando no se requiere dar de alta un nuevo bloque, el registro es agregado y el bloque reescrito; de otra forma, cuando se liga un bloque libre

[19]Cfr. Tsai, Alice, *Sistemas de Bases de Datos*, p52-65

del área de derrama, aparte de dividir en dos la información del bloque actual y reescribirlo, el resto también se transfiere; la probabilidad de ocurrencia de esta situación es de $1/B_{fr}$.

5. Actualización de registros: Como en el punto anterior, aquí encontramos dos casos, en cualquiera de ellos será necesario recuperar el bloque con el registro correspondiente, ahora bien, si el o los campos llave no se alteran bastará con actualizar la información y reescribir el bloque, de modo que el tiempo ocupado vendrá en función de la razón de transferencia y del tamaño de bloque o de lo contrario, si la llave fue alterada, para evitar la pérdida del orden en el archivo aparte de reescribir el bloque con su registro marcado con un epitafio se inserta un nuevo registro.

6. Reorganización de archivos: La reorganización se lleva a cabo mediante una lectura completa del archivo y escritura de otro nuevo, carente de áreas de derrama. Por supuesto, en una segunda fase el índice es reconstruido haciendo que apunte a los bloques reasignados.

Aplicaciones

Con la ventaja que el indexado representa en este tipo de organización, el uso de archivos secuenciales fue abandonado en varias aplicaciones, principalmente dentro del campo de procesamiento de datos comerciales ya que no es necesario correr reorganizaciones para que adiciones y cambios se tomen en cuenta.

Archivos indexados

Cuando el acceso secuencial no es un requisito o actividad frecuente sino que las recuperaciones se hacen en base a peticiones aleatorias, la eficiencia de acceso mejora sacrificando la secuencialidad en el archivo, que en todo caso podrá reproducirse por medio de índices. En cuanto a la estructura propia de los registros, se usan de longitud fija, variable, etiquetados, etc.

La actividad más difícil en el manejo de estos archivos es el mantenimiento de sus índices, pues resulta común tener varios de ellos para un solo archivo, de manera que al agregar o eliminar un registro, todos los índices deben actualizar sus apuntadores ya sean del tipo de desplazamiento (el más recomendable) o de dirección hardware específica. En lo tocante al índice en si, a diferencia de los archivos secuenciales donde solo existe una anotación por bloque con la consecuente reducción de tamaño, en los archivos indexados para tener una estructura completa que apunte a toda la información, debemos mantener una anotación por registro; de ahi que al campo-clave también se le llame clave de inversión.

Existen varias aplicaciones donde un archivo rodeado de varios índices construidos en base a diferentes campos, es sometido a búsquedas sobre aquellos registros cuyos campos-clave cumplan cierta condición, trabajando la información total del archivo a través de un filtro; es recomendable para tales casos con objeto de mejorar la respuesta y disminuir los requerimientos de almacenamiento de archivos índice generar índices "filtrados", mejor conocidos como *índices selectivos*, en contraste con el método tradicional que produce *índices exhaustivos*.

Desempeño

1. Tamaño de registro: Si consideramos que 'R' es la longitud fija o promedio de los 'n' registros (recordemos que la estructura de los registros puede ser variable), 'm' el número de archivos índice, 'Cm' el tamaño de la clave de acceso para cada índice, y 'P' la cantidad de bytes necesarios para los apuntadores de índices a archivo principal, suponiendo todos los índices exhaustivos, el espacio total ocupado será:

$$n(R + mP + C_1 + C_2 + \ldots + C_m)$$

2. Recuperación de registros: La eficiencia de respuesta es la misma que la obtenida por los archivos secuenciales indexados, las mismas consideraciones son válidas; el cargar a memoria principal el índice hace aun más ágil la manipulación de los datos, ya que un solo acceso a bloque nos proporcionará la información, sin embargo el mayor tamaño de estos índices suele imposibilitar esa opción.

3. Recuperación del siguiente registro: A falta de orden serial en los datos del archivo principal, el siguiente registro vendrá dado únicamente por el índice y la probabilidad de ubicarlo en el último bloque recuperado dependerá de la cantidad de registros por bloque y del total en el archivo, por tanto, el tiempo promedio es prácticamente el mismo del punto anterior.

4. Agregado de registros: Ahora el trabajo laborioso se transfiere del archivo principal a los archivos índice, el nuevo registro puede agregarse al final del archivo de datos pero todos y cada uno de los índices exceptuando los selectivos que filtren la nueva información, deberán actualizarse, por tanto el proceso, si no se conservan los índices en memoria será tantas veces más lento como archivos índice haya; este es el precio a pagar por acceso a archivos a través de varias llaves y bien lo vale pues nuevos requerimientos de búsqueda se satisfacen fácilmente agregando un nuevo índice.

5. Actualizacion de registros: Nuevamente, el trabajo sobre el archivo principal es menor, el registro modificado (si los registros son de longitud fija) es reescrito en su misma posición, pero en la mayoría de los índices, dependiendo del grado de cambios realizados tendremos que eliminar clave y apuntador del registro modificado e insertar en el lugar adecuado una nueva anotación. En resumen, la actividad multiplica su complejidad respecto al tipo de archivos estudiado en la sección anterior. El problema más grave que llega a suscitarse es la posible falta de consistencia en la información debido a fallas de energía o errores de escritura al actualizar archivos índice, lo peor del caso es que el error puede pasar desapercibido por largo tiempo hasta corregirse con una reorganización de archivos.

6. Reorganización de archivos: La teoría indica que los archivos indexados no requieren de reorganizaciones periódicas a menos que se desee recuperar espacio perdido por registros marcados con epitafio. Pero en la práctica, esta actividad debe efectuarse con cierta regularidad especialmente en bases de datos donde no hay garantía de integridad ante fallas inesperadas del sistema, tal es el caso de la gran mayoría de los manejadores de base de datos para computadoras personales.

Aplicaciones

Dada la velocidad de respuesta, los archivos indexados se usan principalmente en aplicaciones donde la disponibilidad de información sobre uno o unos cuantos registros es prioritaria y el procesado secuencial poco común, ejemplos se encuentran en redes bancarias, sistemas de reservación, etc.

Archivos directos

Como su nombre lo indica, este tipo de archivos hace uso de direcciones directas para referirse a registros específicos, se trata pues de un concepto diferente a los planteados en organizaciones anteriores, fue especialmente diseñado para dispositivos de almacenamiento secundario con prestaciones de acceso aleatorio.

Para obtener una dirección relativa de desplazamiento dentro del archivo se toma como base algún atributo; el atributo seleccionado se hace pasar por un algoritmo de traslación que puede consistir de un mapeo uno a uno entre los valores posibles de la llave y el espacio físico en el archivo, (a tales algoritmos que se les conoce como *procedimientos determinísticos*) o por medio de *técnicas de aleatorización* que dan un mapeo varios a uno, cuya utilidad se refleja principalmente en ahorro de espacio en disco.

** (Algunos autores llaman a los archivos construidos con procedimientos determinísticos *archivos directos* y a los generados a partir de técnicas aleatorizantes *archivos dispersos*.)

Para diseñar un método de transformación de llave a dirección (llamados también métodos "hash"[20]), el primer paso es seleccionar el o los atributos que formarán la llave cuyo universo de valores respectivo debe conocerse plenamente. Posteriormente analizamos el grado de correspondencia entre el universo teórico y el que en la práctica dominará el archivo; dependiendo de la diferencia en tamaños resultante y en general de los requerimientos del sistema debemos escoger entre un procedimiento determinístico o alguna técnica de aleatorización.

En general se obtiene una función de mapeo "hash" de la forma H: K-> Y donde el dominio de la función (K) equivale al conjunto de valores posibles de la llave, y el codominio (Y) son las direcciones de almacenamiento o desplazamientos relativos en forma de números enteros. Los procedimientos determinísticos son los mas sencillos de implementar; a partir del universo teórico se encuentra una función o algoritmo para obtener direcciones secuenciales de archivo únicas.

Trabajar con técnicas aleatorias exige un estudio previo sobre la cantidad máxima (n) de registros que habrá en el archivo, a este valor se agrega un porcentaje considerable para que prevenga frecuencia alta de colisiones y disminuya el número de intentos al buscar lugar para registros colisionados, resultando en un tamaño de archivo (m). Elegir la función adecuada implica ajustarse a la gráfica de distribución de llaves esperada; cuando no tomamos en cuenta la distribución debido a que todos los valores de llave tiene igual probabilidad de ocurrencia, se

---

[20]Euan, J. L., *Estructuras de Datos*, p186

recurre frecuentemente al método "hash" de conversión de base en el cual la operación módulo[21] es aplicada entre un valor numérico secuencial (calculado a partir de la llave) y m/B_{fr}, obteniendo así un diseño orientado a hardware que nos proporciona número de bloque, unidad de acceso mínima al dispositivo.

Como en las funciones de transformación aleatoria de la forma H: K-> Y distintos elementos de "K" no son necesariamente mapeados en distintos elementos de "Y", es decir que tales funciones no son biyectivas, surge el problema de colisiones con sinónimos cuando con objeto de recuperar o almacenar un dato se obtiene la dirección de un registro ya ocupado con información, es entonces que debemos encontrar espacio disponible y de las estrategias a seguir para la solución de colisiones, la literatura al respecto reconoce tres métodos básicos:

Búsqueda lineal. Método también conocido como "direccionamiento abierto" es el más empleado por su sencillez y velocidad; cuando ocurre una derrama en un bloque (si al buscar un registro no se encuentra o si al intentar escribirlo ya no hay espacio en ese bloque) el bloque físicamente contiguo al último accesado se recupera (o si se trata del último bloque del archivo se recupera el primero) intentando la actividad deseada originalmente en el. Este método, mantiene tiempos de recuperación bajos aún en registros "derramados".

Realeatorización. Como su nombre lo indica cuando hay una colisión, la llave original es sometida a otro algoritmo de aleatorización o al mismo pero cambiando algunos datos. Como puede preveerse, conforme la densidad o factor de carga del archivo aumenta, mayor número de veces tendrá que realeatorizarse, lo que puede resultar impráctico aun cuando el archivo esté almacenado en unidades de disco de bajo tiempo de acceso; por eso su principal aplicación es en archivos cargados en memoria principal donde ante todo debe ahorrarse espacio.

Archivo de derrama. En este último método, las derramas de bloque se agregan secuencialmente a un archivo de derrama independiente o bien, al final del archivo principal, los registros colocados en esta área tienen la estructura que una lista ligada y por eso el método también es conocido como "resolución por encadenamiento".

Una técnica llamada aleatorización variable[22] combina los métodos de archivo de derrama y realeatorización permitiendo el crecimiento dinámico del archivo de datos al ligar nuevos bloques intermedios que son reconocidos por el algoritmo de aleatorización.

Antes de implantar definitivamente un método es fundamental hacer pruebas anotando la cantidad de colisiones y la eficiencia en el tratamiento de las mismas, dichas pruebas pueden llevarse a cabo sobre entradas reales o con un programa de cómputo de simulación.

---

[21]Campderrich B., *Técnicas de Bases de Datos*, p123

[22]Campderrich B., *Técnicas de Bases de Datos*, p129-133

Desempeño

1. Tamaño de registro: El tamaño de registro es por lo general fijo o, variable dentro de un rango pequeño, en cuanto al archivo, su tamaño será determinado principalmente por el grado de densidad que el programador haya elegido.

2. Recuperación de registros: El tiempo de recuperación despreciando el consumido por el procesador en la conversión de la llave a dirección es muy bajo, tanto, que con áreas de derrama de bloque sin llenar basta con recuperar un bloque.

3. Recuperación del siguiente registro: Ya que el concepto de secuencialidad se pierde en este tipo de archivos, no podemos hablar de recuperación del siguiente, a menos que por algún otro medio que no sea a partir de los datos almacenados se conozca la llave siguiente en cuyo caso el tiempo será igual al obtenido en el punto anterior.

4. Agregado de registros: Cuando la cantidad de actualizaciones esperadas es baja y se tiene un programa de reorganizaciones periódicas, el tiempo de agregado puede ser tan reducido como la recuperación y reescritura de un bloque a disco; en general este parámetro se verá seriamente afectado por desbordamientos en las áreas de derrama.

5. Actualización de registros: Como en el caso de los archivos secuenciales indexados, se presentan dos situaciones, si el nuevo registro (de longitud fija) no es modificado en su llave principal, basta con reescribir el bloque con la nueva información, de otra forma, la alteración de la llave obliga a eliminar el registro, reescribir el bloque modificado y seguir el procedimiento descrito para agregado de registros, de modo que en el mejor de los casos podemos esperar dos accesos a disco y en el peor, cuatro.

6. Reorganizacion de archivos: Cuando el número de registros esperados aumenta (n se aproxima a m), o si las áreas de derrama crecen demasiado o desuniformemente, es obligatorio cambiar la relación n:m y además revisar y adecuar el algoritmo de transformación.

Aplicaciones

Tomando en cuenta como ventaja el que no exista índice y que por consiguiente en la mayoría de los casos el registro buscado se encuentra en el primer acceso a disco, y como desventaja la pobre eficiencia de acceso secuencial, este tipo de archivos es empleado en aplicaciones donde la actualización no es actividad frecuente para evitar el crecimiento de áreas de derrama, aplicaciones en las que el acceso rápido es indispensable y las búsquedas sencillas; a partir de una sola llave. Ejemplos prácticos son: listas de precios, directorios telefónicos, tablas, inventarios, etc.

Archivos multianillo

Esta organización se caracteriza por un manejo de la información en paquetes; las búsquedas en lugar de recuperar un registro por petición, "filtrarán" el universo de datos seleccionando aquellos que cumplan ciertas condiciones. Los registros se agrupan formando *subconjuntos* caracterizados por que todos sus elementos cuentan con una clase de atributo en común. Dicho agrupamiento es diseñado a modo de optimizar las consultas y se consigue ya sea por apuntadores o por simple proximidad física.

La organización multianillo tiene su origen en las estructuras de datos denominadas listas ligadas pero ha evolucionado hasta establecer interconexiones jerárquicas entre varias de ellas (*anillos*); el archivo en conjunto es resultado de un profundo análisis de requerimientos y queda íntimamente ligado a la aplicación para la que fué diseñado. Gráficamente y a modo de ejemplo, una compañía con varias sucursales y empleados en cada una podría representar esa información de la siguiente forma:

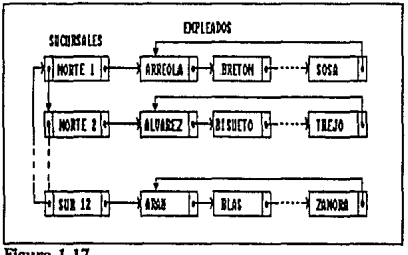


Figura 1.17
Ejemplo de organización multianillo

Como la estructura propia de los registros depende a que anillo en particular pertenecen, los archivos resultantes son heterogéneos; cada registro tendrá los campos que corresponden a los anillos a los que pertenece. Sin embargo, el número de combinaciones es limitado y por ello la separación entre campos no se hace con el método de etiquetado sino mediante un identificador de categoría de registro. En general los registros tienen el siguiente formato:

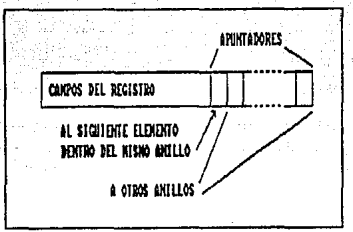


Figura 1.18
Estructura general de los registros en un archivo multianillo

Como los registros no reunen físicamente los atributos con rangos de valores reducidos (en lugar de eso se usan apuntadores a otros anillos), los archivos son mas compactos, carecen prácticamente de redundancia y presentan buena eficiencia de acceso si conglomeramos los datos de cada anillo en un solo cilindro. Por otro lado, es inevitable el deterioro a la eficiencia cuando un proceso intenta recuperar atributos ubicados en diferentes anillos pertenecientes a un mismo registro, ya que la cantidad de accesos a disco consultando diferentes listas puede ser bastante grande.

Desempeño

1. Tamaño de registro: Dado que en un mismo archivo existen varios "tipos" de registro, es necesario recurrir a un promedio para encontrar la longitud media, con lo cual definimos el factor de bloqueo y determinamos la dimensión de bloque más adecuada mediante el cálculo del tamaño correspondiente a cada anillo.

2. Recuperación de registros: En general como la estructura entre anillos es de tipo jerárquico, los campos solicitados con mayor frecuencia deben quedar en los primeros niveles de acceso para facilitar el proceso de recuperación que es lanzado por una petición de la siguiente forma: "Registro cuyos atributos a1, a2, a3,... sean x1, x2, x3,...". El algoritmo de recuperación como primer actividad ubica la cabecera de entrada al anillo con mayor jerarquía, luego suponiendo que en tal lista está el atributo a1, mediante búsqueda lineal ligada visita cada nodo hasta encontrar el que tenga valor x1 que a su vez será usado como cabecera en la búsqueda a través del siguiente anillo y asi sucesivamente.

Es laborioso calcular el tiempo promedio de respuesta a las peticiones de búsqueda, diferentes diseños para una misma aplicación proporcionarán tiempos muy dispares, aproximadamente, si conocemos el peso porcentual 'P_n' correspondiente al número de solicitudes de cada tipo de búsqueda, y tiempo consumido 't_n' en la transferencia de la mitad de los nodos de cada anillo que forma la jerarquía total a recorrer, el tiempo promedio será:

$$P_1t_1 + P_2t_2 + P_3t_3 + \ldots$$

3. Recuperación del siguiente registro: Aquí la idea de "el siguiente registro" va ligada con el anillo sobre el que se está trabajando, tal como sucede en archivos multi-indexados, el esfuerzo de recuperación pocas veces implicará transferencia de nuevos bloques y por tanto será mínimo e igual al de archivos secuenciales, siempre y cuando el diseño haya contemplado conglomeración de registros; de otra forma debe esperarse la petición de un bloque aleatorio.

4. Agregado de registros: El proceso de inserción en esta organización es tan costoso como en las indexadas múltiples; cada anillo es visitado hasta encontrar el lugar que le corresponde al nuevo registro, entonces se rompe la liga y se hace apuntar a un área de derrama previamente dispuesta donde la nueva información es almacenada.

5. Actualización de registros: Existen dos tipos de actualización; en el primero si no afectamos el atributo que regula el ordenamiento de los elementos del nodo la operación se consigue recuperando el registro y visitando cada anillo con objeto de actualizar sus apuntadores. Alterar el atributo-llave obliga a marcar con un epitafio al registro original y seguir el algoritmo de inserción para la versión modificada.

6. Reorganización de archivos: Esta operación es necesaria cuando las áreas de derrama crecen demasiado y obligatoria cuando los requerimientos cambian orillándonos a reestructurar las interconexiones entre anillos, para llevarla a cabo se leen secuencialmente los nodos de cada anillo (incluso los "derramados") y un archivo nuevo es producido.

Aplicaciones

Tomando en cuenta la dificultad para crear nuevos anillos conectados con los ya existentes y su estructura altamente modular, estas organizaciones han sido usadas para contener grandes bases de datos en las que los tipos de consulta están bien definidos, desde sistemas de información a nivel gerencial hasta representación de datos geográficos y arquitectónicos.

1.3.5. Indices

Los índices, archivos auxiliares cuya función[23] es permitir el acceso a los registros de un fichero principal a través del valor de uno o mas datos, se forman mediante conjuntos de anotaciones ordenadas una por cada registro-dato (en índices exahustivos) del archivo principal, dichas anotaciones contienen el valor de un atributo llave y un apuntador hacia el registro[24], de modo que aunque los registros sean grandes, las anotaciones serán considerablemente menores y de igual forma el archivo índice también será menor esto, agregado a que las anotaciones se mantienen en orden, agiliza de manera muy importante las búsquedas.

---

[23]Campderrich B., *Técnicas de Bases de Datos*, p92

[24]Wiederhold, Gio, *Diseño de Bases de Datos*, p113

Si el archivo principal es demasiado extenso provocando que el índice también lo sea, este último puede conformarse en una estructura multinivel tipo árbol, en la que al nodo raíz se le conoce como índice primario. Las anotaciones de primer nivel de acuerdo a lo que apuntan se dividen en *índices de cilindro* y en *anclas de bloque*.

Los índices tipo ancla de bloque, considerando que la menor unidad de acceso es precisamente el bloque, proporcionan acceso rápido ya que contienen la anotación correspondiente al primer registro de cada bloque. La ventaja es que el archivo índice resulta de menor tamaño que un índice exhaustivo pero, la eficiencia de acceso se reduce ya que para saber si existe el registro buscado aparte del índice, tenemos que leer el bloque respectivo del archivo.

En general para diseñar un índice es útil determinar la *razón de abanico de salida* "y" que nos da la cantidad de bloques de información del archivo principal a los que un bloque índice puede apuntar, para calcularla basta con conocer el tamaño de bloque (B), de la llave (V) y del apuntador (P), la expresión matemática del abanico es:

```
y= B/(V + P)
```

Una vez calculado este parámetro, dividimos la cantidad de registros esperados en el archivo principal entre el factor de bloqueo, lo que nos da cuantos bloques a lo más formarán el índice y será al mismo tiempo el total de anotaciones necesarias si es de un solo nivel. Ahora bien, si el tamaño es excesivo como para mantenerse en memoria y facilitar la localización de los registros, entonces divisiones posteriores serán deseables y cada una hará aparecer nuevos niveles en la gráfica del archivo, a la cantidad de niveles de un árbol se le conoce como *altura de árbol*. Este concepto lleva a dos filosofías; dejar la altura del árbol fija y permitir aumento ilimitado en número de entradas (aplicado en archivos secuenciales indexados) o bien, proceder al contrario.

El diseño de índices del tipo árbol-B se puede realizar como quedó explicado en el párrafo anterior, pero el abanico de salida (y) debe afectarse por un factor porcentual (comúnmente el 50%) para dejar espacio a las inserciones de nuevas claves en cada bloque. Otra consideración de flexibilidad en el proceso de diseño que disminuye sensiblemente el espacio ocupado por el índice es la abreviación de llaves; por ejemplo, tratándose de una llave tipo entero de 8 dígitos los primeros dos se pueden incluir en las anotaciones del índice primario, los siguientes tres en las de un nivel intermedio y los últimos tres dejarlos para el nivel que apunta directamente a datos. Así, tendríamos *abreviatura de orden superior* (los dígitos a la derecha) y *abreviatura de orden inferior* (los dígitos a la izquierda).

En ciertas aplicaciones donde la prestancia de la información no es prioritaria, la actualización de los índices de archivos múltiplemente indexados se hace en momentos de baja o nula demanda al sistema por parte de usuarios; es común en aplicaciones administrativas que al dar de alta una cuenta o un empleado nuevo su registro en computadora sea reconocido hasta unos días después del ingreso de la solicitud.

1.4. MODELADO

1.4.1. Proceso de diseño

Las bases de datos pequeñas, de poca importancia funcional al igual que los programas de este tipo se acostumbran diseñar intuitivamente, a partir de un sencillo diagrama en papel y considerando experiencias previas; pero los sistemas complejos deben seguir una secuencia lógica en su diseño. Por *diseño* debemos entender que es aquel proceso creativo que planea o arregla las partes de un todo para satisfacer los objetivos implícitos en el mismo[25].

Aunque no existe un método estandarizado, los procesos de diseño de una base de datos expuestos por diferentes autores tienen en común varios aspectos; tomando estos enfoques, podemos proponer el siguiente:

El diseño de una aplicación cualquiera de software, de un módulo específico de la misma o de una base de datos empieza por el análisis de las necesidades que se deben satisfacer sean estas de los usuarios o de programas de cómputo. Durante esta fase no consideramos eficiencia en el desempeño; es decir que solo se determinan entidades y atributos sin tomar en cuenta su importancia o nivel de disponibilidad, además debe cuidarse que la estructura refleje fielmente la realidad que representa.

Asi hablando específicamente de una base de datos, el primer paso, en especial si se desconoce el sistema al que prestará servicios, es entrar en contacto con las personas que nos puedan proporcionar y describir los campos o entidades dato necesarios desde el punto de vista del usuario; consideraciones que sirven para definir formalmente los datos y sus relaciones.

Con estas definiciones básicas se empieza a armar el *diccionario de datos* del cual parte es el listado de los datos requeridos por cada aplicación, esta lista deriva en el compendio de nombres de atributos que en sistemas grandes conviene agrupar en apartados dentro del diccionario para facilitar su manejo.

A partir del conjunto de atributos y aplicando procedimientos de normalización obtenemos el refinamiento de la estructura de la base a nivel conceptual; considerando los tipos básicos de modelos de datos en la siguiente fase se transforma el modelo conceptual en modelo lógico (tema que abordaremos en la sección 1.5); posteriormente a partir del diccionario de datos preparamos un *esquema* que es puesto directamente a disposición de un DBMS o, si vamos a usar un lenguaje de alto nivel nos servirá para codificar la estructura de la base, obteniendo asi su modelo físico.

Aun con todas las previsiones la base de datos no debe instalarse en el medio definitivo sin antes realizar pruebas con un prototipo de lo que será el modelo físico final, asegurándose que los requerimientos hayan sido satisfechos.

---

[25] Fitzgerald, *Análisis de Sistemas*, p547

1.4.2. Modelado conceptual

Definición y objetivos del modelado conceptual

El modelo conceptual de una base de datos divide en bloques el problema de diseño[26], mostrando el flujo de la información relacionada con el sistema; en el quedan representadas las *entidades* por relaciones entre campos, y debe contemplar necesidades actuales y previsibles.

Para comprender la definición anterior debemos aclarar que *entidad*[27] es aquel objeto, persona, evento, etc; tangible o intangible sobre el cual se registra cierta información. La descripción de las entidades la hacemos dando *valor* a sus *atributos*. Es importante diferenciar estos dos últimos conceptos; por ejemplo, si tenemos un formato llamado "Datos Personales" en el que llenamos los renglones de NOMBRE, DIRECCION, EDAD, (atributos), etc. la información que escribimos son *valores de atributo*. A partir de lo anterior, podemos retomar el concepto de registro expresándolo en términos de estas definiciones como "La reunión de valores dato de atributos agrupados en una estructura lineal referentes a una o más entidades".

El modelo conceptual es una descripción global de la base de datos resultado de la integración de los requerimientos conceptuales de usuarios y procesos, y como dichos requerimientos quedan definidos en base a archivos conceptuales formados por registros lógicos, los detalles sobre almacenamiento relacionados con los registros físicos no importan, siendo inadecuado en este punto intentar determinar el DBMS que se requerirá, incluso las características del modelo conceptual lo hacen independiente hasta del hardware mismo y del tipo de base de datos a usar.

Pasos a seguir para obtener el modelo conceptual

Para diseñar un modelo conceptual iniciamos cuestionando a los usarios sobre sus necesidades prácticas tanto de recuperación como de almacenamiento, tanto actuales como futuras; previendo el surgimiento de nuevos requerimientos cuando el sistema de información quede implantado, adentrándose en el procesamiento y flujo de información e incluso llevando a cabo un cuestionario para recabar información sobre los datos desde el punto de vista del usuario, estos conceptos formarán parte del diccionario de datos:

- Entidades del sistema, atributos correspondientes
- Nombre y sinónimo de cada campo, adquiridos y calculados
- Descripción de campos
- Fuente de la que se obtiene el valor de los atributos
- Utilidad y área en la que se ocupa cada campo
- Recopilar rango, tipo y unidades de cada campo

---

[26]Wiederhold, Gio, *Diseño de Bases de Datos*, p410

[27]Martin, James, *Principles of Data Base Management*, p19

- Grado de alterabilidad de las características antes mencionadas
- Restricciones de acceso a la información
- Relaciones entre campos

Una técnica de modelado propuesta en varios textos sugiere la normalización de un modelo relacional prototipo construido en base a los conceptos registrados; vinculando algunos datos representativos del universo total se forman *eneadas* cuya agrupación produce archivos conceptuales en forma de *tablas* (en algunos textos se llama a los renglones de dichas tablas *tuplas*[28]), estableciendo *relaciones de dependencia* entre las columnas. Por último sometemos las tablas al proceso de normalización, obteniendo al final un modelo conceptual que fácilmente es transportado al lógico de tipo relacional.

Existen otros métodos de modelado como el de Abrial[29] y el de *Entidad - Relación* que con ligeras adaptaciones en cuanto a su simbología será el que ocupemos para el diseño. Fue elegido por que representa claramente y sin implicaciones de archivos físicos la estructura de cualquier sistema real. En las siguientes secciones abordaremos primero la simbología usada y después el proceso de normalización sobre este modelo.

Cuando una base de datos tiene más de una finalidad, el modelado conceptual es fraccionado en *modelos de visión* que asi como pueden desembocar en bases de datos independientes pueden también, si los enlaces lo ameritan integrarse en una sola; la importancia de los enfoques radica en que dos modelos de visión tomados independientemente para formar un modelo lógico llevarían a un mismo diseñador a diferentes resultados, pues aunque ambas visiones compartan los mismos campos, para satisfacer el enfoque requerido por cada una es necesario establecer diferentes vínculos.

De igual forma, ya desde el modelado conceptual debemos esquematizar la distribución de la base de datos; por ejemplo, el manejo por bases independientes es útil para controlar sistemas con restricciones de conexión entre sus bases de datos, disminuye el tráfico de información a través de la red, contempla cierta redundancia en el modelado y delega la responsabilidad de mantener la consistencia a procedimientos de actualización. Un sistema donde la prestancia de información es prioritaria como el caso de los sistemas de tiempo real, constituido por varios nodos que tienen derecho de acceso al universo de datos presentará serios problemas si los procedimientos de actualización y consulta no responden a la velocidad requerida, de ahí que llegue a ser válido no solo duplicar en cada localidad el modelo global de base de datos[30] sino incluso repetir por completo la base haciendo que todos los nodos escuchen las actualizaciones.

---

[28]Date, *Introducción a los Sistemas de Base de Datos*, p72

[29]Campderrich B., *Técnicas de Bases de Datos*, p69

[30]Wiederhold, Gio, *Diseño de Bases de Datos*, p462

Relaciones entre los datos

Dentro de los conceptos inherentes al modelado de bases de datos están los de *vínculo* y *relación*, ambos tratan sobre los enlaces que de alguna manera existen entre dos conjuntos de datos. Ahora bien, el vínculo denota la existencia de un enlace entre elementos de conjuntos mientras que la relación sirve para indicar enlace entre dos conjuntos.

Aplicando el concepto a una base de datos los vínculos se muestran entre valores de campos y registros mientras que las relaciones de manera más global lo hacen entre atributos y entidades, cabe señalar que en la mayoría de los textos sobre diseño de bases de datos la palabra "relación" es usada para referirse a archivos conceptuales de modelos relacionales, nosotros para evitar confusiones emplearemos el término "tabla". En general, los enlaces entre datos ya sean del tipo vínculo o relación, presentan tres posibles variantes; uno a uno, uno a varios y varios a varios:

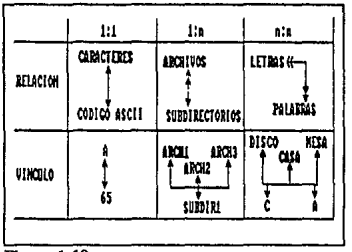


Figura 1.19
Vínculos y relaciones

Vínculos

A través de vínculos construimos archivos conceptuales ya que estos son el modo básico de enlace entre elementos dato, siendo el más simple el vínculo binario que relaciona el valor de un atributo con su objeto. Conjuntos de este tipo de vínculos nos llevan a los vínculos múltiples que son la base de construcción de las *eneadas* concepto definido como la agrupación de elementos dato primarios heterogéneos relacionados lógicamente. La conjunción de eneadas semejantes produce archivos conceptuales que por su forma denominamos *tablas*, en ellas cada renglón corresponde a una eneada y en cada columna se registran campos correspondientes a un mismo atributo.

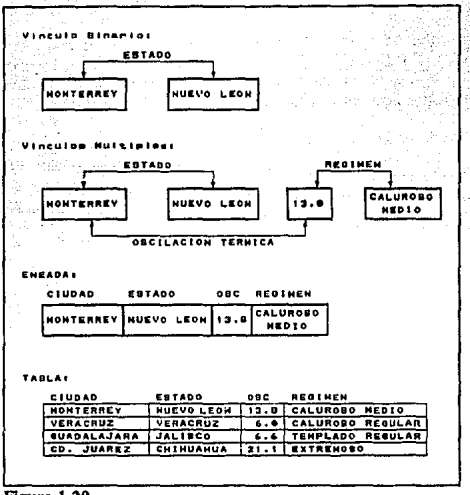


Figura 1.20
Vínculos, eneada y tabla

En la transformación de vínculos a eneada y de ahi a tabla el nombre de cada vínculo pasa a ser la identificación del atributo en la tabla; además los campos que corresponden al objeto de cada vínculo binario se denominan *parte rectora* pues identifican cada ocurrencia del objeto, los atributos restantes se llaman *parte dependiente*

Relaciones

Aunque por medio de vínculos podemos esquematizar un modelo conceptual, las relaciones nos ofrecen un nivel de claridad superior ya que por definición una relación manifiesta el enlace entre dos *conjuntos de datos*.

En cuanto a las clases de relaciones, por la cantidad de elementos que enlazan, tenemos tres variantes:

1) *Uno a uno*: Es el enlace entre dos entidades en ambos sentidos donde a cada ocurrencia de una corresponde una y solo una de la otra, estableciéndose una relación biunívoca.

2) *Uno a varios* (También conocida como *uno a muchos*): En este caso se permiten enlaces múltiples entre cada encada de una de las entidades y varias entidades de la otra.

3) *Varios a varios* (También llamada *muchos a muchos*): Es una relación igual que la anterior pero en ambos sentidos; cabe aclarar que aunque conceptualmente existe este tipo de relación, físicamente no se puede establecer entre dos archivos. (Posteriormente veremos que la relación varios a varios se puede sustituir por dos del tipo uno a varios, agregando una entidad conectora)

Todos los autores están de acuerdo con esta clasificación pero algunos de ellos en base a otras características elaboran otras tipificaciones, en nuestro caso gracias a la modificación del modelo Entidad-Relación (E-R) que será considerado en la siguiente sección solo requerimos introducir la siguiente diferenciación:

Dependencias funcionales

Un tipo especial de relación, la *dependencia funcional*, encontrada comúnmente entre dos conjuntos de atributos de una misma entidad establece que los elementos del segundo conjunto dependen directamente del valor tomado por los del primero, a semejanza de una aplicación entre conjuntos o de una función matemática; por ejemplo, dada y = f(x), decimos que el valor de "y" depende del que tenga "x", igualmente, si los valores de un grupo de atributos Y1,Y2,...,Yn están determinados por los de otro grupo X1,X2,...,Xn, la dependencia queda expresada como sigue:

```
DF(X1,X2,...,Xn)= Y1,Y2,...,Yn
```

En términos de bases de datos, al conjunto de atributos que gobierna o determina se le llama parte rectora y al otro parte dependiente, estableciéndose la relación RECTORA ----> DEPENDIENTE. Por ejemplo la entidad "Empleado" puede tener como parte rectora "No._empleado" y como partes dependientes "Nombre", "Dirección", "Puesto", etc.

Dentro de las dependencias funcionales tenemos cuatro tipificaciones:

1) *Dependencia Funcional Multivaluada:* Puede darse el caso que a cada ocurrencia del conjunto rector le corresponda un subconjunto de valores de la parte dependiente (por ejemplo, la relación entre el conjunto de "empleados de una empresa" y el conjunto de "hijos de los empleados", ya que un empleado puede tener cero, uno o más hijos); semejantes relaciones presentan diversos problemas en el desempeño que se resuelven a través del proceso de normalización y por esa misma razón, deben identificarse e indicarse en el diagrama conceptual. Para expresar matemáticamente esta dependencia escribimos:

```
DMV(X1,X2,...,Xm)= Y1,Y2,...,Yn
```

2) *Dependencia Funcional Total:* Se dice que un elemento dependiente (uno o varios atributos) presenta este tipo de dependencia cuando al quitar cualquier elemento de la parte rectora perdemos la dependencia funcional; es decir, que el conjunto rector no está sobredefinido y todos sus atributos son necesarios para identificar al elemento dependiente en cuestión. Las expresiones correspondientes a esta dependencia según el caso son:

```
DF[X1,X2,...,Xm]=  Y1,Y2,...,Yn
DMV[X1,X2,...,Xm]= Y1,Y2,...,Yn
```

** Obsérvese la diferencia entre paréntesis () y corchetes []: los paréntesis denotarán simplemente dependencia funcional mientras que los corchetes cuyo uso estamos introduciendo indicarán dependencia funcional total.

3) *Dependencia Funcional Transitiva:* Dada la relación de dependencia funcional:

```
DF[X1,X2,...Xm]= Y1,Y2,...,Yn
```

Cualquier otra dependencia que caiga dentro de las siguientes categorías es considerada *transitiva:*

```
1) DF[Ya,Yb,...,Yc]= Yi,Yj,...,Yk
2) DF[Ya,Yb,...,Yc]= Xi,Xj,...,Xk
3) DF[Xa,Xb,...,Xc]= Xi,Xj,...,Xk
```

** Las consideraciones anteriores se cumplen también para dependencias multivaluadas.

Hay que reconocer que del tipo (3) no existe ninguna referencia en los textos consultados; sin embargo, a juicio personal me parece indispensable considerarlo en el modelo conceptual; por ejemplo, supongamos que la parte rectora de la una entidad identifica las variables de una aplicación en tiempo real con los campos SISTEMA, SUBSISTEMA, NO_VARIABLE; y su parte dependiente consiste de VALOR_ADQ, CRIT_INF, CRIT_SUP, PREC_INF y PREC_SUP. Si además, solo existen ciertos SUBSISTEMAS válidos para cada SISTEMA, la relación uno a varios entre SISTEMA y SUBSISTEMA no puede dejarse a un lado:

```
DF[SISTEMA,SUBSISTEMA,NO_VARIABLE]=
     VALOR_ADQ,CRIT_INF,CRIT_SUP,PREC_INF,PREC_SUP

DMV[SISTEMA]= SUBSISTEMA
```

4) *Dependencia Funcional Doble (Léxico):*Es un tipo de dependencia transitiva muy especial en el que dos grupos de atributos pueden igualmente ser la parte rectora de una entidad, existiendo por razones obvias correspondencia biunívoca entre ellos.

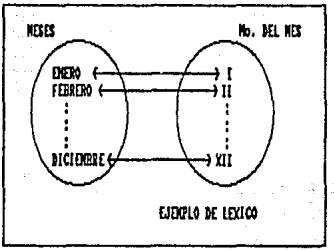


Figura 1.21
Lexico

Matemáticamente, tenemos que:

```
DF[X1,X2,...,Xm]= Y1,Y2,...,Yn          y,
DF[Y1,Y2,...,Yn]= X1,X2,...,Xm
```

Ahora, si las X's son la parte rectora, expresaremos la dependencia de la siguiente forma:

```
LEX[X1,X2,...,Xm]= Y1,Y2,...,Yn
```

Simbología

Los modelos E-R (entidad - relación) se construyen con simbolismos de entidad y relaciones etiquetadas representados de la siguiente forma:

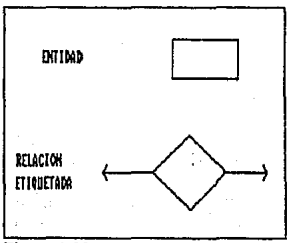


Figura 1.22
Modelos E-R: Elementos gráficos

Sin embargo, el diagramado tradicional adolece de varias deficiencias, entre las cuales tenemos: carencia de un elemento gráfico para representar agrupamientos de entidades afines, confusión provocada al representar modelos conceptuales con muchas entidades relacionadas entre si, ya que no hay varios niveles gráficos de representación de la misma información sino que estamos restringidos a una visión "plana", sin jerarquía, de la realidad. Es por eso que características propias de las hipergráficas ("hypergraphs"[31]) consiguiendo modularidad, ortogonalidad y presencia de varios niveles de abstracción.

Primero, introduciremos algunos conceptos relacionados con las hipergráficas; para empezar, son de gran semejanza con los conjuntos ordinarios pero en lugar de usar curvas sin restricción de forma los conjuntos se limitan con rectángulos curveados; otra diferencia la encontramos en cuanto al cruce de dos curvas, si en los diagramas de Venn esto representa la intersección de conjuntos, en nuestro caso no significa nada a menos que dentro del área de intersección quede encerrado un rectángulo completamente; es decir, los únicos conjuntos válidos son los encerrados por rectángulos.

En la Figura 1.23 mostramos algunos ejemplos de esta notación; la intersección entre A y B no implica existencia de algún elemento en común, pero en la intersección entre B y C si, ya que tenemos a D; en cuanto a E podemos decir que es resultado de unir A y B.

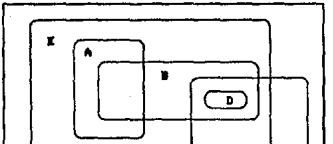


Figura 1.23
Hipergráficas

Las hipergráficas facilitan la presentación de un modelo en diferentes niveles de abstracción; por ejemplo, en la figura anterior, "D" podría contener otros conjuntos en su interior aunque en este nivel solo la representemos asi. Por si fuera poco es factible representar el producto cartesiano entre uno o varios conjuntos mediante líneas entrecortadas dentro de un conjunto:

[31] Harel, David, *Communications of the ACM*, p514

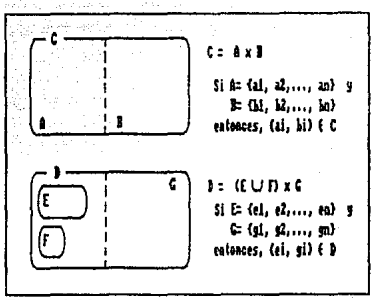


Figura 1.24
Ortogonalidad: Producto cartesiano

En el diagramado conceptual además de la notación expuesta en la sección anterior, se usarán las siguientes convenciones.:

1) Los atributos de cada entidad se representarán aprovechando el simbolismo de producto cartesiano. De manera textual, las referencias a los atributos de cada entidad se podrán hacer con el formato (Entidad.Atributo).

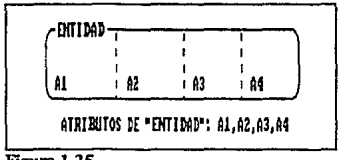


Figura 1.25
Representación de entidades y sus atributos

2) El tipo de dependencia funcional por omisión entre la parte rectora y la dependiente será siempre total y se indicará con barras o líneas inclinadas 45 grados antes del operador de producto cartesiano según sea dependencia simple o multivaluada respectivamente.

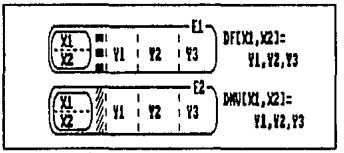


Figura 1.26
Dependencias simple y multivaluada

3) Las excepciones a la dependencia funcional indicada con el método anterior, incluyendo dependencias transitivas, se marcarán explícitamente con grafos dirigidos; es decir, líneas con una cabeza de flecha, el tipo de flecha denotará si es dependencia simple o multivaluada.

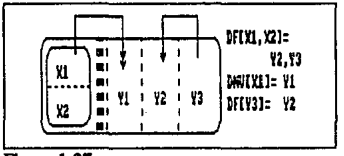


Figura 1.27
Dependencias transitivas

4) Las relaciones entre entidades se representan con grafos doblemente dirigidos indicando en el tipo de cabeza de flecha si son uno o varios los elementos que relacionan, las líneas se etiquetan con identificaciones encerradas dentro de óvalos y como habíamos mencionado, tienen tres posibles variantes.

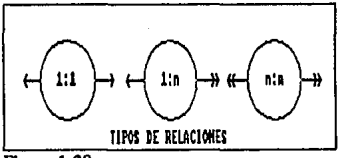


Figura 1.28
Relaciones

5) Las relaciones entre dos entidades se especificarán con flechas tocando el contorno de sus rectángulos respectivos.

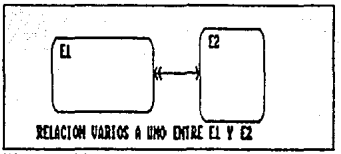


Figura 1.29
Ejemplo de relación (5)

6) En algunos casos, para mayor claridad se dibujarán las cabezas de flecha entrando a la entidad con objeto de indicar el atributo específico que relaciona a las entidades.

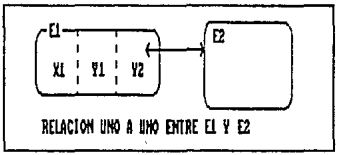


Figura 1.30
Ejemplo de relación (6)

Con el único proposito de mostrar la ventaja de esta simbología compárense los esquemas conceptuales presentados a continuación; uno se construyó con la notación E-R tradicional y el otro con las reglas anteriormente expuestas.

Ambos modelan de manera muy informal los siguientes datos relacionados con un ambiente universitario:

1) La comunidad universitaria se forma por: estudiantes, profesores, personal de mantenimiento y administrativo.

2) Los estudiantes pagan cuotas y los demás miembros de la comunidad reciben salarios.

3) La universidad ofrece cursos impartidos en salones y talleres por profesores; los cursos se dan en diferentes horarios y a ellos asisten estudiantes.

4) El personal administrativo presta sus servicios en beneficio de diferentes áreas de la comunidad. El personal de mantenimiento se ocupa del cuidado de las instalaciones.

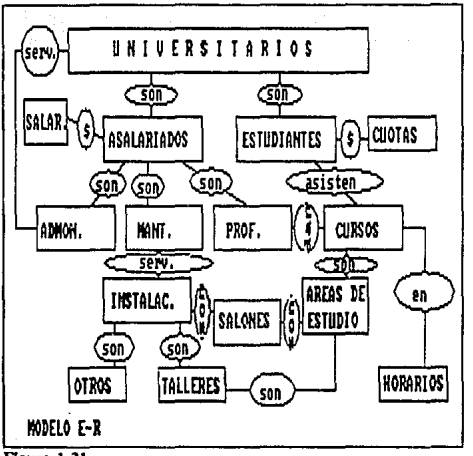


Figura 1.31
Universidad: Modelo E-R

Como podemos ver, a pesar de la simplificación impuesta al sistema, el modelo E-R no ofrece una interpretación sencilla; las relaciones tuvieron que etiquetarse y varias indican pertenencia (las que tienen la palabra "son"). Aplicando nuestra simbología obtenemos un esquema más claro incluso sin vernos obligados a etiquetar relaciones.

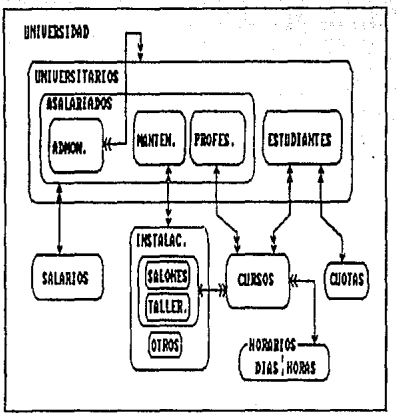


Figura 1.32
Universidad: Modelo con hipergráficas

Normalización

El modelo de base de datos obtenido por la aplicación de relaciones entre elementos-dato puede usarse como fundamento para el modelado lógico y físico, desafortunadamente es muy probable que el resultado presente varias deficiencias de funcionamiento o aunque trabaje bien, su estructura será difícil de modificar y su desempeño seguramente no será el óptimo. La normalización es un proceso ideado a partir de observar[32] que algunos sistemas de relaciones cumpliendo determinadas condiciones responden mejor que otros ante las operaciones básicas del manejo de una base de datos[33]; la normalización permite no solo detectar sino también corregir posibles defectos en los archivos conceptuales que provocan anomalías de almacenamiento.

El proceso de normalización se realiza por etapas, al final de cada una queda garantizado que todas las relaciones se hallan en la forma normal correspondiente a esa etapa:

---

[32]Atre, Shakuntala, *Técnicas de Bases de Datos*, p105

[33]Vid Sección 1.1.5, *Operaciones*

Primera forma normal

Un modelo con relaciones no normalizadas presentará multiples ocurrencias de un atributo para una misma eneada, es decir que para una sola clave u ocurrencia de la parte rectora corresponden varios valores de atributo perdiéndose por consiguiente la dependencia funcional; el problema se corrige haciendo un registro por cada valor diferente de un campo. A continuación mostramos un modelo relacional prototipo no normalizado y el resultado después de normalizarlo.

TABLA SIN NORMALIZAR

CUENTA	SUBCUENTA	NOMBRE	SALDO
110	10	Salarios	9,000
	20	Viáticos	3,500
	30	Bonos	2,000
120	10	Equipo oficina	2,000
	20	Mantenimiento	1,000
210	10	Renta	7,500
	20	Agua,luz,tel.	1,500
	30	Préstamos	6,000

TABLA EN LA PRIMERA FORMA NORMAL

CUENTA	SUBCUENTA	NOMBRE	SALDO
110	10	Salarios	9,000
110	20	Viáticos	3,500
110	30	Bonos	2,000
120	10	Equipo oficina	2,000
120	20	Mantenimiento	1,000
210	10	Renta	7,500
210	20	Agua,luz,tel.	1,500
210	30	Préstamos	6,000

Los elementos de modelado que propusimos, aparentemente, nos obligan a construir el modelo ya desde un principio en la primera forma normal; pero el problema estará oculto bajo la presencia de cualquier dependencia funcional transitiva multivaluada, sin embargo su solución no será abordada hasta la tercera y cuarta formas normales.

De llevar hasta la implementación un modelo con sus relaciones únicamente en la primer forma normal se corre el riesgo de funcionamiento inadecuado:

1) Como la primer forma normal acepta que en una misma eneada los atributos tengan dependencia funcional con diferentes claves o subconjuntos de la parte rectora, es posible que tengamos varias ocurrencias de dos o más entidades por eneada, de modo que no es posible dar de alta una ocurrencia de entidad independientemente, pues tenemos que agregar una eneada completa.

2) De igual forma las supresiones independientes no son posibles y se corre el riesgo de perder información sobre otras entidades al cancelar la eneada.

3) Un problema derivado del anterior es la redundancia, pues pueden presentarse relaciones uno a varios entre entidades reunidas en una misma tabla, además, si queremos actualizar los datos correspondientes a una entidad que tiene ocurrencias múltiples nos veríamos obligados a hacer las modificaciones en todas y cada una de ellas con objeto de mantener consistencia

Para ejemplificar estas fallas de funcionamiento usaremos el siguiente modelo simplificado de los fletes que realiza un compañía:

1) Cada FLETE tiene cierto COSTO y se identifica por el CAMION que lo hizo y la FECHA del viaje. Además los viajes se hacen en RUTAS que van de una ciudad ORIGEN a otra ciudad DESTINO.

2) Los OPERADORES conducen camiones y de ellos se tiene conocimiento sobre su número de LICENCIA, DOMICILIO, EDAD y NOMBRE.

3) Los camiones tiene diferentes CARACTerísticas: MARCA, MODELO y número de PLACAS.

```
FLETE=    FECHA, CAMION, CARACT, COSTO, OPERADOR, RUTA
OPERADOR= NOMBRE, EDAD, DOMICILIO, LICENCIA
CARACT=   MARCA, MODELO, PLACAS
RUTA=     ORIGEN, DESTINO

DF(CAMION,FECHA)= RUTA, CARACT, OPERADOR, COSTO

DF[CAMION,FECHA]= RUTA, OPERADOR, COSTO
DF[CAMION]=       CARACT

DF[NOMBRE]= EDAD, DOMICILIO, LICENCIA
```

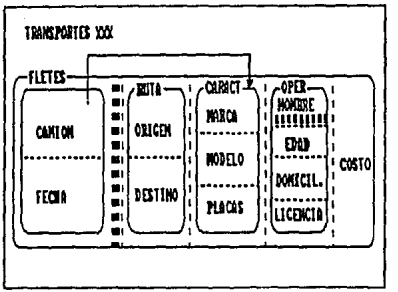


Figura 1.33
Modelo en la primera forma normal

Los problemas ocasionados por la dependencia funcional no total DF(CAMION,FECHA) son: falla de inserción si la compañía adquiere un nuevo camión y este todavía no ha hecho ningún viaje; perdida de información referente a fletes cuando un camión se venda; si hay un cambio en el número de placas de un camión el esfuerzo de actualización se multiplicará por el número de eneadas donde aparezca el camión; además como en cada eneada se repiten las características de los camiones tenemos un nivel alto de redundancia.

En resumen, se dice que una archivo conceptual está en la primera forma normal si sus campos tienen un solo valor por cada registro lógico[34].

Segunda forma normal

La condición para que una relación se encuentre en la segunda forma normal es que todo atributo fuera del área de la parte rectora tenga dependencia funcional total respecto a la clave. Para conseguir esto es probable que nuevas entidades antes no detectadas se tengan que considerar.

Para conseguir esta forma normal transformamos nuestro modelo relacional en dos pasos: Primero, quitando los campos dependientes que no sean totalmente funcionales de la parte rectora anotándolos aparte; y segundo, formando nuevas tablas con estos campos y partes rectoras que de ser necesario contengan un subconjunto de la parte rectora original; por último, estableciendo las relaciones adecuadas entre ellas. Asi, el ejemplo antes considerado se transforma:

[34]Tsai, Alice, *Sistemas de Base de Datos*, p456

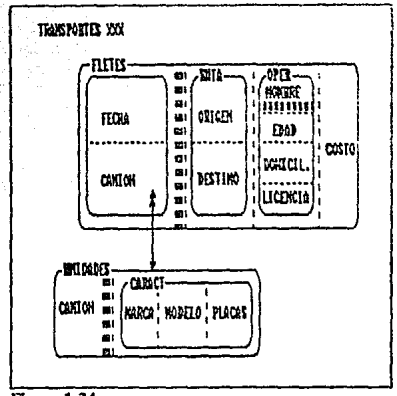


Figura 1.34
Modelo en la segunda forma normal

Aun después de esta normalización el modelo conceptual presenta varios problemas que surgen principalmente por la existencia de dependencias transitivas[35]; problemas de inserción, supresión y actualización muy semejantes a los contemplados en la primera forma normal pues todavía pueden existir varias entidades en una misma tabla.

En el ejemplo mostrado, habrá falla de inserción si es contratado un nuevo operador y no ha hecho ningún viaje todavía; o bien, si es suprimido del archivo conceptual con él se irá la información de los viajes que realizó; nuevamente como en la primera forma normal el alto grado de redundancia multiplica el esfuerzo de actualización.

Tercera forma normal

Esta forma normal la determinó en 1972 Codd posteriormente, Boyce la revisó y mejoró creando la forma BCNF (*Boyce-Codd Normal Form*).

Se dice que una relación está en la tercera forma normal si no hay relaciones de dependencia funcional entre atributos que están fuera de la parte rectora, es decir, atributos no clave. Por consiguiente la tercera forma normal por definición elimina las dependencias transitivas entre atributos no rectores; el proceso de normalización en esta fase es el siguiente:

[35] Vid sección 1.4.2 *Dependencias*

Buscar en cada atributo no pertenenciente a la parte rectora posibles dependencias funcionales respecto a otros atributos también dependientes y crear una nueva tabla donde queden registrados; luego eliminar de la tabla original el o los atributos que juegan el papel de parte dependiente en la tabla nueva; por último establecer la relación adecuada entre ambas entidades. A continuación transformamos el modelo que hemos mantenido como ejemplo desde la primera forma normal:

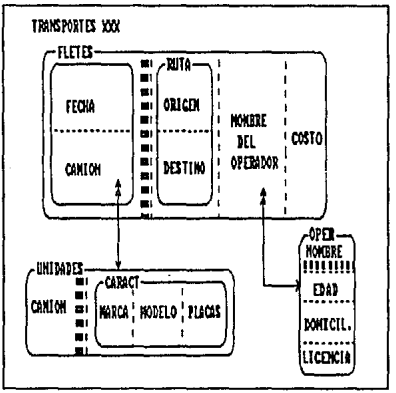


Figura 1.35
Modelo en la tercera forma normal

Frecuentemente la normalización termina cuando todas las relaciones están en la tercera forma normal, sin embargo es posible que todavía existan algunas anomalías de funcionamiento presentadas por tipos de dependencia transitiva especiales; además, como en nuestro esquema de modelo conceptual pueden existir todavía dependencias funcionales sencillas sobrepuestas con multivaluadas debemos continuar la normalización hasta la cuarta forma normal.

En cuanto a la mejoría realizada por Boyce, se dice que una relación es BCNF si está en la tercera forma normal y no existe ningún tipo de dependencia funcional transitiva[36]. Recordemos cuales son:

[36]Vid *Tipos de dependencia funcional* en secc. 1.4.2

Para una dependencia funcional de la forma:

```
DF[X1,X2,...,Xm]= Y1,Y2,...,Yn
```

Se tienen las siguientes posibles dependencias funcionales transitivas:

```
1) DF[Ya,Yb,...,Yc]= Yi,Yj,...,Yk
2) DF[Ya,Yb,...,Yc]= Xi,Xj,...,Xk
3) DF[Xa,Xb,...,Xc]= Xi,Xj,...,Xk
```

El proceso de normalización a la tercera forma elimina las dependencias del tipo 1; ahora bien, las dependencias entre atributos rectores (tipo 3) se resuelven haciendo una entidad auxiliar que contenga esos atributos, y relacionando ambas entidades con la forma uno a muchos.

Tomemos el ejemplo de la Figura 1.36 que cumple las condiciones impuestas por la tercera forma normal pero que tiene una relación no BCNF:

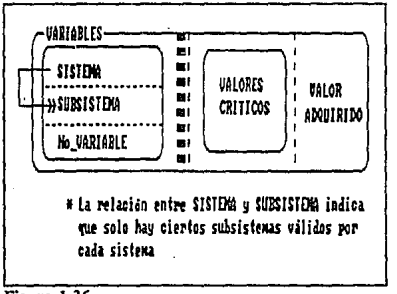


Figura 1.36
Entidad "variables" con relación no BCNF

El mismo modelo ya en la forma de Boyce-Codd:

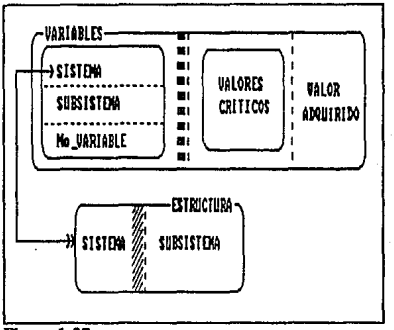


Figura 1.37
Modelo en la forma Boyce-Codd

El método para eliminar la dependencia del tipo (2) presenta diversas variantes y de hecho, es común que no se le considere con la profundidad adecuada[37].

El primer paso del procedimiento de normalización para este tipo de dependencia funcional transitiva en una entidad E1 consiste en eliminar la dependencia de la entidad E1 y hacer una nueva entidad E2 que contenga como parte rectora al elemento dependiente de E1 (Ya,Yb,...,Yc) y como parte dependiente al subconjunto de atributos rectores de E1 (Xi,Xj,...,Xk).

Durante este primer paso si el conjunto (Ya,Yb,...,Yc) es igual a (Y1,Y2,...,Yn); es decir, que constituye toda la parte dependiente, podemos simplificar la parte rectora de E1 eliminando al subconjunto (Xi,Xj,...,Xk). Como segundo y último paso establecemos la relación adecuada entre E1 y E2.

Para dejar este punto mas claro expondremos dos ejemplos basándonos en modificaciones al modelo conceptual tratado anteriormente, en ambos agregamos una dependencia transitiva entre OPERADOR y CAMION, lo que quiere decir que cada operador maneja solamente un camión pero puede ser que una misma unidad sea manejada por dos operadores en diferentes fechas:

1) Para el primer ejemplo se definen las siguientes expresiones matemáticas de las relaciones entre campos:

[37]Cfr. Tsai, Alice, *Sistemas de Base de Datos*, p467

```
DF[FECHA,CAMION]= OPERADOR
DF[OPERADOR]= CAMION
```

Para obtener la BCNF, tal como se indicó en el primer paso, creamos una entidad auxiliar con OPERADOR como parte rectora y CAMION como parte dependiente. Y ya que la parte dependiente de FLETES, tiene por campo único a OPERADOR, simplificaremos su parte rectora quitando el campo CAMION. En el segundo paso establecemos una relación varios a uno entre ambas entidades, dada la condición de que un operador puede manejar diferentes camiones en distintas fechas, pero solamente uno en un mismo dia. La Figura 1.38 muestra el modelo antes y después de normalizar

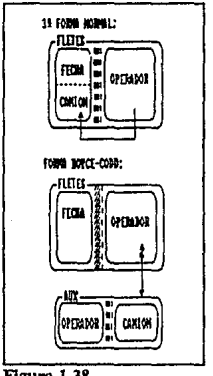


Figura 1.38
Normalización con reducción de parte rectora

2) El segundo ejemplo contemplará la existencia de más campos en la parte dependiente de FLETES:

```
DF[FECHA,CAMION]= OPERADOR, OTROS_DATOS
DF[OPERADOR]= CAMION
```

En este caso, la parte rectora de FLETES no puede simplificarse ya que su parte dependiente no forma parte en su totalidad de la dependencia funcional (OTROS_DATOS no establece dependencia funcional con CAMION); por lo demás, el procedimiento es el mismo que el seguido en el ejemplo anterior.

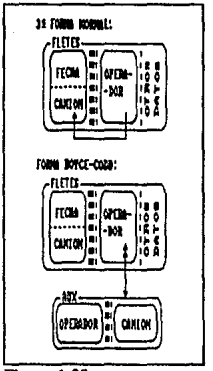


Figura 1.39
Normalización conservando parte rectora intacta

Cuarta forma normal

La eliminación de dependencias multivaluadas que implica el proceso de normalización a la cuarta forma, representa mejoras ya no de tipo funcional sino en cuanto a optimización de espacio; las DMV son insustituibles pero cuando debido a su presencia a una parte rectora le corresponden diversos valores de atributos dependientes, la redundancia se multiplica, por ejemplo:

Sea la dependencia funcional:

```
DMV[X]= Y,Z
```

Tendremos que por la ocurrencia específica "x1" de X habrá varias -n- ocurrencias de Y y -m- de Z, y por consiguiente existirán n veces m eneadas con el valor "x1" para X; existen dos caminos para reducir el problema:

1) Si -n- y -m- son limitados podemos reexpresar la dependencia como:

```
DMV[X]= Y1,Y2,...,Yn,Z1,Z2,...,Zm
```

2) Si -n- y -m- varían en límites muy grandes, entonces conviene hacer dos dependencias individuales:

```
DMV[X]= Y
DMV[X]= Z
```

Consideraciones finales

Algunos autores hablan de la existencia de una quinta forma normal[38] sin embargo, las deficiencias del modelo conceptual quedan eliminadas al llegar a la cuarta forma normal; es en el mapeo a un modelo lógico (seleccionando el tipo de la base) que se deben refinar las especificaciones. Antes de dar por terminado el tema de normalización y aunque ya no esté dentro de este proceso, conviene eliminar las relaciones del tipo varios a varios entre entidades pues ningún tipo de base de datos permite de manera directa establecer esa relación entre dos archivos. La eliminación se hace a través de una entidad conectora:

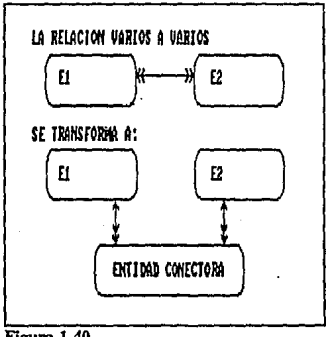


Figura 1.40
Sustitución de relaciones varios a varios

1.4.3. Diccionario de datos

Un diccionario de datos contiene información sobre las entidades que forman parte de un sistema, desde el nombre hasta su representación física. Son compendios sobre la información relativa a los datos de un sistema, las relaciones entre ellos y su uso. Por esta razón, resulta ser una herramienta útil para cualquier modelo conceptual, lógico o físico; conforme el proceso de diseño

[38] Date, C.J., *Introducción a los Sistemas de Bases de Datos*, p267

avanza, el diccionario crece al tiempo que definimos la base y en una etapa posterior, juega un papel importante durante el desempeño de la misma, especialmente si se encuentra integrado al manejador de base de datos, en cuyo caso, el DBMS podrá consultar aspectos de almacenamiento físico relacionados con las entidades tales como la descripción de la estructura de la base, mejorándose asi la independencia de datos.

Algunos autores consideran que no es función del diccionario de datos poner a la disposición del manejador la información de tipo físico; aseguran que los diccionarios de datos son tan solo predecesores[39] de una descripción formal automatizada de la estructura de la base de datos llamada *esquema*. Por otro lado existe la tendencia a incluir[40] el esquema dentro del diccionario de datos, ya sea que este último se encuentre integrado al DBMS o se lleve de forma manual, método donde es de considerar el inconveniente de las anotaciones dobles; una, en el diccionario escrito o en procesador de palabras y otra en el esquema del DBMS.

Nosotros tomaremos la posición que considera al esquema como parte integral del diccionario de datos; de hecho, ya que el proceso de construcción del diccionario de datos va de la mano con el proceso de modelado, el esquema que viene a ser la descripción lógica de la base de datos[41] se prepara como pre-requisito del modelo físico. Por último cabe señalar que en cuanto a su estructura, un diccionario es similar a una base de datos, aunque en lugar de contener datos en si, nos proporciona la información que los describe y permite manipular; por ello también es conocido como *diccionario meta-datos*[42]

Objetivos

El diccionario de datos tiene dos objetivos generales: ser el compendio de "reglas gramaticales" que las personas relacionadas con la base de datos debe conocer para comunicarse con un lenguaje común y, encargarse de centralizar y controlar el flujo de datos para los diferentes modelos mediante la descripción de la estrutura de la base en todos sus niveles es decir, dentro de los campos, dentro de registros y entre registros. Más ampliamente, el diccionario de datos debe:

* Establecer un lenguaje común.
* Resumir técnicamente la historia del sistema
* Propiciar la comunicación entre usuarios.
* Guiar a programadores dentro de la estructura.
* Mantener la información necesaria para garantizar independencia.
* Si está integrado, controlar las características de los campos.

---

[39]Wiederhold, Gio, *Diseño de Bases de Datos*, p480

[40]Campderrich, B., *Técnicas de Bases de Datos*, p39

[41]Martin, James, *Principles of Data Base Management*, p74

[42]Tsai, Alice, *Sistemas de Base de Datos*, p178

* Fungir como herramienta para los diferentes modelos.
* Manejar información durante las fases de diseño, implantación, operación y mantenimiento.

Información a registrar

Dos son los pasos iniciales al empezar la construcción de un diccionario de datos; primero, determinar cuales son las entidades, los campos y su descripción textual; segundo, estudiar las relaciones entre campos, entidades y aplicaciones. El resto de la información se agregará durante el proceso de modelado, ofreciéndose al principio de cada etapa la siguiente información.

Modelo Conceptual:

Por Entidad:
- Nombre y sinónimos
- Descripción textual
- Relaciones entre entidades
- Atributos que la forman
- Descripción de partes rectora y dependiente
- Frecuencia de uso

Por Atributo:
- Nombre y sinónimos
- Descripción textual
- Relaciones entre atributos
- La(s) persona(s), área(s) o programa(s) que lo requiere
- Utilidad e importancia
- Frecuencia de uso

Modelo Lógico:

Modelo conceptual gráfico
Atributos que forman las claves de acceso
La descripción de los archivos lógicos
Segmentos de la información que cada módulo o programa usa
Lenguaje de programación o paquete DBMS
Flujo de la información a través de los diferentes módulos

Modelo Interno:

El modelo conceptual mapeado hacia un tipo de base de datos

Por Campo:
- Tipo de dato a almacenar
- Longitud (No_Bytes)
- Tipo de alineación
- Constante o variable
- Algoritmo relacionado (si es variable calculada)
- Transformaciones de formato en almacenamiento interno
- Códigos de autorización

Por Registro:
- Estructura física de almacenamiento
- Longitud (No_Bytes)
- Códigos de autorización

1.5. TIPOS DE BASE DE DATOS

El paso de modelo conceptual a modelo lógico nos obliga a definir más formalmente la estructura, seleccionando un tipo de base de datos o enfoque de acuerdo a las estructuras de datos y operadores asociados que debe soportar el sistema[43]. Durante la década de los sesenta solo había dos opciones bien definidas y probadas: los enfoques jerárquico y de red; fue hasta inicios de los años setenta cuando el modelo relacional tomó auge.

Comercialmente existen varios DBMS desarrollados bajo los tres enfoques[44]:

JERARQUICO: IMS de IBM, SYSTEM 2000 de Intel

RED: IDMS de Cullinet, DBMS-10/20 de DEC, IMAGE de HP

RELACIONAL: SQL/DS y DB2 de IBM, ORACLE de RSI e INGRES de Relational Technology

---

[43]Date, C.J., *Introducción a los Sistemas de Bases de Datos*, p71

[44]Tsai, Alice, *Sistemas de Base de Datos*, p204

Antes de entrar en detalles sobre las tres principales opciones, presentamos a grandes razgos las características de cada una:

Jerárquico: La estructura de este tipo de bases se inspira en los diagramas de árbol y queda determinada por el conocimiento de los argumentos de búsqueda originando rutas de acceso bien definidas; por tanto, los procedimientos de recuperación de información pueden proporcionar un registro de estructura completa o un segmento de la base que corresponda a una subestructura de la misma.

Red: Una estructura de red como su nombre lo indica, tiene menos limitantes en cuanto a conectividad entre entidades respecto al tipo jerárquico, sin embargo, la construcción de la estructura de la base también depende de los campos llave. Es de notar que la flexibilidad adicional de la estructura permite recorrerla completamente.

Relacional: Apoyándose en teorías matemáticas las bases de datos de este tipo se construyen a partir de archivos conceptuales en forma de tablas bidimensionales, destacando como diferencia respecto a los tipos anteriores el hecho de no existir apuntadores físicos para relacionar registros sino "campos de conexión".

1.5.1. Enfoque jerárquico

Historia

De los tres enfoques que abordaremos, el jerárquico fue primero en surgir formalmente definido y totalmente desarrollado en sistemas como el IMS (*Information Management System*) que apareció en 1960; sin embargo, no hay una norma referente al enfoque y por consiguiente cada sistema tiene características propias y existen diferencias considerables entre cada uno. Aunque desde el punto de vista de software es un sistema anticuado, se le han hecho considerables adaptaciones para enfrentar la fuerte competencia que representan los sistemas de red y relacionales.

Las bases de datos del tipo jerarquico ofrecen como ventajas probidad, extensa documentación sobre su uso en diferentes campos y lo que es más importante, como a continuación veremos, alta eficiencia de acceso y almacenamiento para volúmenes de información grandes.

Características

El enfoque jerárquico se basa en estructuras de tipo árbol; por la afinidad que tiene el ser humano a este tipo de estructuras puestos en práctica en múltiples circunstancias como organigramas de empresas y árboles genealógicos, el modelo es relativamente fácil de comprender; las bases de datos jerárquicas de sistemas complejos inician su estructura a partir de niveles donde varias bases de datos co-existen y almacenan diferentes porciones del universo total de datos. Una base de datos específica tiene en los niveles superiores, los más cercanos al nodo raiz o inicial, entidades "dominantes" sobre las de niveles inferiores. Entre niveles se establecen relaciones uno a varios asociando una entidad "padre" con varias ocurrencias de entidades "hijo". Un ejemplo de diagrama jerárquico es el mostrado en la Figura 1.41.

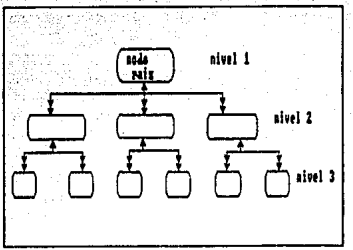


Figura 1.41
Ejemplo: Diagrama jerárquico

Las reglas para construir jerarquías son:

1) En el primer nivel del modelo solo existe un nodo llamado nodo raíz, este nodo es único. Por el contrario, los nodos de los últimos niveles se llaman hojas.

2) Pueden existir varios niveles y dentro de cada uno (a excepción del primero), tantos nodos como sean necesarios.

3) A cada nodo le corresponde comunmente una entidad aunque no hay impedimento para la existencia de varias en un mismo nodo, siempre y cuando los atributos que las describen se incluyan.

4) Los nodos correspondientes al nivel "X" donde $X > 1$ se conectan hacia arriba únicamente con un nodo del nivel X-1 (el cual es llamado nodo "padre") pero pueden conectarse con varios del siguiente nivel.

5) Las operaciones de recuperación dirigidas a un nodo deben recorrer la estructura de la base a partir del nodo raíz; dadas esta condición, solo existe una ruta de acceso válida para cada nodo.

6) No hay restricción en cuanto al número de ocurrencias de un nodo hijo por cada ocurrencia del nodo padre; puede haber varias, una o ninguna. Es decir que en general el tipo de relación que enlaza nodos de niveles diferentes es del tipo uno a varios.

Las bases de datos jerárquicas por sus características estructurales presentan "deficiencias" de inserción y supresión semejantes a las existentes en modelos conceptuales que no llegan a la tercera forma normal[45]; semejante comportamiento puede representar serios inconvenientes si el

[45]Vid, *Normalización*, secc. 1423

sistema es de información y los requerimientos cambian. Estos problemas se originan por un lado porque una ocurrencia de nodo hijo no puede existir sin su correspondiente ocurrencia de nodo padre y porque al eliminar una ocurrencia de nodo padre, las asociadas con este en niveles inferiores automáticamente desaparecen.

Por la estructura de los nodos que conforman un árbol se distinguen dos tipos de árbol:

Arbol homogéneo: Si todos los nodos tienen la misma estructura, es decir que representan el mismo tipo de registro.

Arbol heterogéneo: Cuando los nodos que forman al árbol corresponden a tipos de registro diferentes.

Mapeo

El paso de modelo conceptual a modelo lógico del tipo jerárquico nos obliga a modificar su estructura hasta conseguir una forma arborescente; para esto los conflictos principales son en lo tocante a paternidad múltiple y se resuelven básicamente de dos formas:

El primer método consiste en crear bases de datos independientes por cada problema de paternidad múltiple, la forma de separación depende de los requerimientos; por ejemplo, sea el caso simplificado de los nodos A,B,C,D y E expuesto en la Figura 1.42.

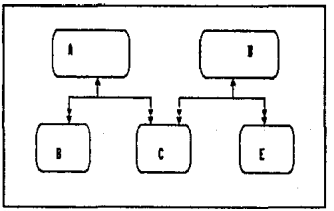


Figura 1.42
Paternidad múltiple

Si los elementos A y D son de un tamaño reducido lo suficiente como para que duplicaciones de ellos no aumenten en mucho la redundancia y si las peticiones de recuperación ya sean por el nodo A o el D frecuentemente van hacia el nodo C; conviene resolver el problema de paternidad múltiple derivando dos bases de datos independientes de la siguiente forma:

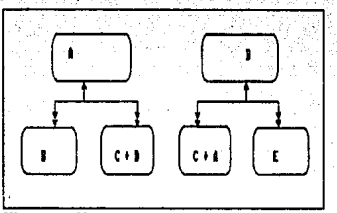


Figura 1.43
Paternidad múltiple: separación en dos bases independientes

Otra posibilidad con un nivel de redundancia menor pero que da trabajo extra al manejador de la base consiste en la creación de dos bases de datos independientes relacionadas a nivel lógico, este método solo se puede llevar a cabo si el manejador específico es capaz de establecer relaciones entre bases de tipo lógico (como ejemplo, el IMS lo hace); la Figura 1.44 muestra la solución del problema indicandose con líneas punteadas las relaciones lógicas.

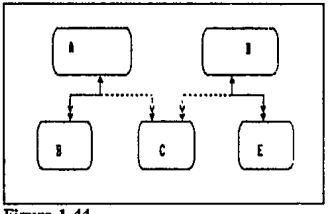


Figura 1.44
Paternidad múltiple: solución por relaciones lógicas

En bases de datos de estructura sencilla la adecuación a un modelo jerárquico se hace simplemente resolviendo los problemas de paternidad múltiple con alguno de los métodos antes expuestos, pero formalizando, el procedimiento al detalle puede dividirse en varios pasos:

1) Simplificar relaciones múltiples entre nodos eliminando aquellas con *transitividad* (transitividad se trata aquí de un modo diferente al aplicado en dependencias funcionales, hace referencia a relaciones derivables de otras; deben identificarse claramente para que al eliminarlas no se pierda información):

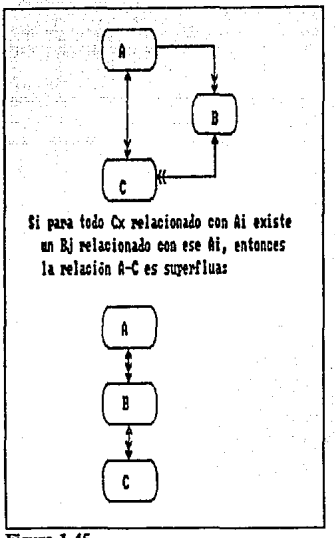


Figura 1.45
Simplificación de relaciones

2) Obtener relaciones candidatas del tipo *padre-hijo*; por ejemplo, si la relación entre el nodo A y el B es de uno a muchos, podemos proponer a A como padre de B. Se dice "proponer" por que el resultado obtenido de una primera asignación muy probablemente no cumplirá con los requisitos necesarios para considerarse como árbol principalmente por problemas de paternidad múltiple.

3) Si el manejador de base de datos que se va a usar soporta paternidad múltiple el problema se simplifica, sin embargo los manejadores tradicionales no lo hacen o bien, ofrecen un número limitado de padres por nodo y es necesario adecuar el modelo siguiendo alguno de los procedimientos antes descritos.

4) Buscar un balance entre desempeño y memoria requerida, combinando hijos únicos con sus padres. Por ejemplo, en el diagrama siguiente para ocurrencias múltiples de los atributos presentes en el nodo D, podemos encontrar una sola ocurrencia del nodo B (relación uno a varios); en este caso la redundancia es baja pero a cambio de ello el funcionamiento se verá

afectado por el acceso a dos nodos a través de apuntadores y debemos decidir si conviene la unión de B con D.

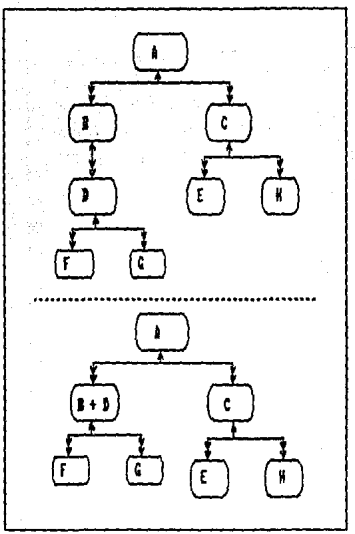


Figura 1.46
Mejoría en el funcionamiento a costa de redundancia

Por último, debe cuidarse que en cada nodo solo tengan la categoría de clave aquellos atributos que así lo requieren, prestando especial atención a los nodos que hayan absorbido a sus hijos.

1.5.2. Enfoque de red

Historia

El desarrollo del modelo reticular (también llamado *estructura plex*) está ampliamente ligado con los trabajos de CODASYL (Conferece on Data Systems Language: *Conferencia sobre Lenguajes de Sistemas de Datos*), este grupo se estableció en 1959 en la ciudad de Washington y fue resultado de un encuentro entre algunos de los mas importantes usuarios de computadoras,

fabricantes, institutos de investigación y departamentos de gobierno. Su objetivo no ha sido establecer normas sino desarrollar y "recomendar" técnicas de programación de sistemas relacionados con sistemas de procesamiento de datos. La primer tarea de CODASYL fue el desarrollo de un lenguaje de programación adecuado para la implementación de aplicaciones en el área de los negocios en general, de esta forma nace COBOL, (*Common Business Oriented Language*).

Años después, en 1966 CODASYL se lanzó a la tarea de ampliar el COBOL para soportar el manejo de bases de datos; para 1971 ya se presentaba un modelo completo que estableció los conceptos fundamentales del modelo reticular y han seguido mejoras llevadas a cabo por diferentes grupos de trabajo dentro del mismo CODASYL como el comité para el lenguaje de descripción de los datos, *DDLC*, que surgió para proponer recomendaciones de almacenamiento interno de datos, y el grupo de tareas de bases de datos, *DBTG*.

Características

En el modelo reticular el concepto de nodo dominante y nodo subordinado expuesto en el modelo jerárquico se extiende ya que un mismo nodo puede ser "propiedad" o en la terminología empleada para el enfoque jerárquico, puede ser "hijo" de uno o más nodos.

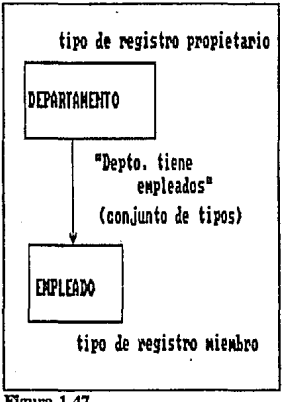


Figura 1.47
Ejemplo de conjunto de tipos

La simbología introducida por C. W. Bachman[46] se usa para los diagramas de estructuras de red, en ella cada bloque representa un tipo de registro (recordemos que *tipo de registro* hace referencia a una clase de información, sin tomar en cuenta una ocurrencia o registro físico de la misma.) que está compuesto por al menos un atributo; los bloques se unen mediante grafos dirigidos donde el tipo de registro del que sale la flecha se le conoce como tipo de registro propietario y el bloque apuntado viene a ser el tipo de registro miembro; al grafo que une esos elementos, se le llama *conjunto de tipos* y establece relación uno a varios entre propietario y miembro. (Es de hacer notar que el concepto "conjunto" tiene un tratamiento muy diferente al dado por la teoría de conjuntos).

Observando la notación expuesta en la Figura 1.47, vemos que no difiere en mucho de la que hemos propuesto para el modelado conceptual y que podemos mantener esta última en los diagramas. Las reglas de formación de sistemas reticulares son:

1) Los conjuntos de tipos como ya se había indicado son agrupaciones de al menos dos tipos de registro: el "propietario" y el "miembro", relacionados entre si e identificados por un nombre. Es posible que un conjunto tenga varios miembros pero solo puede tener un propietario.

2) En cada ocurrencia de conjunto puede haber solo una ocurrencia del tipo de registro propietario.

3) Por otro lado, pueden presentarse una, varias o ninguna ocurrencia de registros miembro por ocurrencia de conjuntos de tipo.

4) Cualquier ocurrencia de registro miembro solo puede pertenecer a una ocurrencia de cada registro propietario.

Los tipos de conjunto se dividen en tres clases de acuerdo a los elementos que relacionan:

1) Si entre dos bloques o tipos de registro existen dos tipos de conjunto diferentes, podemos decir que hay una relación de *contexto múltiple* entre ellos.

2) Un mismo tipo de conjunto que tenga más de un miembro se dice que es de *múltiples miembros.*

3) Por último está el tipo de conjunto singular que si bien relaciona un propietario con un miembro no permite mas que una sola ocurrencia del tipo en la base; este tipo es útil cuando se deben reunir registros en un conjunto pero ninguno de ellos es propietario o cuando temporalmete por falta de información completa se introducen registros sin propietario para que en un futuro sean actualizados.

---

[46] Aire, *Técnicas de Bases de Datos*, p121

Un análisis de la estructura de red demuestra que por un lado tiene notorias ventajas sobre el modelo jerárquico no solo porque el grupo CODASYL continua refinando el modelo y dando "sugerencias" que se convierten en estándares a la larga, sino por aspectos de funcionamiento dinámico tanto en inserción como supresión gracias principalmente a la posibilidad de parentesco múltiple, pero por otro lado, es también debido a esta característica que la complejidad del modelo se multiplica siendo aun para los programadores de aplicaciones difícil de manejar; el programador debe ubicarse dentro de la estructura de la base y ser capaz de moverse a través de ella, por lo que es recomendable escribir una biblioteca de funciones de uso general que efectúen esta tarea. Otro inconveniente ha sido detectado durante reorganizaciones de las bases de datos en las que la integridad de los datos se reporta dañada principalmente por detalles no contemplados en el desarrollo de los programas que manipulan la información.

Mapeo

El proceso de conversión de modelo conceptual a reticular es mas simple que la transformación a jerárquico aunque tampoco podemos decir que exista una receta o método único; durante la fase de refinamiento del modelo, diferentes consideraciones de funcionamiento llevarán a diferentes resultados.

La primer tarea del mapeo es encontrar las relaciones varios a varios, tipo de relación con la que el modelo reticular no está preparado para trabajar directamente por ejemplo, un tipo de registro 'A', propietario de 'B' no puede ser al mismo tiempo miembro de 'B' siendo inválidas las siguientes relaciones:

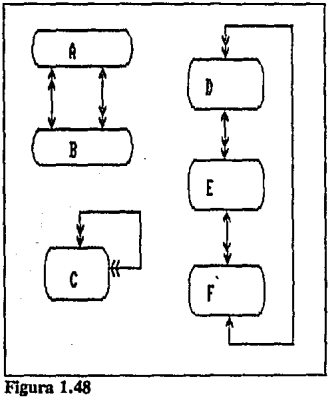


Figura 1.48
Relaciones no permitidas dentro del enfoque de red

Para reemplazar las relaciones varios a varios tenemos que crear una entidad adicional que sirva de enlace, tal como se indicó en el último paso del modelado conceptual. Si ya hemos tomado en cuenta esta consideración, el primer paso es simplificar las partes rectoras de los tipos de registro eliminando campos repetidos en tipos miembro, observemos el siguiente ejemplo:

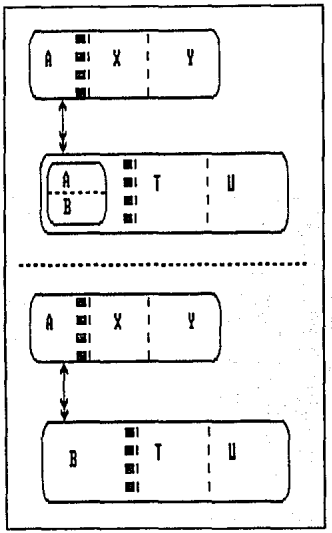


Figura 1.49
Sustitución de las relaciones varios a varios

Posteriormente solo queda adecuar el modelo conceptual a modo que sea factible implantarlo con el DBMS disponible, ya que algunos pueden tener limitantes como número de tipos de registro o de conjuntos. También conviene en este momento eliminar aquellos tipos de registro con información repetida y unir los tipos de registro con campos cuyos datos sean prácticamente estáticos siempre y cuando el resultado de la unión no afecte en mucho la redundancia; tómese como ejemplo el modelo no normalizado de la sección 1.4.2 de un ambiente universitario, en este se dejó como entidad independiente HORARIOS, entidad relacionada solamente con CURSOS; resulta claro que accesar otro tipo de registro en el modelo relacional requiere tiempo, uniendo ambos aunque incrementamos la redundancia ganamos en tiempo y lo que es mas, queda independiente el horario de cada curso pudiendo asi modificarlo fácilmente.

Consideraciones como la anterior parecen insignificantes en papel pero afectan profundamente el desempeño de una base de datos, debe tenerse presente que el objetivo de un modelo conceptual es representar claramente la realidad mientras que el modelo lógico ya va dirigiéndose a aspectos de funcionamiento.

1.5.3. Enfoque relacional

Historia

La teoría que sustenta al modelo relacional fue introducida por el Dr. E. F. Codd en 1970, y fue hasta principios de la década de los 80 que aparecieron los primeros prototipos experimentales basados en ella tales como SYSTEM R, INGRES y QUERY-BY-EXAMPLE. Hoy en día, la popularidad de este tipo de base de datos crece constantemente y es frecuentemente la opción más viable aun para sistemas de información grandes o complejos. Esto sin contar los ambientes de computadoras personales, universitarios y de investigación donde prácticamente no tiene rival.

Características

Los cimientos del modelo relacional son tablas bidimensionales usadas para representar no solo las entidades con sus campos sino las mismas relaciones, aunque en la terminología convencional de este enfoque se les llama "relaciones" a estos arreglos, nosotros mantendremos el término "tabla" para evitar confusiones.

Para que una tabla se considere como tal y pueda llegar a la categoría de archivo conceptual, debe presentar las siguientes características:

1) Los renglones (llamados también *n-adas*) no pueden duplicarse.

2) Los renglones deben tener estructura semejante, es decir que deben poderse alinear en columnas las cuales se identificarán por un nombre único y corresponden a los atributos de la entidad además, se especifica para cada atributo cierto dominio o conjunto de valores posibles.

3) Aunque el orden de los renglones y de las columnas carece de importancia, es conveniente presentarlos en forma ordenada de acuerdo a su clave de acceso para facilitar la comprensión.

4) Pueden presentarse en una misma tabla hasta tres tipos de claves: clave principal, clave aspirante (definidas en la sección 1.3.1) y clave externa por medio de la que se relaciona la tabla con otra.

La característica más interesante del modelo relacional es que las tablas no ocupan un nivel jerárquico fijo, dicho de otra forma, para accesar un dato no tenemos que recorrer cierta trayectoria dentro de la estructura pasando por diferentes nodos como en el caso de los modelos jerárquico y de red; lo podemos accesar directamente recurriendo a la tabla donde se encuentra.

Presentar la información mediante este formato facilita la comprensión de la organización de los datos no solo a los programadores sino además a los usuarios finales quienes no tienen que preocuparse por la forma en que quedan almacenados físicamente los datos; es por esto que han podido lanzarse al mercado exitosamente manejadores de base de datos con intérpretes de comandos de alto nivel integrados.

El problema de relacionar diferentes entidades se aborda desde otro punto de vista, en lugar de proponer nodos padre o propietarios y nodos hijos o miembros, son repetidos campos clave en las tablas a relacionar. Al no existir una trayectoria de acceso fija la productividad del programador que trabaja sobre el núcleo del manejador de base de datos aumenta, pero tomemos en cuenta que el desempeño del sistema contruido muy probablemente será baja, ya que ejecutar los accesos requerirá de bastante esfuerzo.

Mapeo

Algunos autores ligan fuertemente el proceso de normalización con la representación en tablas de las entidades y proponen el desarrollo de un modelo conceptual relacional normalizado aprovechando la claridad con que se muestran los datos y sus relaciones, a partir de este modelo derivan fácilmente el modelo lógico relacional y posteriormente si es necesario los modelos jerárquicos y de red. Nosotros hemos desarrollado en secciones anteriores una simbología propia para el modelo conceptual y por tanto, es necesario un mapeo al modelo relacional que como se verá es bien sencillo; retomemos el modelo conceptual antes presentado:

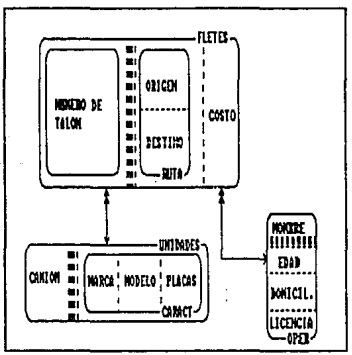


Figura 1.50
Modelo conceptual de una compañía de transportes

Primero transformamos las entidades del modelo conceptual a tablas dejando las flechas indicadoras de relación, agregando un encabezado para los nombres de atributos y marcando de modo especial la división entre parte rectora y dependiente.

FLETES

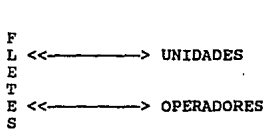
TALON	ORIGEN	DESTINO	COSTO
2340	MEXICO	PUEBLA	653000
2341	SLP	MEXICO	880000
2342	MEXICO	APIZACO	520000

UNIDADES

CAMION	MARCA	MODELO	PLACAS
K1	KENWORTH	89	AD2012
K2	KENWORTH	90	FG2456
D4	DINA	87	LK9672

OPERADORES

NOMBRE	EDAD	DOMICILIO	LICENCIA
LOPEZ	28	CIRUELOS 23	55072
PEREZ	32	LAGOS 85	89243



Hecho lo anterior, de ser necesario agregamos campos de conexión que nos permitan eliminar las flechas. En beneficio de la claridad en los diagramas es útil agregar un encabezado más indicando a que tabla sirven de conexión.

FLETES

****	******	*******	******	UNIDADES	OPERAD.
TALON	ORIGEN	DESTINO	COSTO	CAMION	OP
2340	MEXICO	PUEBLA	653000	K1	LOPEZ
2341	SLP	MEXICO	880000	D4	PEREZ
2342	MEXICO	APIZACO	520000	K1	LOPEZ

UNIDADES

CAMION	MARCA	MODELO	PLACAS
K1	KENWORTH	89	AD2012
K2	KENWORTH	90	FG2456
D4	DINA	87	LK9672

OPERADORES

NOMBRE	EDAD	DOMICILIO	LICENCIA
LOPEZ	28	CIRUELOS 23	55072
PEREZ	32	LAGOS 85	89243

Pocas veces se hace énfasis en la toma de precauciones sobre aspectos de funcionamiento ya que este tipo de bases de datos se desarrollan pensando en la flexibilidad ofrecida para recuperar datos; experiencias personales nos llevan a declarar que el único camino para optimizar un sistema basado en el modelo relacional haciéndolo competitivo ante el jerárquico y de red es el análisis cuidadoso de los requerimientos y la adecuación del modelo lógico a estas necesidades. El administrador de base de datos novato acostumbrado a la facilidad de diseño en ambientes personales cae en la tentación de implantar una base muy flexible pero poco eficiente; deben eliminarse o fusionarse las tablas de campos poco extensos y con información estática, suele ser necesario crear una tabla que contenga los campos mayormente solicitados para la aplicación; llendo más lejos, si se conoce la dinámica del sistema recomendamos con objeto de optimizar tiempos de respuesta, enfocar el modelo lógico a los requerimientos de búsqueda y actualización.

1.5.4. Análisis comparativo

Para el usuario final de un sistema de información y para el programador de una aplicación de un sistema de operaciones cuyo centro de información es una base de datos ya definida es importante el tiempo que tarda en responder el sistema a sus peticiones, el espacio consumido por los archivos de la base, y la serie de pasos necesarios para recuperar cierta información.

El enfoque jerárquico por su construcción rígida con puntos de entrada a la base bien definidos, es ideal para consultas en sistemas de operaciones[47] donde la velocidad de respuesta es lo importante; por ejemplo, una base de datos HSAM almacena por registro de forma contigua en cada ocurrencia del nodo raiz toda la información de sus segmentos, tomemos el siguiente modelo lógico jerárquico:

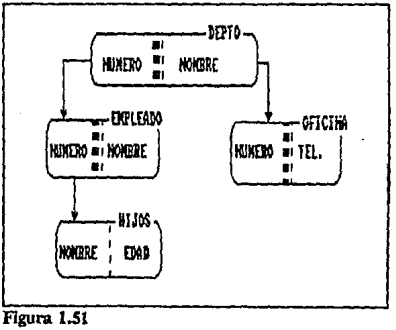


Figura 1.51
Modelo jerárquico de una base con los empleados de una empresa

[47] Vid, *Sistemas de operaciones*, secc. 1.1.1

Los datos en un archivo físico se almacenarían de la siguiente forma:

```
Datos del Depto 1
  Datos del Empleado 1 (Depto 1)
    Datos del Hijo 1 (Empleado 1) (Depto 1)
    Datos del Hijo 2 (Empleado 1) (Depto 1)
      .     .     .     .          .
    Datos del Hijo n (Empleado 1) (Depto 1)
  Datos del Empleado 2 (Depto 1)
    Datos del Hijo 1 (Empleado 2) (Depto 1)
    Datos del Hijo 2 (Empleado 2) (Depto 1)
      .     .     .     .          .
    Datos del Hijo n (Empleado 2) (Depto 1)
      .     .     .     .          .
      .     .     .     .          .
  Datos del Empleado n (Depto 1)
    Datos del Hijo 1 (Empleado n) (Depto 1)
    Datos del Hijo 2 (Empleado n) (Depto 1)
      .     .     .     .          .
    Datos del Hijo n (Empleado n) (Depto 1)
  Datos de la Oficina 1
  Datos de la Oficina 1
  Datos de la Oficina 2
    .     .     .     .     .
  Datos de la Oficina n
Siguientes Deptos
```

Como limitante, tenemos que una vez iniciado el "viaje" dentro de la estructura jerárquica solo podemos recuperar el siguiente segmento del mismo tipo o uno de sus segmentos hijos; el resultado obtenido tendrá la forma de una subestructura de la base.

El enfoque de red aunque presenta una estructura predefinida como el jerárquico, es lo suficientemente versátil como para permitir al usuario recorrer toda su estructura, dando opción a organizaciones de tipo secuencial o aleatorio en los miembros; la Figura 1.52 muestra ambos encadenamientos entre registros miembro y propietario:

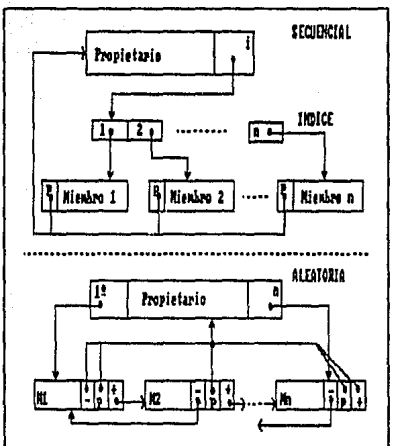


Figura 1.52
Organizaciones secuencial y aleatoria en la formación de tipos de conjunto

Al ofrecer apuntadores a diferentes elementos de la base, (al siguiente, al propietario, al miembro, etc.) los caminos para recorrer la red recuperando registros son diversos y por tanto los datos obtenidos forman una cadena de eslabones con estructura variable. Aunque conceptualmente cada nodo de la estructura de red representa un archivo, físicamente el administrador de la base de datos puede reunir varios nodos en un mismo archivo, mejorando así el desempeño de la base[48]

El **enfoque relacional** al carecer de "rutas de acceso" obligadas por un esquema jerárquico o construcciones de conjunto que relacionan propietario y miembros remiten al manejador de base de datos la obligación de mantener validez en las relaciones entre tablas a través de sus campos conectores, por esta característica se aplican principalmente en sistemas de información[49] en los que las necesidades de recuperación cambian fácilmente, sin embargo cabe aclarar que un diseño cuidadoso optimizará ciertas operaciones de recuperación. Cuando una operación de consulta implica recuperar varios registros el resultado será una tabla nueva.

---

[48]Tsai, Alice, *Sistemas de Bases de Datos*, p375

[49]Vid, *Sistemas de Información*, secc. 1.1.1

En general, debido a la duplicidad de ciertos campos existirá un grado considerable de redundancia en bases relacionales y debe ponerse especial cuidado al actualizar campos conectores para no perder integridad en la información.

REFERENCIAS

Gio Wiederhold
Diseño de Bases de Datos
McGraw Hill
México, Febrero 1991
921p

Shakuntala Atre
Técnicas de Bases de Datos
Trillas
México, 1988
439p

Alice Y. H. Tsai
Sistemas de Bases de Datos
Prentice-Hall Hispanoamericana
México, Junio 1990
607p

J. Fitzgerald, A. F. Fitzgerald, W. D. Stallings Jr.
Fundamentos de Análisis de Sistemas
CECSA
México, Febrero 1990
558p

Jorge L. Euan A., Luis G. Cordero B.
Estructuras de Datos
UNAM
México, 1988
220p

C. J. Date
Introducción a los Sistemas de Bases de Datos
SITESA
México, 1986
648p

James Martin
Principles of Data Base Management
Prentice-Hall Inc., 1976
325p

Benet Campderrich
Técnicas de Bases de Datos
Editores Técnicos Asociados
México, 1988
400p

VAX Staion 3100 - Customer Hardware Information Planning and Preparation - Owner's Manual
Digital Equipment Corp.
Maynard, Massachussets
Enero, 1989

Lawrence J. Kenah, Ruth E. Goldenberg, Simon F. Bate
VAX/VMS Version 4.4
Internals and Data Structures
Digital Press, 1988

CAPITULO 2

ESTUDIO SOBRE LA INTERFAZ HOMBRE-MAQUINA

2.1. PLANTEAMIENTO DEL PROBLEMA

2.1.1. La Central Termoeléctrica de Gómez Palacio

Proceso de generación

La Central Termoeléctrica de Gómez Palacio, Durango es una central generadora de energía eléctrica del tipo ciclo combinado, ubicada en la región de generación Centro-Norte. Ocupa una superficie de 15 hectáreas y fue puesta en servicio oficialmente en Septiembre de 1976, por sus características es la segunda en su tipo que se instala en el país. Se dice que es de *ciclo combinado* pues en una misma planta encontramos el ciclo correspondiente a una central termoeléctrica de gas y el de una de vapor. De hecho en la CCC (Central de Ciclo Combinado) de Gómez Palacio existen dos turbinas de gas (TG1 y TG2) y una turbina de vapor (TV) cada una con su respectivo generador eléctrico.

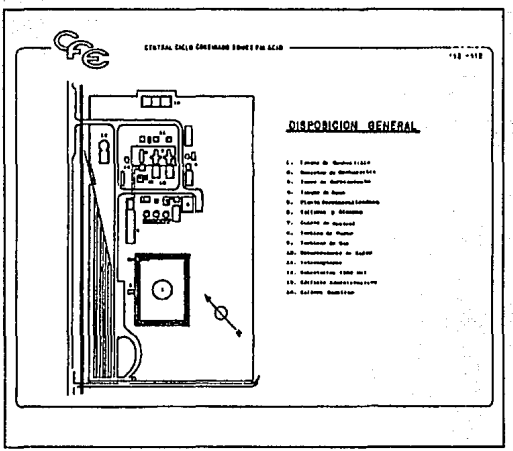


Figura 2.1
Central de ciclo combinado de Gómez Palacio: Disposición general

A continuación se explica a grandes razgos el funcionamiento de la planta:

Comenzando con el ciclo de la turbina de gas, tenemos que el combustible (diesel o gas) es bombeado a presión hacia la cámara de combustión donde se forma una mezcla explosiva con aire, la expansión de los gases ejerce presión en los álabes de la turbina originando energía mecánica la cual es posteriormente transformada en eléctrica mediante un generador con capacidad de 94400 KVA.

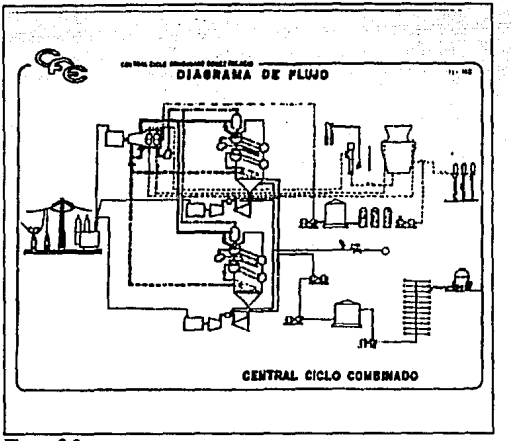


Figura 2.2
CCC Gómez Palacio: Diagrama de flujo general

A la salida de la turbina de gas, los gases después de haber pasado por 4 niveles de álabes dentro de la misma, aun conservan una temperatura relativamente alta (de 400 a 500 °C); con objeto de aprovechar esta energía, los gases se conducen a través de un ducto que va hacia el recuperador de calor (HRSG: "Heat Recovery Steam Generator") y justo a la entrada del mismo para elevar aun más su temperatura existen 8 quemadores alimentados con gas natural.

Ya dentro del recuperador, se realiza un proceso de intercambio de calor entre los gases y el evaporador de alta presión que consiste de una serie de tubos con superficie de calentamiento efectiva de mas de 9900 m^2. El agua contenida en estos tubos acaba por cambiar de fase hasta convertirse en vapor saturado, el cual es conducido a la turbina de vapor que está acoplada mecánicamente con el rotor de un generador con capacidad de 130000 KVA.

Resumiendo, con la combinación de ambos ciclos, se aprovecha gran parte de la energía que todavía tienen los gases de escape de TG1 y TG2 para ayudar en la producción de vapor en los recuperadores de calor, vapor necesario para hacer trabajar la TV.

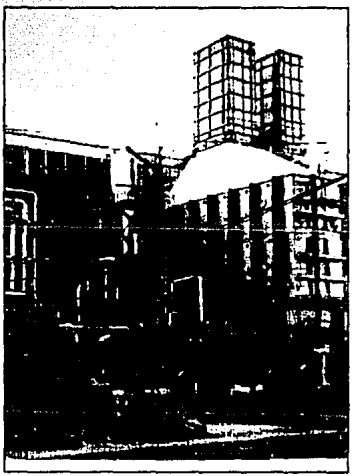


Figura 2.3
CCC Gómez Palacio: Recuperador de calor

Control del proceso

Ciertamente la generación de electricidad mediante sistemas de ciclo combinado conformados por turbinas de gas, recuperadores de calor y turbinas de vapor representan una opción más eficiente que otros métodos pero, paralelamente, se incrementa la complejidad del control de la planta y el número de variables a manejar. En el cuarto de control de la CCC Gómez Palacio fueron instaladas originalmente dos computadoras digitales, una con funciones de monitoreo y otra dedicada al control, siendo la interfaz hacia los operadores a través de tableros de alarmas, de monitoreo y control para cada turbina y equipo de generación. Después de más de 10 años de servicio el ciclo de vida de este sistema está llegando a su fin práctico, no solo por que gran parte del equipo es analógico sino principalmente porque las partes de repuesto y el servicio a los equipos es cada vez mas difícil de conseguir; además el desarrollo de nuevas tecnologías de hardware y software exige su modernización.

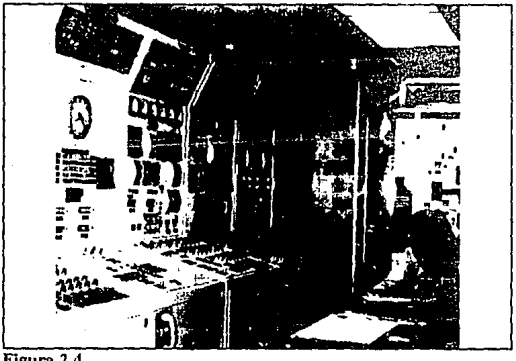


Figura 2.4
Cuarto de control: Tablero analógico

El IIE tomó a su cargo el diseño y puesta en marcha de un nuevo sistema de control digital distribuido que tendrá como efecto positivo la automatización de tareas rutinarias y directas en campo; sobresaliendo en el aspecto digital la manipulación y monitoreo del proceso a través de una interfaz hombre-máquina del tipo GUI ("Graphical User Interface") con respuesta en tiempo real y, en el distribuido la posibilidad de que cualquier estación de trabajo sea capaz de tomar las actividades dejadas por otra en caso de descompostura.

La topología general del sistema de control distribuido propuesto se divide en dos subsistemas:

Subsistema de Presentación:
Cuyos elementos principales son las cinco estaciones de trabajo VAX Station 3100 enlazadas via Ethernet.

Subsistema de Control:
Sus elementos principales son *canastas* con tarjetas de la línea SAC[1], desarrollo propio del IIE.

[1] *Sistema de Adquisición y Control*

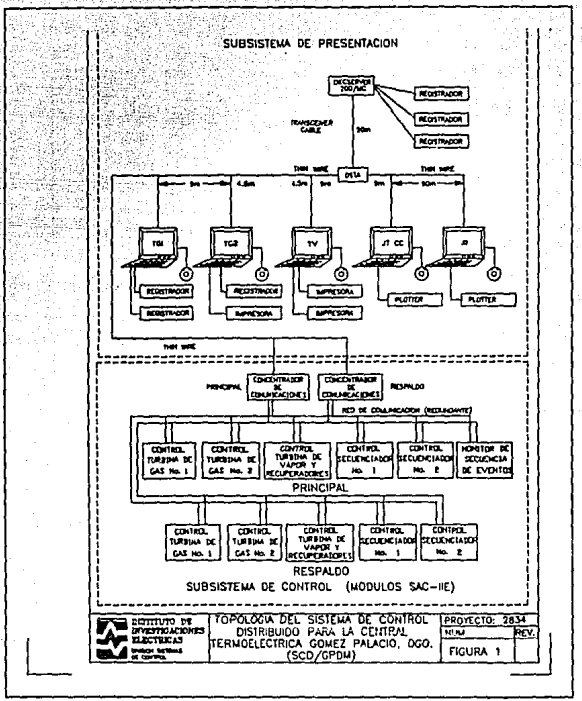


Figura 2.5
Topología del SCD

El flujo de la información a lo largo de los elementos expuestos en la gráfica anterior es el siguiente:

De campo se obtienen señales eléctricas de voltaje y corriente producidas por sensores de temperatura, presión, posición, etc. Estas señales llegan en rangos muy variados y por tanto se *acondicionan* o trasladan a un rango de cero a diez volts.

Después de los acondicionadores y a través de un tablero de interposición las señales se alambran a su canasta, tarjeta y pin correspondiente. En sentido opuesto, las canastas envían a actuadores en campo señales o comandos cerrando así el ciclo de control.

Las canastas son módulos con fuente integrada y un trasplano en el que se insertan tarjetas SAC, cada canasta cuenta con una tarjeta procesadora (microprocesador 8088 y programas descargados en memorias EPROM) tarjeta de comunicaciones, y las tarjetas de entradas o salidas analógicas o digitales necesarias. Las canastas se programan para adquirir señales y controlar un equipo específico de la planta, de ahí que se encuentren duplicadas y pueda activarse el respaldo en caso de falla.

Las canastas se comunican entre sí por medio de una red half duplex serial estándar RS-485 que es controlada por el CC (concentrador de comunicaciones o "maestro") activo; además en un esquema por excepción las tarjetas envían al CC aquellas señales adquiridas por las tajetas SAC que hayan cambiado y el maestro a su vez responde con mensajes de las variables que el operador manipula en su estación de trabajo.

Los CC son el puente de unión entre ambos subsistemas y se trata de computadoras industriales del tipo AT con tarjeta Ethernet y serial RS-485; son pues los traductores entre ambos protocolos.

Figura 2.6
Cuarto de control: Estación de trabajo

Por el lado de las estaciones de trabajo, cada una tiene bajo el ambiente VAXELN una presentación basada en ventanas, los programas de aplicación y una réplica de la base de datos misma que será actualizada constantemente con los datos enviados por el CC; los valores de inicialización para esta base se obtienen a partir de los capturados en una PC. Es importante acotar

que si bien la parte gruesa del control se lleva en las tarjetas SAC, los programas que corren en las estaciones de trabajo no son únicamente del tipo supervisorio, es decir que en su ambiente por ventanas el operador podrá activar tanto módulos informativos (p.ej. alarmas) como diálogos de control.

2.1.2. Requerimientos de diseño para la BD

El diseño de toda base de datos implica un análisis de las necesidades de los usuarios de la misma, en este caso tenemos dos tipos de usuarios:

* Programadores
* Operadores

Los programadores necesitan saber como obtener datos sobre variables para poder procesarlas, presentar información sobre ellas y actualizarlas.

Los operadores necesitan de un procedimiento en línea para modificar los atributos de las variables.

Antes de pasar al planteamiento de estas necesidades, clasificaremos desde el punto de vista funcional a las protagonistas del sistema: Las variables.

Variables a manejar

1) ADQUIRIDAS

Analógicas Adquiridas
Lógicas Adquiridas

Son variables que corresponden a señales de entrada provenientes de campo y cuyo propósito fundamental es monitorear el comportamiento de los equipos. p.ej. 'Temperatura de gases al evaporador', 'Posición de álabes móviles turbina', etc.

2) CONTROL

Control Entradas Analógicas
Control Salidas Analógicas
Control Entradas Lógicas
Control Salidas Lógicas

Se trata de variables relacionadas con acciones de control, un cambio en su valor implica comandos hacia campo y realimentación sobre el resultado del comando hacia el sistema. p.ej. 'Abrir válvulas de combustible', 'Válvulas de drenaje cerradas'.

3) CALCULADAS

Eficiencias
Flujos
Horas de Operación
Número de Arranques
Analógicas Calculadas
Lógicas Calculadas

Son variables no adquiridas cuyo propósito es servir de base para estadísticas sobre la operación de los equipos de planta. p.ej. 'Eficiencia diaria tubina de vapor', 'Consumo diario de combustible TG1'.

4) INTERNAS

Analógicas
Lógicas

Son variables de uso netamente interno y aunque en la interface tengan asociado algún elemento gráfico (un botón, selector de posición, etc.) de una ventana, el operador ignora su existencia. p.ej. 'Botón paro emergencia TG1', 'Selector de sincronismo en auto'.

5) CONSTANTES

Constantes en canasta

En los programas de control que corren en canasta existen ciertas constantes intrínsecas al proceso, su modificación en muchos casos altera radicalmente el desempeño de la planta. p.ej. 'Punto de ajuste max quemadores posteriores', 'Tiempo max arranque TG1'.

6) DIAGNOSTICOS

Diagnóstico de Equipo SAC
Diagnóstico de Equipo de Cómputo

Contienen el estado operativo de la parte hardware del sistema de control distribuido. p.ej. 'Diagnóstico tarjeta 4 canasta 3', 'Software MTOS canasta 5'.

7) CONFIGURACIONES

Configuración de Tarjetas SAC

Indican la configuración de las tarjetas SAC, de entradas, salidas analógicas o digitales.

Tratamientos a las variables

1) Ya se había mencionado[2] que el sistema IHM instalado correrá bajo el ambiente VAXELN que es una herramienta (*toolkit*) especial para el desarrollo de aplicaciones en tiempo real, en VAXELN pueden correr concurrentemente varios procesos con una limitante fundamental **SOLO UN PROCESO PUEDE ESCRIBIR EN UN AREA DE MEMORIA COMPARTIDA POR VARIOS PROCESOS[3]**, es responsabilidad del programador no violar esta regla cuando intente modificar directamente atributos de una variable que pueden ser accesados para escritura por otros programas, por ello se le deben proporcionar los elementos adecuados.

2) El valor de todas las variables analógicas se guardará en dos bytes (el atributo VALOR ANALOGICO) tomando en cuenta únicamente los 12 bits menos significativos; la razón fundamental es que el valor de dichas variables enviado por las canastas se recibe en ese formato, para convertir este valor a su equivalente en flotante el programador deberá aplicar la ecuación de *linearización* adecuada, comúnmente (mas del 98% de las variables) consiste simplemente en una recta. Partiendo que el valor 0x000 corresponde al *límite de operación bajo* y 0xFFF al *límite de operación alto* se puede aplicar la ecuación de punto pendiente para la recta:

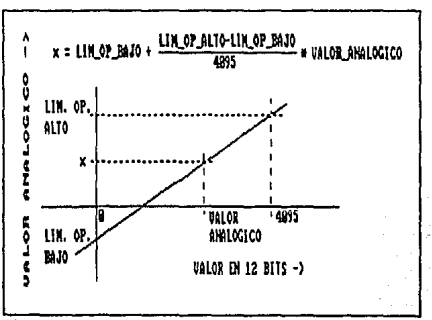


Figura 2.7
Conversión de 12 bits a flotante

3) Las variables lógicas pueden considerarse alarmadas cuando su estado pasa de 0 a 1 lógico y/o viceversa según el caso.

---

[2]Vid sección 1.1.2, *Sistemas Operativos*

[3]VAXELN Vol. 3, *Runtime Facilities Guide*

4) Las variables analógicas tienen estado normal, alarmado precrítico y alarmado crítico de acuerdo a la relación que exista entre su valor y los límites críticos y precríticos definidos para ella.

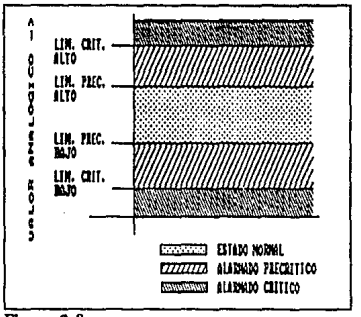


Figura 2.8
Límites críticos y precríticos

5) Es común que el rendimiento máximo de un equipo se encuentre gráficamente cerca de sus límites de operación normal, justo cuando sus variables se encuentran bajo el límite crítico alto; para mantener al equipo lo más cerca de ese estado óptimo, se aplica el concepto de *"Banda Muerta"* a sus variables; la banda muerta se calcula aplicando un porcentaje a los límites críticos y precríticos e indica al sistema que una vez alarmada la señal, debe mantenerse en ese estado hasta que su valor salga de la región comprendida por la banda. Por ejemplo, supongamos que una señal de temperatura que recibe el sistema tiene 2% de banda muerta y su nivel para alarma crítico alto es de 1000 °C, entonces, cuando la señal sobrepase este valor se alarmará y a diferencia del tratamiento tradicional se mantendrá alarmada hasta que disminuya a 1000 °C * (100 - 2)% = 980 °C; el resultado operativamente se manifestará como ligeras variaciones de 20 °C entre estos puntos.

6) Algunas variables analógicas para su control requieren de la aplicación de un límite especial, el *límite dinámico*; que se calcula aplicando un porcentaje a los límites de operación normal, es decir, a los límites precríticos. Cuando la variable sobrepasa los límites dinámicos puede o no según el caso, considerarse como alarmada.

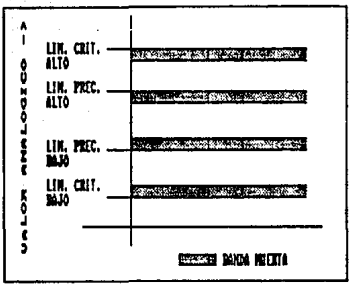


Figura 2.9
Ubicación de bandas muertas

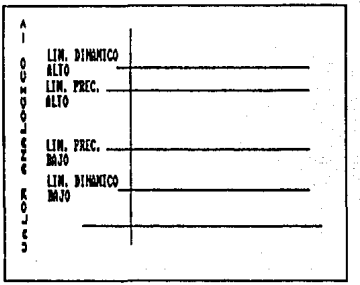


Figura 2.10
Límite dinámico

7) El cálculo de variables de consumos, horas de operación, etc. se hace mediante un procedimiento conocido como *"integración"*; en el cual es necesario tener una estructura lineal para cada variable con los siguientes datos:

* Factor de Integración (FI)
* Periodo de Muestreo (PM)
* Tiempo Transcurrido (TT)
* Tiempo de Integración Máximo (TM)
* Cuenta Corriente (CC)
* Cuenta Máxima (CM)

* Ultima Cuenta (CU)
* Valor de la Variable (VV)

El algoritmo que permite mantener siempre en CU la última suma de los valores multiplicados por el factor FI de una variable (VV) muestreada cada PM durante un máximo de TM segundos o hasta alcanzar el valor maximo de CM es el siguiente:

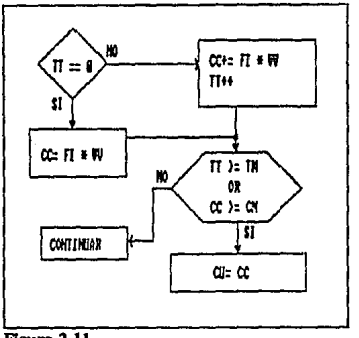


Figura 2.11
Algoritmo para realizar "integraciones"

Para acabar de ilustrar este concepto, se expone el siguiente ejemplo: supongamos que cada segundo (PM= 1) se adquiere la temperatura de un elemento de la planta y se desea obtener el promedio de esta variable durante 60 segundos (FI= 1/60, TM= 60), sabiendo que el valor máximo posible para la temperatura antes de considerarse fuera de rango es de 4500; cada renglón de la siguiente tabla, contiene los valores que segundo a segundo podrían tomar los datos dado un valor supuesto para VV. Nótese como en letra negrita se marcaron los datos constantes.

FI	PM	TT	TM	CC	CM	CU	VV
1/60	1	0	60	47	4500	3250	2820
1/60	1	1	60	93	4500	3250	2760
1/60	1	2	60	145	4500	3250	3120
1/60	1	3	60	199	4500	3250	3240
--	-	-	-	-	-	-	-
--	-	-	-	-	-	-	-
1/60	1	58	60	2980	4500	3250	?
1/60	1	59	60	3031	4500	3250	3060
1/60	1	0	60	48	4500	3031	2880

Necesidades de los módulos de la IHM

La interfaz Hombre-Máquina se divide conceptualmente en dos partes, cada uno con submódulos, objetivos y necesidades particulares:

1) Programas de infraestructura

ARRANQUE DEL SISTEMA: En este submódulo se arrancan y sincronizan el resto de los programas de infraestructura; en cuanto a la base de datos se deberán *inicializar los descriptores* para los procedimientos de alto nivel.

ADQUISICION DE SEÑALES: Se trata del programa que enlaza a las estaciones de trabajo con la realidad de campo; por medio de mensajes recibe y envía de/hacia los concentradores de comunicaciones la *"dirección hardware"* (*canasta, tarjeta, bit*) junto con la información adecuada sobre variables cuyos atributos valor *lógico* o *analógico*, *ganancia*, y *límites de operación* hayan cambiado, accesando para ello la base de datos tanto para escritura como para lectura.

ALARMAS: Recibe las variables cuyo valor ha cambiado y aplicando los conceptos de *límites críticos*, *banda muerta*, etc. determina el estado de cada una para asignarle un *color*, atributo muy importante ya que debe reflejarse en todas las pantallas de funciones de presentación donde se presente la variable; además si la variable resulta alarmada envía su *dirección hardware* a la FP (Función de Presentación) ALARMAS y al SPOOLER para reportarla como alarmada en la impresora que indique su atributo *imprimir alarma*.

MONITOR DE SECUENCIA DE EVENTOS: La detección de un disturbio de magnitud considerable enviado por la canasta del monitor de secuencia de eventos e identificado por cambios en variables de campo que son la causa de un disparo, activa la generación de un reporte impreso y la escritura en disco de la secuencia de los cien eventos más recientes, con lo cual se podrá determinar con mayor exactitud la causa del disturbio. Lee la *dirección hardware*, el *valor de la variable* y los *límites crítico y precrítico*.

HISTORIAS: Guarda en disco en una lista circular hasta los últimos 1024 valores que ha tenido una o un grupo de variables, muestreadas cada cierto tiempo, el programa se relaciona con GRAFICAS DE TENDENCIAS, GRAFICAS EN PAPEL, y REPORTES manteniendo varios archivos históricos; no hace escrituras a la base de datos pero lee frecuentemente *valor analógico*, *valor lógico* y *dirección hardware* de la variable.

CALCULOS ESPECIALES: Trabaja principalmente sobre variables analógicas calculadas o deriva información a partir de las adquiridas aplicando algoritmos tales como el proceso de integración antes descrito, para hacer posible y facilitar la presentación de información organizada estadísticamente. Por tanto, escribe y lee el *valor analógico* de las variables y los elementos de la *estructura de integración* mencionada en secciones anteriores.

SPOOLER: Recibe y encola información que se debe imprimir, enviada por los módulos de ALARMAS, MONITOR DE SECUENCIA DE EVENTOS Y REPORTES.

TOLERANCIA A FALLAS: Detecta problemas de funcionamiento en las estaciones de trabajo, verificando constantemente que estén activas; cuando por cualquier razón alguna de ellas sale de servicio todas se enteran de la situación y si en ella se corría CONTROL COORDINADO, el módulo pasa a encargo de otra estación. En relación a la base de datos, cuando la estación de trabajo regresa a servicio se envían mensajes para *actualizar su propia base*.

ADMINISTRADOR DE LAS FUNCIONES DE PRESENTACION: Para la presentación de información en pantalla, el IIE adquirió un editor gráfico que aplica la filosofía de programación orientada a objetos, el SL-GMS[4], con este editor es posible ligar propiedades de objetos gráficos (color, posición, etc.) con valores de variables declaradas en los programas; para ello, es necesario "definirle" *todos los atributos* de las variables que se usan en los diferentes modelos y esa es, precisamente, la primer tarea de este programa de infraestructura. Posteriormente, queda corriendo para abrir ventanas y cargar modelos de las funciones de presentación que el operador desee activar.

2) Funciones de presentación

LISTA DE VARIABLES: Su objetivo es permitir la consulta de las variables organizadas por sistema y subsistema, mostrando en el *color* de la variable los atributos *identificador, descripción, valor, unidades de ingeniería, límites de operación alto y bajo*. Requiere de un método de acceso (un índice) por *sistema y subsistema*.

DIALOGO DE ALARMAS: Muestra en su ventana por grupos de 20 renglones hasta 100 variables que hayan sido alarmadas, mantiene *dos índices*: *cronológico y por prioridad*; el orden de presentación es seleccionado por el operador. En su desplegado, muestra en el *color* adecuado la hora en que se alarmó cada variable, *identificador, descripción, valor, unidades de ingeniería, límite rebasado* y *prioridad* de la misma.

GUIAS EN CASO DE ALARMAS: Pasando por un menú de sistemas donde los nombres de los mismos tendrán color normal o alarmado dependiendo del estado de sus variables principales, se selecciona alguna de las guías disponibles, que proporcionará al operador la información necesaria para el manejo de situaciones que en el sistema específico involucran estados alarmados. Hace muy poco uso de la base de datos, solo lee *nombres de sistemas* y *subsistemas* y *colores* de variables. Requiere acceso por *sistema y subsistema*.

REPORTES: Genera los siguientes listados;

* Valores históricos guardados en disco por HISTORIAS
* Flujos acumulados: diario, mensual o anual
* Eficiencias

---

[4]Sherrill-Lubinski, Graphical Modeling System

* Variables que no se muestrean
* Horas de operación de los equipos principales
* Estado de la planta
* Número de arranques de equipos principales

En los reportes los atributos requeridos son *identificador*, *descripción*, *valor*, *unidades de ingeniería* y según el caso la *última cuenta* de integración[5].

ARRANQUE Y PARO: Presenta información de las variables relacionadas con los procesos de arranque y paro de los equipos principales organizados por sistema y subsistema; para cada uno se muestran diagramas de compuertas lógicas y también puede monitorearse el valor de variables mediante diagramas de barras y gráficas de tendencias, por tanto requiere para lectura los atributos *identificador*, *descripción*, *color*, *sistema*, *subsistema*, *límites críticos y precrítico*, *unidades de ingeniería*.

GRAFICAS DE COORDENADAS: De la selección de un par de variables analógicas tomada de un conjunto de parejas ya determinado, se presenta en un plano coordenado la curva típica que relaciona dichas variables en la forma $y = f(x)$ y el punto que corresponde a sus valores actuales, esto es con el fin de hacer una comparación entre el comportamiento real y el esperado. Lee de la base de datos *color*, *valor analógico*, *descripción* y *unidades de ingeniería*.

DIAGRAMAS DE FLUJO: Presenta a los operadores diagramas de flujo de la planta *por sistema y subsistema* con objeto de determinar el estado operativo y funcional de los mismos, los objetos gráficos de los diagramas según el caso tendrán animación y/o cambiarán de *color* dependiendo del estado de la variable asociada a ellos, en muchos casos se muestra el *valor analógico* con sus respectivas *unidades de ingeniería*.

DIAGRAMAS DE BARRAS: Presenta al usuario el valor en tiempo real en forma de barra de las variables analógicas, agrupadas de ocho en ocho, el *color* de la barra de cada variable cambia dependiendo del estado de la misma, (rojo alarma crítica, amarillo alarma precrítica, azul estado normal), requiere para lectura varios atributos, *descripción*, *unidades*, *límites*, y por supuesto, *valor analógico*.

GRAFICAS DE TENDENCIAS: Muestra en gráficas que se actualizan en tiempo real o tomando datos de archivos históricos los valores de grupos de variables previamente definidos contra el tiempo; al igual que DIAGRAMAS DE BARRAS lee los atributos *descripción*, *unidades*, *límites*, *valor analógico* y *color*.

GRAFICAS EN PAPEL: Permite al usuario seleccionar las variables cuyo comportamiento desea analizar mandando su valor adquirido a alguno de los registradores conectados al sistema. Hace uso de *valor analógico* y los *límites de graficación en papel* de la variable.

---

[5] Vid sección 2.1.2, *Tratamientos a las Variables*

DIALOGO DE CONTROL: Su objetivo es proporcionar al operador elementos gráficos (botones, selectores, etc.) para que con el uso del track-ball mande comandos de control a los equipos principales de planta: Turbina de Vapor y Turbinas de Gas 1 y 2; por tanto, lee y escribe en la base *valores lógicos* y *analógicos*.

CONTROL COORDINADO: Trabaja en un nivel superior al de DIALOGO DE CONTROL ya que se encarga de enviar la serie de comandos necesarios a fin de coordinar la actividad de los equipos principales distribuyendo la carga y conseguir asi una demanda determinada, no se considera como módulo de infraestructura pues aunque algunos valores deben calcularse antes de ser enviados, la *parte* gruesa de los cálculos la realiza una canasta de la línea SAC.

DIALOGO DE SERVICIOS: Permite al operador dar de alta y baja variables y modificar valores de atributos, es la función de presentación que da mantenemiento a la base de datos.

3) Requerimientos generales

Dada la experiencia que se tiene en el IIE en el uso del sistema operativo VMS y del VAXELN para la construcción de aplicaciones en tiempo real, estos son respectivamente los ambientes de desarrollo y de ejecución para la IHM; el VAXELN permite multitareas, multiprogramación y multiprocesamiento bajo un esquema de desplazamiento de procesos por prioridad[6], siendo asi, y como los accesos a la base de datos por procesos diferentes son muy frecuentes, el control de escrituras y lecturas no se dejó al cargo de un proceso o programa que tomara el control del CPU cada vez que fuese llamado sino que se diseñaron dos niveles de acceso que el programador liga a su módulo como veremos a continuación.

Para cualquier atributo, los programadores deben contar con la posibilidad de accesar directamente y de la forma más rápida la información partiendo de la clave primaria "Dirección Hardware"; esto es, a través de un índice de acceso aleatorio de no mas de un nivel. Esta opción se aplica en algoritmos estrechamente involucrados con procesos en tiempo real y la coordinación de acceso para escritura necesaria por el problema expuesto en el punto (1) de la sección sobre Tratamiento a las Variables queda a responsabilidad del programador.

A nivel superior, el programador requiere de una serie de herramientas, una biblioteca de funciones que le sirvan para manipular la base de datos a través de descriptores de campo, registro e índice; estos procedimientos aunque no estén optimizados para una situación particular, reducen el esfuerzo de programación en algoritmos donde se hace un tratamiento estadístico o se manipulan bloques de datos. Y lo que es más importante, estandarizan el acceso garantizando la consistencia de la información. En este nivel incluso se deben simplificar las relaciones lógicas entre dos tipos de registro lo cual se logra estableciendo una relación física por medio de apuntadores o campos tipo *"reference"* de modo que el programador cuando solicite el valor de un campo de este tipo obtiene directamente el valor del campo apuntado.

---

[6]Vid sección 1.1.2, *VAXELN*

Para coordinar escrituras se usan MUTEX (objeto del VAXELN) por cada tipo de variable, los MUTEX son semáforos optimizados que sirven para evitar que dos procesos traten de escribir en la misma área de memoria, una vez declarados se usan de la siguiente forma:

```
ELN$LOCK_MUTEX(objeto_mutex);
  acción de escritura;
ELN$UNLOCK_MUTEX(objeto_mutex);
```

El lenguaje de programación elegido para el desarrollo de la IHM fue 'C', esto quedó plenamente justificado por que aunque se contaba con bibliotecas de VAXELN para Pascal y C en este último lenguaje es posible manejar directamente variables tipo bit con el consecuente ahorro de memoria al declarar la base de datos que como veremos contiene bastantes atributos de ese tipo, además, el código del editor gráfico GMS está escrito originalmente en C asi que la interfaz con el mismo es más directa.

Dado que se trata de una aplicación en tiempo real, y que se dispone de suficiente memoria RAM (16 MBytes por estación de trabajo) los registros de variables se mantendrán en RAM para acelerar su acceso; al arranque del sistema se cargará de archivos en disco cada tipo de variable a memoria, la actualización de la información en los archivos será diferida (cada 30 seg para las variables analógicas calculadas). Sin embargo existen algunos módulos (DIALOGO DE SERVICIOS) que requieren la actualización inmediata de la información correspondiente a una variable específica en disco.

2.2. ANALISIS DE LA INFORMACION DEL SISTEMA

Para llegar a un modelo conceptual de la base de datos a diseñar, tenemos que organizar el universo de datos que debe visualizar el sistema de control distribuido en entidades y determinar los atributos válidos para ellas.

2.2.1. Entidades

En principio y dado que la base de datos debe responder a un sistema en tiempo real, dividimos las entidades en dos grandes grupos: *entidades que representan variables* afectadas en tiempo real y son por tanto las más importantes y *entidades auxiliares*.

Entidades de variables

Definimos las entidades principales o de variables a partir de la exposición hecha en la sección anterior sobre las variables que manejará el sistema, haciendo una reagrupación de acuerdo a características comunes para los diferentes tipos de variable:

```
1) ANALOGICAS:
        Su valor se representa en 2 bytes.
        Se alambran a campo.

     Analógicas de adquisición
     Entradas analógicas de control
     Salidas analógicas de control
     Analógicas internas

2) LOGICAS:
        Para representar su valor basta con un bit.
        Se alambran a campo.

     Lógicas de Adquisición
     Entradas lógicas de control
     Salidas lógicas de control
     Lógicas internas
     Entradas de secuenciador
     Salidas de secuenciador
     De secuenciador internas
```

3) ANALOGICAS_CALC (Analógicas Calculadas):
Representar su valor requiere de 2 bytes.
No se alambran a campo.

Analógicas calculadas
Flujos acumulados
Horas de operación
Número de arranques
Eficiencias

4) LOGICAS_CALC (Lógicas Calculadas):
Basta un bit para representar su valor.
No se alambran a campo.

Lógicas calculadas

5) CONSTANTES:
Requieren de 2 bytes para representar su valor.
No se alambran a campo.

Constantes en canasta

6) DIAGNOSTICOS:
Codifican buen estado o malo, un bit.
No se alambran a campo.

Diagnóstico de equipo SAC
Diagnóstico de equipo de cómputo

7) CONFIGURACIONES:
Su valor en 2 bytes.
No se alambran a campo.

Configuración de tarjetas SAC

Entidades auxiliares

Estos tipos de registro se determinaron considerando por un lado, que existen características comunes que clasifican a las variables en grupos (p.ej. podemos hablar de las *variables del sistema Turbina de Vapor* o de las *variables con unidades de ingeniería en volts*, etc) y con objeto de evitar redundancia tales características no se deben repetir explícitamente en cada registro, por otro lado, hay otras entidades que se relacionan de modo indirecto y conviene verlas independientemente.

1) UNIDADES:

Unidades de ingeniería en que está dado el valor de una variable analógica.

2) ESTADOS:

Estados que dan significado a los valores *cero* y *uno* de una variable lógica; el *estado* de una variable es para las variables lógicas como lo son las unidades de ingeniería para variables analógicas.

3) ACCIONES:

Para las salidas lógicas de control en lugar de considerar a la variable en un *estado*, se dice que existe una *acción de control* asociada.

4) SISTEMAS:

Nombres de los sistemas.

5) SUBSISTEMAS:

Nombres de los subsistemas.

6) LINEARIZACIONES:

Contiene referencias a algoritmos necesarios para linearizar las variables adquiridas de campo.

7) PROGRAMAS:

Identificación de programas residentes en disco que se asocian con ciertas variables de control y que deben ejecutarse según las condiciones del sistema.

8) INTEGRACIONES:

Es un tipo de registro independiente sobre el que se trabaja para obtener 'acumulaciones', 'promedios' o integraciones de variables analógicas.

9) CANASTAS:

Conserva características funcionales de las canastas.

2.2.2. Atributos

A continuación se detallan las características de cada atributo o campo, abordando principalmente aspectos conceptuales pero mencionando también algunos relacionados con el almacenamiento interno[7] como son tipo y rango. Los atributos se han agrupado en dos grandes conjuntos; atributos propios de entidades que representan variables afectadas en tiempo real y atributos encontrados en entidades auxiliares.

Para cada atributo además de su descripción y utilidad se mencionan las relaciones que existen entre este y los demás, entidades a las que es aplicable, el nombre-identificador reconocido por procedimientos de alto nivel, el nombre empleado para su declaración en estructuras y que el programador usa para trabajar directamente con la base y el tipo y rango.

[7]Vid sección 1.4.3, *Diccionario de Datos*

Atributos de variables

1) REGISTRO BLOQUEADO

a) Identificador

PUNTO_BLOQUEADO

b) Nombre

Punto_bloqueado

c) Descripción

Dado que el operador interactuará con el sistema a través de ventanas, es posible tener activas dos de ellas con la misma función de presentación ya sea en dos estaciones de trabajo diferentes o en la misma; en el caso de Diálogo de Servicios es necesario un mecanismo para evitar que dos operadores intenten modificar al mismo tiempo los atributos de una variable lo que se consigue haciendo que PUNTO_BLOQUEADO funcione como semáforo.

d) Uso

Al arranque del sistema en todos los registros se inicializa en cero, posteriormente solo la FP Diálogo de Servicios cuando va a modificar un registro determinado verifica si el campo está en cero y de ser asi lo pone en uno, realiza la escritura y lo vuelve a dejar en cero. Si al hacer la verificación del valor de PUNTO_BLOQUEADO se encuentra que está en uno, eso quiere decir que otra ventana con Diálogo de Servicios ya está escribiendo en ese registro.

e) Atributos relacionados

Conceptualmente para la FP Diálogo de Servicios PUNTO_BLOQUEADO es el candado de acceso a la escritura de todos los demás campos de una entidad.

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC, CONSTANTES, DIAGNOSTICOS, CONFIGURACIONES

g) Tipo y rango

Un bit, (0 ó 1)

2) SCAN INHIBIDO

a) Identificador

SCAN_INH

b) Nombre

Scan_inhibido

c) Descripción

Indica si el valor de la variable se está actualizando (0) o no (1). Este campo es útil no solo en entidades de variables adquiridas de campo sino también en variables internas ya que una variable adquirida genera con frecuencia varias internas y asi el operador podrá inhibir la actualización de una interna sin afectar la actualización de la proveniente de campo.

d) Uso

SCAN_INH será modificado por el operador a través de la FP Diálogo de Servicios siendo por tanto muy baja la frecuencia de escritura, por el contrario, un proceso antes de alterar el valor de una variable, debe verificar su estado, asi que la frecuencia de lectura será muy alta. Por otro lado, el campo es muy importante, un mal uso del mismo puede dejar al sistema incomunicado con la realidad en campo.

e) Atributos relacionados

Conceptualmente, SCAN_INH se relaciona como candado de acceso a escritura para VALOR_LOGICO o VALOR_ANALOGICO según el tipo de registro.

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC, CONSTANTES, DIAGNOSTICOS, CONFIGURACIONES

g) Tipo y Rango

Un bit, (0 ó 1)

3) ALARMA INHIBIDA

a) Identificador

ALARMA_INH

b) Nombre

Alarma_inhibida

c) Descripción

Mediante este atributo se indica si una variable es susceptible de alarmarse (0) o no (1), es decir que aún cuando el estado de una variable pase a alarmado, dicho estado solo se reportará cuando ALARMA_INH sea cero.

d) Uso

La FP Diálogo de Alarmas verifica el estado de ALARMA_INH antes de considerar a un registro dentro de su desplegado o mandarlo a impresión. El operador a través de la FP Diálogo de Servicios podrá inhibir o desinhibir alarmas. Los cambios a este campo deben hacerse cuidadosamente para no sacar del contexto del Diálogo de Alarmas a una variable de importancia.

e) Atributos relacionados

Conceptualmente, da validez a VAR_ALARMADA.

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC

g) Tipo y Rango

Un bit, (0 ó 1)

4) VARIABLE ALARMADA

a) Identificador

VAR_ALARMADA

b) Nombre

Bandera_alarma

c) Descripción

En uno, indica que la variable está alarmada, para determinarlo se aplican diferentes criterios de acuerdo al concepto de "Variable Alarmada"[8].

d) Uso

El módulo de Alarmas actualiza el campo si la variable pasa a un estado alarmado. Las FP's lo verifican para determinar el estado de una variable, durante disturbios será accesado frecuentemente.

e) Atributos relacionados

Se relaciona con ALARMA_INH, VALOR_LOGICO, VALOR_ANALOGICO y con los límites críticos y precríticos.

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC

g) Tipo y Rango

Un bit, (0 ó 1)

[8]Vid sección 2.1.2, *Tratamientos a las Variables*

5) EVENTO INHIBIDO

a) Identificador

EVENTO_INH

b) Nombre

Evento_inhibido

c) Descripción

En cero indica que la variable será reportada de manera impresa cuando alguno de sus atributos sean alterados (por ejemplo, los límites críticos) o bien cuando una acción de control se lleve a cabo. Si el valor del campo es uno, no habrá impresión. El propósito de estos reportes es llevar un registro histórico de las actividades mas importantes sobre los equipos de planta y los cambios a las propiedades de sus señales, para en un momento dado determinar una secuencia de eventos y el responsable de ellos.

d) Uso

La frecuencia de escritura y lectura es relativamente baja ya que tanto acciones de control como cambios a valores de atributos se generan gracias a una interacción entre el operador y el sistema.

e) Atributos relacionados

Ninguno

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC

g) Tipo y Rango

Un bit, (0 ó 1)

6) ESTADO FUNCIONAL

a) Identificador

ESTADO_FUNCIONAL

b) Nombre

Estado_funcional

c) Descripción

Cuando una señal proveniente de campo está fuera de su rango de transmisión el sistema de adquisición reporta a la señal como dada de baja funcionalmente. Para las variables calculadas si su(s) variable(s) de origen está(n) fuera de servicio, también ellas estarán en ese estado.

d) Uso

El campo puede ser alterado solo por los módulos de Adquisición y Envío de Mensajes y el de Cálculos Especiales, en la FP Lista de Variables se lee para reportar su estado al operador.

e) Atributos relacionados

Se relaciona, dando validez, con VALOR_LOGICO o VALOR_ANALOGICO según el tipo de registro.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Un bit, (0 ó 1)

7) VALOR DADO POR EL OPERADOR

a) Identificador

VALOR_DADO

b) Nombre

Valor_operador

c) Descripción

. En uno, indica que el valor de la variable (lógico o analógico) fue dado por el operador (para que dicho valor se mantenga en variables adquiridas, SCAN_INH debe estar en uno previniendo asi su alteración).

d) Uso

La frecuencia de uso de este campo es muy baja considerando que los cambios al mismo se realizan por comando explícito del operador.

e) Atributos relacionados

Ninguno

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC, CONSTANTES, DIAGNOSTICOS, CONFIGURACIONES

g) Tipo y Rango

Un bit, (0 ó 1)

8) CALIDAD DEL DATO

a) Identificador

CALIDAD_DATO

b) Nombre

Calidad_dato

c) Descripción

Cuando el valor de una variable sobrepasa sus límites de operación esto es, que por adquisición se determina que el sensor está dando un valor de la variable por debajo o por encima de su rango CALIDAD_DATO se pone en cero, indicando calidad del dato mala.

d) Uso

Al arranque del sistema y hasta determinar la calidad de las señales recibidas, tendrá uno por omisión. Su frecuencia de acceso es baja.

e) Atributos relacionados

Da validez a VALOR_ANALOGICO, se relaciona con LIM_OP_ALTO y LIM_OP_BAJO.

f) Entidades donde se le encuentra

ANALOGICAS

g) Tipo y Rango

Un bit, (0 ó 1)

9) BIT DE BANDA MUERTA

a) Identificador

BANDA_MUERTA

b) Nombre

Banda_muerta

c) Descripción

En uno, quiere decir que el concepto de *"Banda Muerta"*[9] es aplicable a la variable. Es importante hacer notar la diferencia entre este atributo y el de BANDA_MUERTA_PUESTA; el operador podrá modificar para una variable específica este último siempre y cuando BANDA_MUERTA sea uno, es decir siempre y cuando la variable sea susceptible de tener *"Banda Muerta"*.

d) Uso

Se tomará el valor especificado por el campo correspondiente en la base de datos en PC y no podrá alterarse durante el funcionamiento del sistema, BANDA_MUERTA es leído por el módulo de Alarmas para determinar la aplicabilidad de "Banda Muerta" a la variable y según sea el caso, enviar un mensaje de variable alarmada a la FP Diálogo de Alarmas.

e) Atributos relacionados

Da validez a BANDA_MUERTA_PUESTA y PORC_BANDA_MUERTA.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Un bit, (0 ó 1)

---

[9]Vid sección 2.1.2, *Tratamientos a las Variables*

10) BANDA MUERTA PUESTA

a) Identificador

BANDA_MUERTA_PUESTA

b) Nombre

Banda_puesta

c) Descripción

En uno, indica que el concepto *"Banda Muerta"* se debe aplicar en el análisis de variables a enviar a la FP Diálogo de Alarmas.

d) Uso

El operador podrá modificar este campo si BANDA_MUERTA es uno, es decir si el concepto es aplicable a la variable en cuestión, siendo así, el acceso para escritura es bajo pero la solicitud de lecturas será mayor especialmente durante disturbios considerando la cantidad de señales que resultan alarmadas.

e) Atributos relacionados

El campo se relaciona con BANDA_MUERTA y PORC_BANDA_MUERTA, el primero valida y el segundo determina valor.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Un bit, (0 ó 1)

11) ALARMA DE RETORNO A LIMITES DE OPERACION NORMAL

a) Identificador

ALARMA_RETORNO

b) Nombre

Alarma_retorno

c) Descripción

Comúnmente, de acuerdo al concepto de "Variable Alarmada"[10]. una variable se alarma cuando pasa de un estado de alarma inferior a otro superior; este campo en uno indica que debe verificarse una condición mas en el análisis de la variable, alarmándola cuando regrese a sus límites de operación normales es decir dentro de los límites precrítico alto y bajo (y sus bandas muertas si están puestas).

d) Uso

La inicialización de ALARMA_RETORNO viene dada por la base de datos en PC pero puede alterarse por el operador a través de la FP Diálogo de Servicios. Así que la frecuencia de escritura es baja pero la de lectura tendrá picos durante el restablecimiento de disturbios.

e) Atributos relacionados

Se relaciona conceptualmente con VALOR_ANALOGICO, LIM_PRECRITICO_ALTO, LIM_PRECRITICO_BAJO, BANDA_MUERTA, BANDA_MUERTA_PUESTA y PORC_BANDA_MUERTA.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Un bit, (0 ó 1)

---

[10]Vid sección 2.1.2, *Tratamientos a las Variables*

12) BIT DE LIMITE DINAMICO

a) Identificador

LIM_DINAMICO

b) Nombre

Limite_dinamico

c) Descripción

Cuando el concepto de *"Límite Dinámico"*[11] se puede aplicar a una variable, el campo tendrá uno.

d) Uso

El módulo de Alarmas agrega una condición en su análisis de aquellas variables que tengan límite dinámico, enviándolas como alarmadas a la FP Diálogo de Alarmas cuando sobrepasan los límites calculados. La inicialización se hará a partir de la base en PC y no podrá modificarse durante el funcionamiento del sistema, quedando desde un principio determinadas las variables susceptibles de tener límite dinámico.

e) Atributos relacionados

Proporciona validez al campo LIM_DINAMICO_PUESTO.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Un bit, (0 ó 1)

---

[11] Vid sección 2.1.2, *Tratamientos a las Variables*

13) LIMITE DINAMICO PUESTO

a) Identificador

LIM_DINAMICO_PUESTO

b) Nombre

Limite_puesto

c) Descripción

Indica que la variable tiene activo el límite dinámico. Si LIM_DINAMICO es uno y este campo también es uno, entonces se hará el análisis a las variables indicado en el punto anterior.

d) Uso

EL operador podrá alterar su valor, activando asi el concepto de "Límite Dinámico".

e) Atributos relacionados

Se relaciona por validación con LIM_DINAMICO y el porcentaje para la creación lo da PORC_LIM_DINAMICO.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Un bit, (0 ó 1)

14) ALARMA NO RECONOCIDA

a) Identificador

ALARMA_NO_RECONOCIDA

b) Nombre

Alarma_nreconocida

c) Descripción

Cuando su valor es uno, para las variables alarmardas, significa que el operador no ha dado su aviso de enterado en el Diálogo de Alarmas.

d) Uso

Este campo se pone a uno cada vez que una variable ingresa a la FP Diálogo de Alarmas y se reestablece por comando del operador, por tanto tendrá picos de acceso a escritura durante disturbios.

e) Atributos relacionados

Se relaciona directamente con VAR_ALARMADA.

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC

g) Tipo y Rango

Un bit, (0 ó 1)

15) ACCION DE CONTROL INHIBIDA

a) Identificador

ACCION_INH

b) Nombre

Control_inhibido

c) Descripción

ACCION_INH funciona como protección contra cambios en variables muy importantes especialmente las relacionadas con acciones de control sobre la planta, estando en uno, para hacer un cambio se solicitará la palabra clave del jefe de operación en las FP de Diálogo de Control.

d) Uso

Se inicializará con los valores de la base en PC y solo el jefe de operación está autorizado a modificarlo, su frecuencia de uso es muy baja.

e) Atributos relacionados

Es conceptualmente, candado de acceso a escritura de VALOR_LOGICO.

f) Entidades donde se le encuentra

LOGICAS

g) Tipo y Rango

Un bit, (0 ó 1)

16) ACCION DE CONTROL EN PROGRESO

a) Identificador

ACCION_EN_PROGRESO

b) Nombre

control_progreso

c) Descripción

En uno indica que la acción de control no ha finalizado, que se está llevando a cabo.

d) Uso

El atributo es especialmente útil en variables donde se aplica el concepto de retroalimentación para informar al operador de la respuesta real que tiene el sistema ante la ejecución de un comando de control. Su frecuencia de uso es baja pues está directamente ligado con la velocidad con que el operador emite sus comandos.

e) Atributos relacionados

Se relaciona con los campos DURACION_ACCION y RETRASO_RETAL.

f) Entidades donde se le encuentra

LOGICAS

g) Tipo y Rango

Un bit, (0 ó 1)

17) NO HAY VARIABLE DE RETROALIMENTACION

a) Identificador

NO_RETAL

b) Nombre

No_retro

c) Descripción

En uno significa que la variable no tiene asociada otra de retroalimentación y por consiguiente no hay modo de saber el resultado de la solicitud de acción de control a través del sistema.

d) Uso

La inicialización vendrá dada por los archivos de la base en PC y debe mantenerse durante el funcionamiento del sistema a menos que se agregue una señal (con su sensor correspondiente en campo) y entonces el cambio deberá notificarse por medio de la FP Diálogo de Servicios, de este modo, el valor de NO_RETAL es casi constante pero será accesado con tanta frecuencia como acciones de control se inicien.

e) Atributos relacionados

Se relaciona con el campo AP_RETAL.

f) Entidades donde se le encuentra

LOGICAS

g) Tipo y Rango

Un bit, (0 ó 1)

18) ESTADO NORMAL

a) Identificador

ESTADO_NORMAL

b) Nombre

Estado_normal

c) Descripción

Se aplica a variables de tipo lógico únicamente, y contiene el valor (1 ó 0) que la variable debe tener para poderla considerar en *estado normal*. Este atributo es necesario por que en los sensores se maneja tanto lógica positiva como negativa y, por ejemplo, mientras una variable en uno relaciona el estado de un equipo con "ABIERTO" otra en uno relacionará a su equipo correspondiente con "CERRADO".

d) Uso

La forma de actualización es a través de la FP Diálogo de Servicios y debe hacerse cuando la lógica de un sensor cambie.

e) Atributos relacionados

Se relaciona con VALOR_LOGICO, AP_ESTADO_NORMAL, AP_ESTADO_ANORMAL.

f) Entidades donde se le encuentra

LOGICAS, LOGICAS_CALC, DIAGNOSTICOS

g) Tipo y Rango

Un bit, (0 ó 1)

19) ALARMA DE 0 A 1
20) ALARMA DE 1 A 0

a) Identificadores

ALARMA_0A1,
ALARMA_1A0

b) Nombres

Alarma_01,
Alarma_10

c) Descripción

Aplicable únicamente a variables de tipo lógico, los campos ALARMA_0A1 y ALARMA_1A0 indican (cuando estan en uno) que la variable se alarmará cuando su estado cambie de 0 a 1 o de 1 a 0 respectivamente.

d) Uso

El módulo de Alarmas los lee para determinar si la transición de una variable lógica de un valor a otro debe enviarse a la FP Diálogo de Alarmas. La inicialización de los campo se hace de acuerdo con la base en PC pero puede modificarse con el Diálogo de Servicios.

e) Atributos relacionados

Conceptualmente se relaciona con ALARMA_INH y VALOR_LOGICO.

f) Entidades donde se le encuentra

LOGICAS, LOGICAS_CALC, DIAGNOSTICOS

g) Tipo y Rango

Un bit, (0 ó 1)

21) IDENTIFICADOR

a) Identificador

IDENTIFICADOR

b) Nombre

Identificador

c) Descripción

Se usa para identificar de manera única las variables del sistema; por ejemplo, "1V1236", "2S2024", etc. Consta de seis caracteres, los primeros tres indican el tipo y a que parte de la planta pertenece la variable, los últimos tres son para diferenciar variables que caigan en la misma categoría. En cuanto a los primeros tres caracteres tenemos:

1er caracter: Equipo Principal al que pertenece la Variable

```
1..... TG1 - HRSG1 - Generador de TG1 - BTG de TG1
2..... TG2 - HRSG2 - Generador de TG2 - BTG de TG2
5..... TV - Condensador - Generador de TV - BTG de TV
7..... CC - Miscelaneos - BTG de CC
9..... Diagnóstico de Equipo de Cómputo

Donde: TG= Turbina de Gas
       HRSG= Recuperador de Calor
       BTG= Tablero de Control
       TV= Turbina de Vapor
       CC= Control Coordinado
```

2ndo caracter: Tipo de Variable

```
V..... Analógica de Control, de Adquisición e Internas
A..... Calculadas Analógicas
F..... Flujos Acumulados (Analógicas)
H..... Horas de Operación (Analógicas)
N..... Número de Arranques (Analógicas)
E..... Eficiencias (Analógicas)
Q..... Analógicas de Configuración Inicial para Tarjetas SAC
K..... Constantes en Canastas (Analógicas)
L..... Lógicas de Control, de Adquisición e
         Internas entre canastas y VAX
S..... Lógicas del Secuenciador
D..... Calculadas Lógicas
I..... Lógicas Internas en VAX
Z..... Lógicas de Diagnóstico de Equipo de Adquisición
         (línea SAC)
```

```
Y..... Lógicas de Diagnóstico de Equipo de Cómputo, Canastas y
       red de información
```

3er caracter: Equipo del Proceso al que pertenece la Variable

```
1..... Turbocompresor de TG1 - Generador de TG1
2..... Turbocompresor de TG2 - Generador de TG2
3..... TV - Condensador - Generador de TV
4..... HRSG1
5..... HRSG2
6..... Control Coordinado y Equipo Común
9..... Otros
```

d) Uso

Este campo es estático y no se permiten desviaciones del valor tomado originalmente de la base de datos en PC, si es realmente necesario cambiar su valor debido a que una variable cambia de sistema, deberá a través de la FP de Diálogo de Servicios darse de baja y luego de alta una nueva variable; por otro lado, se trata del campo leído con mayor frecuencia por las diferentes FP's. IDENTIFICADOR es de gran importancia ya que en el "lenguaje" de planta las variables se reconocen por medio de el.

e) Atributos relacionados

A un nivel lógico existe relación de clave aspirante[12] respecto a los demás campos; solo existe en las entidades de variables otra clave aspirante que se forma a partir de CANASTA, TARJETA y BIT.

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC, CONSTANTES, DIAGNOSTICOS, CONFIGURACIONES

g) Tipo y Rango

Siete caracteres, el último tendrá el terminador nulo, los tres primeros ya fueron descritos, en cuanto al cuarto, quinto y sexto pueden variar de '0' a '9'.

---

[12]Vid sección 1.3.1 *Conceptos Generales - clave aspirante*

22) DESCRIPCION

a) Identificador

DESCRIPCION

b) Nombre

Descripcion

c) Descripción

Contiene el nombre de la variable. Existe una norma para *armar* este nombre: en primer lugar se escribe la variable de proceso (presión, temperatura, nivel, etc.), luego el lugar específico del equipo (flecha, escape, etc.) y por último el equipo principal (TG1, recuperador, etc.). Para las variables lógicas a esta descripción se le agrega en los desplegados el campo de estado normal o anormal según sea el estado de la variable obteniendo asi una descripción completa.

d) Uso

DESCRIPCION se inicializa con los datos de la base en PC pero puede modificarse a través de la FP Diálogo de Servicios lo cual no es muy frecuente, sin embargo, su lectura si lo es y la realizan varias funciones de presentación.

e) Atributos relacionados

En el caso de las variables lógicas, se complementa con el campo de ESTADO_NORMAL o ESTADO_ANORMAL según el caso.

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC, CONSTANTES, DIAGNOSTICOS, CONFIGURACIONES

g) Tipo y Rango

Treinta y un caracteres (char[31]), el último tendrá el terminador nulo.

23) PRIORIDAD

a) Identificador

PRIORIDAD

b) Nombre

Prioridad

c) Descripción

Determina que importancia tiene una variable; a menor valor, mayor la importancia de la variable.

d) Uso

En las FP's de Diálogo de Alarmas y Lista de Variables se usa para generar desplegados alternativos ordenados por importancia de las variables. Diálogo de Servicios podrá modificar el valor que en principio es el existente en la base en PC.

e) Atributos relacionados

Ninguno

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC, CONSTANTES, DIAGNOSTICOS, CONFIGURACIONES

g) Tipo y Rango

Un byte (char), (1 a 9)

24) PORCENTAJE PARA CREACION DE BANDA MUERTA

a) Identificador

PORC_BANDA_MUERTA

b) Nombre

Porc_banda_muerta

c) Descripción

Contiene el porcentaje a usar para el cálculo de *"Banda Muerta"*[13] como se guarda en un byte, no están permitidos decimales, por ejemplo un 2% de banda muerta se indica con el valor 0x02.

d) Uso

El módulo de Alarmas ocupa este campo en su análisis a la variable para determinar si está alarmada. A través de Diálogo de Servicios el operador puede cambiar el valor del mismo.

e) Atributos relacionados

Se relaciona con BANDA_MUERTA, BANDA_MUERTA_PUESTA.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Un byte (char), (0 a 99)

---

[13] Vid sección 2.1.2, *Tratamientos a las Variables*

25) PORCENTAJE PARA CREACION DE LIMITE DINAMICO

a) Identificador

PORC_LIM_DINAMICO

b) Nombre

Porc_limite_dinamico

c) Descripción

Contiene el valor del porcentaje a usar para calcular el *"Límite Dinámico"*[14] al igual que en PORC_BANDA_MUERTA el valor no permite decimales.

d) Uso

En el análisis de una variable hecho por el módulo de Alarmas cuando LIM_DINAMICO_PUESTO es válido se calcula este tipo de límite y la señal se considerará alarmada cuando rebase los valores obtenidos. Los cambios al campo se hacen a través del Diálogo de Servicios.

e) Atributos relacionados

Se relaciona con LIM_DINAMICO, LIM_DINAMICO_PUESTO.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Un byte (char), (0 ó 99)

---

[14]Vid sección 2.1.2, *Tratamientos a las Variables*

26) LIMITE CRITICO ALTO
27) LIMITE CRITICO BAJO
28) LIMITE PRECRITICO ALTO
29) LIMITE PRECRITICO BAJO

a) Identificadores

LIM_CRITICO_ALTO,
LIM_CRITICO_BAJO,
LIM_PRECRITICO_ALTO,
LIM_PRECRITICO_BAJO

b) Nombres

Limite_crit_alto,
Limite_crit_bajo,
Limite_prec_alto,
Limite_prec_bajo

c) Descripción

Estos campos contienen los puntos que limitan los diferentes estados en los que una variable analógica puede caer, el formato en que se guardan es de 2 bytes de los cuales los 12 menos significativos se toman en cuenta, esto es para facilitar su comparación con VALOR_ANALOGICO que se guarda bajo el mismo formato.

d) Uso

La inicialización se toma de los valores de la base en PC pero pueden alterarse a través de Diálogo de Servicios, los campos son leídos por varias FP's como Diagramas de Barras, Diagramas de Flujo, etc. y por el módulo de Alarmas en su análisis a variables.

e) Atributos relacionados

Se relacionan directamente con VALOR_ANALOGICO.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0x0FFF)

30) APUNTADOR AL TIPO DE LINEARIZACION

a) Identificador

AP_LINEARIZACION

b) Nombre

Ap_linearizacion

c) Descripción

El método más simple (y el más usado) de linearización del valor digitalizado de una señal proveniente de un sensor es por medio de una recta, sin embargo, algunos sensores tienen un comportamiento no-lineal y para ellos la obtención en flotante de VALOR_ANALOGICO requiere de un algoritmo especial, este campo contiene un apuntador tipo "reference"[15] a un registro de tipo LINEARIZACIONES, al campo FUNCION_LINEARIZACION que es a su vez un apuntador a la función con la que podemos obtener el valor flotante. Para detalles, consúltese mas adelante la documentación sobre este último en la sección sobre Atributos Auxiliares.

d) Uso

Para aquellas pocas variables cuyo sensor es no-lineal, las FP's que desplieguen su valor en flotante deben llamar a la función apuntada. Como la conversión más frecuente es por recta, el campo será rara vez accesado y no podrá alterarse de su valor inicial mediante Diálogo de Servicios pues los algoritmos son muy específicos para cada caso.

e) Atributos relacionados

El campo relaciona físicamente el registro-variable donde se encuentra con un registro tipo LINEARIZACIONES. Conceptualmente está ligado a LIM_OP_ALTO, LIM_OP_BAJO y VALOR_ANALOGICO.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0xFFFF)

---

[15] Vid sección 2.1.2, *Requerimientos Generales*

31) VALOR ANALOGICO DE LA VARIABLE

a) Identificador

VALOR_ANALOGICO

b) Nombre

Valor_variable

c) Descripción

En 2 bytes tomando los 12 bits menos significativos representa el valor de la variable que para el caso de las variables adquiridas se obtiene por las tarjetas SAC.

d) Uso

La escritura en el caso de las variables adquiridas lo hace el módulo de Adquisición de Señales, para las variables calculadas el módulo de Cálculos Especiales lo hace. Prácticamente todas las FP's que despliegan valores análogicos lo requieren para lectura. Es un campo muy importante y su acceso debe ser lo más rapido posible.

e) Atributos relacionados

Está relacionado con los límites crítico y precríticos y los de operación.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC, CONSTANTES, CONFIGURACIONES

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0x0FFF)

32) APUNTADOR A LA TABLA DE UNIDADES

a) Identificador

AP_UNIDAD

b) Nombre

Unidades

c) Descripción

Contiene un apuntador tipo "reference" a un registro de tipo UNIDADES, al campo NOMBRE_UNIDAD que a su vez contiene el nombre de la unidad de ingeniería en que estará expresado el flotante correspondiente a VALOR_ANALOGICO de la variable.

d) Uso

Las FP's que despliegan valores de variables análogicas siempre agregan al valor la unidad, p. ej. "3600 RPM", por tanto es un campo muy importante y en caso de requerir actualización, se hará a través de Diálogo de Servicios. Su frecuencia de escritura es baja pero la de lectura es tan alta como desplegados de variables se tengan.

e) Atributos relacionados

Conceptualmente se relaciona con el valor flotante de VALOR_ANALOGICO. El campo relaciona físicamente el registro-variable donde se encuentra con un registro tipo UNIDADES.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC, CONSTANTES

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0xFFFF)

33) ZUMBA ALARMA
34) IMPRIME ALARMA

a) Identificadores

ZUMBA_ALARMA,
IMPRIME ALARMA

b) Nombres

Zumba_alarma,
Imprime_alarma

c) Descripción

Cada bit de estos campos empezando por el menos significativo representa una estación de trabajo o una impresora del sistema, cuando el bit correspondiente a un equipo está en uno quiere decir que un mensaje auditivo o impreso (según el caso) será enviado a el cuando la variable se alarme.

d) Uso

Consultando estos campos, el módulo Alarmas puede determinar en que estación de trabajo debe hacerse zumbar la bocina y en que impresora se hará el reporte de la variable cuando pasa a un estado alarmado. A través del Diálogo de Servicios el operador puede inhibir o habilitar la impresión o el aviso auditivo.

e) Atributos relacionados

Existe relación con VAR_ALARMADA.

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC, CONSTANTES

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0xFFFF)

35) APUNTADOR A ESTADOS (ESTADO NORMAL)
36) APUNTADOR A ESTADOS (ESTADO ANORMAL)

a) Identificadores

AP_ESTADO_NORMAL,
AP_ESTADO_ANORMAL

b) Nombres

Ap_estado_normal,
Ap_estado_anormal

c) Descripción

Ambos campos, contienen cada uno, un apuntador del tipo "reference" a un registro de tipo ESTADOS, al campo NOMBRE_ESTADO que a su vez contiene el nombre de un estado como puede ser "ALTO ", "BAJO ", etc.

d) Uso

Dependiendo de VALOR_LOGICO y de ESTADO_NORMAL se desplegará uno u otro: Cuando VALOR_LOGICO == ESTADO_NORMAL, se agregará a la descripción de la variable en los desplegados de las FP's el nombre apuntado por AP_ESTADO_NORMAL, si no se cumple la igualdad mencionada, entonces se tomará AP_ESTADO_ANORMAL. La frecuencia de uso de los campos es tan alta como la del campo DESCRIPCION (en variables lógicas).

e) Atributos relacionados

Hay relación conceptual con DESCRIPCION, VALOR_LOGICO y ESTADO_NORMAL, además, El campo relaciona físicamente el registro-variable donde se encuentra con un registro tipo ESTADOS.

f) Entidades donde se le encuentra

LOGICAS, LOGICAS_CALC, DIAGNOSTICOS

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0xFFFF)

37) APUNTADOR A LA TABLA DE NOMBRES DE ACCIONES

a) Identificador

AP_ACCION

b) Nombre

Accion_control

c) Descripción

Contiene un apuntador tipo "reference" a un registro de tipo ACCIONES, al campo NOMBRE_ACCION que a su vez contiene el nombre de una acción de control.

d) Uso

La FP Diálogo de Control mostrará este valor en sus desplegados para indicar al operador que tipo de acción se realizará al seleccionar un elemento de la pantalla. El valor al que apunta podrá cambiarse mediante Diálogo de Servicios. Es un campo de baja frecuencia de acceso.

e) Atributos relácionados

Se relaciona con NOMBRE_ACCION y con DESCRIPCION ya que es necesario conocer sobre que elemento se efectuará la acción.

f) Entidades donde se le encuentra

LOGICAS

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0xFFFF)

38) TIMER DE LA ACCION DE CONTROL

a) Identificador

DURACION_ACCION

b) Nombre

Timer_control

c) Descripción

Es el tiempo que se debe mantener un comando de control, por ejemplo, el comando enviado a un actuador en campo para cerrar una válvula puede requerir sostenerse durante 5 segundos.

d) Uso

La FP Diálogo de Control mantiene por omisión las acciones del operador sobre objetos que simulan botones durante 150 ms, sin embargo cuando el campo es diferente de cero la acción (cambio de cero a uno o de uno a cero según sea el caso en el campo VALOR_LOGICO) deberá sostenerse durante el tiempo que se indique. Cabe señalar que el formato en que se guarda el tiempo corresponde al manejado por las funciones de VAXELN.

e) Atributos relacionados

Se relaciona con VALOR_LOGICO.

f) Entidades donde se le encuentra

LOGICAS

g) Tipo y Rango

Cuatro bytes (int), (0 a 0xFFFFFFFF)

39) TIEMPO MAXIMO DE ESPERA A RETROALIMENTACION

a) Identificador

RETRASO_RETAL

b) Nombre

Timer_retro_control

c) Descripción

Contiene el tiempo máximo en que una acción de control debe completarse, es decir, el retraso máximo en que la variable de retroalimentación debe reportar como concluida la acción.

d) Uso

La FP Diálogo de Control al comprobar que VALOR_RETAL_ESPERADO no coincide con el valor de la variable apuntada por AP_RETAL notificará de manera especial al operador que la acción de control que intentó llevar acabo no pudo realizarse. La frecuencia de acceso al campo es baja.

e) Atributos relacionados

VALOR_RETAL_ESPERADO y VALOR_LOGICO de la variable AP_RETAL.

f) Entidades donde se le encuentra

LOGICAS

g) Tipo y Rango

Cuatro bytes (int), (0 a 0xFFFFFFFF)

40) APUNTADOR A LA VARIABLE DE RETROALIMENTACION

a) Identificador

AP_RETAL

b) Nombre

Ap_retro

c) Descripción

Contiene un apuntador tipo "reference"[16] a un registro de tipo LOGICAS, al campo VALOR_LOGICO que como veremos más adelante, contiene el valor lógico de la variable (en este caso la variable de retroalimentación).

d) Uso

La FP Diálogo de Control revisará este campo (si NO_RETAL == 0) cuando se envíe una comando de control en espera de su efecto, comparándolo con VALOR_RETAL_ESPERADO.

e) Atributos relacionados

VALOR_RETAL_ESPERADO

f) Entidades donde se le encuentra

LOGICAS

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0xFFFF)

[16]Vid sección 2.1.2, *Requerimientos Generales*

41) VALOR ESPERADO DE LA VARIABLE DE RETROALIMENTACION

a) Identificador

VALOR_RETAL_ESPERADO

b) Nombre

Estado_control

c) Descripción

Contiene el valor que se espera adquiera la variable de retroalimentación una vez que la acción de control haya sido llevada a cabo.

d) Uso

La FP Diálogo de Control revisará este campo (si NO_RETAL == 0) cuando se envíe un comando de control, comparándolo con el valor que realmente tiene la variable en AP_RETAL.

e) Atributos relacionados

AP_RETAL

f) Entidades donde se le encuentra

LOGICAS

g) Tipo y Rango

Un bit, (0 ó 1)

42) VALOR LOGICO

a) Identificador

VALOR_LOGICO

b) Nombre

Valor_logico

c) Descripción

Contiene el valor lógico de la variable.

d) Uso

VALOR_LOGICO es actualizado por el módulo de Adquisición de Señales, mismo que recibe constantemente el estado de las tarjetas de entradas lógicas donde haya cambiado al menos una señal además, la FP Diálogo de Control escribe en el. Tanto la frecuencia de lectura como la de escritura son muy altas especialmente durante disturbios.

e) Atributos relacionados

ALARMA_0A1, ALARMA_1A0

f) Entidades donde se le encuentra

LOGICAS, LOGICAS_CALC, DIAGNOSTICOS

g) Tipo y Rango

Un bit, (0 ó 1)

43) APUNTADOR A LA TABLA DE DE INTEGRACIONES

a) Identificador

AP_INTEGRACION

b) Nombre

Ap_est_integracion

c) Descripción

Contiene un apuntador tipo "reference" a un registro de tipo INTEGRACIONES que es una estructura donde se "integran" o totalizan valores adquiridos de variables analógicas.

d) Uso

En la inicialización del sistema se definen los apuntadores de las variables susceptibles de integrarse (tales como flujos, consumos, etc.) y aunque la FP Diálogo de Servicios puede alterar los datos de la estructura correspondiente a una variable, no se podrá modificar el apuntador en sí ya que esto ocasionaría cruzamientos en los algoritmos del módulo de Cálculos Especiales.

e) Atributos relacionados

Se relaciona físicamente con un registro tipo INTEGRACIONES.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0xFFFF)

44) LIMITE DE OPERACION ALTO
45) LIMITE DE OPERACION BAJO

a) Identificadores

LIM_OP_ALTO,
LIM_OP_BAJO

b)Nombres

Limite_oper_alto,
Limite_oper_bajo

c) Descripción

Con estos campos podemos obtener el valor en flotante de una variable a partir de VALOR_ANALOGICO, el método de linearización más simple consiste de una sola recta; para más detalles sobre la conversión consúltese la sección *"Tratamientos a Variables"*.

d) Uso

En general las FP's que muestran el valor de variables analógicas requieren de estos valores para hacer la conversión; la obligación de transformar VALOR_ANALOGICO en flotante se remite a cada desplegado ya que es más económico desde el punto de vista tiempo, convertir únicamente las variables desplegadas en un momento dado que convertir todas las recibidas por el módulo de Adquisición.

e) Atributos relacionados

Los límites se relacionan conceptualmente con AP_LINEARIZACION y con VALOR_ANALOGICO.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Cuatro bytes (float)

46) RANGO DE GRAFICACION EN PAPEL INFERIOR
47) RANGO DE GRAFICACION EN PAPEL SUPERIOR

a) Identificadores

RANGO_GRAF_INFERIOR,
RANGO_GRAF_SUPERIOR

b) Nombres

Rango_gra_papel_inf,
Rango_gra_papel_sup

c) Descripción

Contienen el rango de trabajo con que se ajustan los registradores, pueden moverse a voluntad para analizar mejor la variable aunque siempre estarán limitados por el rango de la variable.

d) Uso

El módulo de Graficación en Papel que controla los graficadores lee estos campos para enviar los comandos adecuados. Se inicializa con los datos de la base en PC pero puede alterarse posteriormente, su frecuencia de uso es baja.

e) Atributos relacionados

Se relacionan con VALOR_ANALOGICO y con los límites de operación.

f) Entidades donde se le encuentra

ANALOGICAS, ANALOGICAS_CALC

g) Tipo y Rango

Cuatro bytes (float)

48) BIT DE ALARMA EN DIAGRAMAS DE FLUJO

a) Identificador

ALARMAR_DIAGRAMA

b) Nombre

Alarmar_diagrama

c) Descripción

En uno indica que la variable es muy importante dentro de su equipo y por tanto, en los desplegados de Diagramas de Flujo por sistema y subsistema donde esté presente el equipo asociado debe indicarse la alarma.

d) Uso

La FP Diagramas de Flujo lee ALARMAR_DIAGRAMA y altera el color de la figura que representa a un sistema o equipo. El operador puede modificar el campo desde Diálogo de Servicios. La frecuencia de escritura es por tanto baja y la de lectura depende de cuantos elementos de la planta se desplieguen en un momento dado.

e) Atributos relacionados

VAR_ALARMADA, AP_SISTEMA, AP_SUBSISTEMA

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC

g) Tipo y Rango

Un bit, (0 ó 1)

49) GANANCIA DE LA SEÑAL

a) Identificador

GANANCIA_SENAL

b) Nombre

Ganancia

c) Descripción

Contiene el valor de la ganancia que se le da al convertidor analógico/digital de las tarjetas SAC para incrementar el voltaje recibido. Esto es especialmente útil para variables analógicas cuyos instrumentos de campo tienen rangos pequeños de señal.

d) Uso

Ya que todas las señales de campo se acondicionan a modo de que hacia canastas lleguen con un valor de 0 a 10 Volts, una petición de cambio de ganancia hecha por el operador en Diálogo de Servicios afectará de manera proporcional el campo LIM_OP_ALTO, además el módulo de Adquisición y Envío de Mensajes avisará a la canasta correspondiente sobre el cambio. Por omisión las variables tienen ganacia de uno.

e) Atributos relacionados

LIM_OP_ALTO, VALOR_ANALOGICO

f) Entidades donde se le encuentra

ANALOGICAS

g) Tipo y Rango

Un byte (char), (1,2 ó 4)

50) APUNTADOR A LA TABLA DE SISTEMAS
51) APUNTADOR A LA TABLA DE SUBSISTEMAS

a) Identificadores

AP_SISTEMA,
AP_SUBSISTEMA

b) Nombres

Numero_sistema
Numero_subsistema

c) Descripción

Contienen apuntadores tipo "reference" a registros de tipo SISTEMAS y SUBSISTEMAS, a los campos NOMBRE_SISTEMA y NOMBRE_SUBSISTEMA que a su vez, como se explicará más adelante, contienen los nombres del sistema y subsistema al que pertenece la variable.

d) Uso

Ya que en el campo IDENTIFICADOR de las variables está de manera implícita el sistema, en los desplegados no fue solicitada la presentación del valor de estos atributos, sin embargo son útiles cuando se requieren desplegados de señales ordenadas por sistema y subsistema. El operador podrá cambiar el valor del campo que inicialmente tiene el valor de la base en PC.

e) Atributos relacionados

IDENTIFICADOR, AP_SISTEMA, AP_SUBSISTEMA

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC, CONSTANTES, DIAGNOSTICOS, CONFIGURACIONES

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0xFFFF)

52) COLOR DE LA VARIABLE

a) Identificador

COLOR_VAR

b) Nombre

Color

c) Descripción

El campo representa el estado operativo de la variable. El color que aparece indica si la variable está fuera de servicio, normal, alarmada, etc.

d) Uso

En cuanto al acceso para escritura, el módulo de Alarmas es el único que lo puede alterar, ni siquiera Diálogo de Servicios tiene acceso a el pues se trata de un atributo de uso interno. La lectura a COLOR_VAR es muy frecuente al activar diferentes desplegados y durante disturbios la escritura se incrementa.

e) Atributos relacionados

Se relaciona con todos los límites y campos ligados con alarmas.

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC

g) Tipo y Rango

Un byte (char), (0 a 0x1F)

53) TENDENCIA DE LA VARIABLE

a) Identificador

TENDENCIA_VAR

b) Nombre

Tendencia

c) Descripción

Observando el comportamiento de VALOR_ANALOGICO de las variables analógicas adquiridas lo podemos clasificar en una de tres tendencias: que la variable tiende a aumentar (1), permanece en su valor (2) o disminuye (3). Este campo indica de manera numérica la tendencia.

d) Uso

El módulo de Adquisición es el único que por medio de una comparación entre el valor anterior y el valor actual puede modificar el campo; TENEDENCIA_VAR se usa en FP's como Tendencias y Diagramas de Barras. Es un campo frecuentemente accesado considerando que todas las analógicas adquiridas tienen una de tres posibles tendencias.

e) Atributos relacionados

VALOR_ANALOGICO

f) Entidades donde se le encuentra

ANALOGICAS

g) Tipo y Rango

Un byte (char), (1,2 ó 3)

54) CANASTA
55) TARJETA
56) BIT

a) Identificadores

CANASTA,
TARJETA,
BIT

b) Nombres

Canasta,
Tarjeta,
Bit

c) Descripción

A estos tres atributos en conjunto también se les conoce como *"Dirección Hardware"* y es otra forma de identificar las señales, de hecho, en los programas que forman la IHM, es la clave de acceso primaria a los registros de variables.

d) Uso

Aunque como se mencionó forman la clave primaria, estos campos en sí casi no son accesadps pues como veremos en secciones posteriores, el acceso a los registros-variable se hará a través de un arreglo de apuntadores.

e) Atributos relacionados

Siendo la clave primaria, se relaciona a nivel lógico con todos los campos de las entidades de variables.

f) Entidades donde se le encuentra

LOGICAS, ANALOGICAS, LOGICAS_CALC, ANALOGICAS_CALC, CONSTANTES, DIAGNOSTICOS, CONFIGURACIONES

g) Tipo y Rango

```
Cada uno es de un byte (char).
    CANASTA......   de 0 a 23
    TARJETA......   de 0 a 63
    BIT..........   de 0 a 15
```

Antes de continuar con la sección sobre Atributos Auxiliares, se presenta una tabla con los atributos de variables y las entidades donde son válidos:

ATRIBUTO	V	L	C	I	K	Y	Q
PUNTO_BLOQUEADO	*	*	*	*	*	*	*
SCAN_INH	*	*	*	*	*	*	*
ALARMA_INH	*	*	*	*			
VAR_ALARMADA	*	*	*	*			
EVENTO_INH	*	*	*	*			
ESTADO_FUNCIONAL	*		*				
VALOR_DADO	*	*	*	*	*	*	*
CALIDAD_DATO	*						
BANDA_MUERTA	*		*				
BANDA_MUERTA_PUESTA	*		*				
ALARMA_RETORNO	*		*				
LIM_DINAMICO	*		*				
LIM_DINAMICO_PUESTO	*		*				
ALARMA_NO_RECONOCIDA	*	*	*	*			
ACCION_INH		*					
ACCION_EN_PROGRESO		*					
NO_RETAL		*					
ESTADO_NORMAL		*		*		*	
ALARMA_0A1		*		*		*	
ALARMA_1A0		*		*		*	
IDENTIFICADOR	*	*	*	*	*	*	*
DESCRIPCION	*	*	*	*	*	*	*
PRIORIDAD	*	*	*	*	*	*	*
PORC_BANDA_MUERTA	*		*				
PORC_LIM_DINAMICO	*		*				
LIM_CRITICO_ALTO	*		*				
LIM_CRITICO_BAJO	*		*				
LIM_PRECRITICO_ALTO	*		*				
LIM_PRECRITICO_BAJO	*		*				
AP_LINEARIZACION	*		*				
VALOR_ANALOGICO	*		*		*		*
AP_UNIDAD	*		*		*		
ZUMBA_ALARMA	*	*	*	*			
IMPRIME_ALARMA	*	*	*	*			
AP_ESTADO_NORMAL		*		*		*	
AP_ESTADO_ANORMAL		*		*		*	
AP_ACCION		*					
DURACION_ACCION		*					
RETRASO_RETAL		*					
AP_RETAL		*					
VALOR_RETAL_ESPERADO		*					
VALOR_LOGICO		*		*		*	
AP_INTEGRACION	*		*				

ATRIBUTO	V	L	C	I	K	Y	Q
LIM_OP_ALTO	*		*				
LIM_OP_BAJO	*		*				
RANGO_GRAF_INFERIOR	*		*				
RANGO_GRAF_SUPERIOR	*		*				
ALARMAR_DIAGRAMA	*	*	*	*			
GANANCIA_SENAL	*						
AP_SISTEMA	*	*	*	*	*	*	*
AP_SUBSISTEMA	*	*	*	*	*	*	*
COLOR_VAR	*	*	*	*			
TENDENCIA_VAR	*						
CANASTA	*	*	*	*	*	*	*
TARJETA	*	*	*	*	*	*	*
BIT	*	*	*	*	*	*	*

Atributos auxiliares

1) NOMBRE DE LA UNIDAD

a) Identificador

NOMBRE_UNIDAD

b) Nombre

Nombre_unidades

c) Descripción

El campo contiene todos los posibles nombres de las unidades de ingeniería que en los desplegados se anexan al valor de variables análogicas.

d) Uso

El valor de los campo es constante, ya que si se cambia uno de ellos, todos los registros de variables que apuntan a ese campo verán el nuevo valor; tampoco se espera que la FP Diálogo de Servicios pueda aumentar la cantidad de unidades. Su inicialización es a partir de la base en PC.

e) Atributos relacionados

El campo es apuntado y se relaciona con registros de variables analógicas a través AP_UNIDAD.

f) Entidades donde se le encuentra

UNIDADES

g) Tipo y Rango

Siete caracteres (char[7]), el último contiene el terminador nulo.

2) NOMBRE DEL ESTADO DE LA VARIABLE

a) Identificador

NOMBRE_ESTADO

b) Nombre

Nombre_estado

c) Descripción

Contiene cadenas con el nombre de los estados, se usa para complementar la descripción de señales lógicas.

d) Uso

NOMBRE_ESTADO es apuntado por las referencias a estado normal y estado anormal en registros de variables lógicas, por tanto se requiere cada vez que un desplegado muestre la descripción de una variable lógica con su estado.

e) Atributos relacionados

Existe relación física con los campos AP_ESTADO_NORMAL y AP_ESTADO_ANORMAL.

f) Entidades donde se le encuentra

ESTADOS

g) Tipo y Rango

Siete caracteres (char[7]), el último contiene el terminador nulo.

3) NOMBRE DE LA ACCION DE CONTROL DE LA VARIABLE

a) Identificador

NOMBRE_ACCION

b) Nombre

Nombre_accion

c) Descripción

Contiene las cadenas de los diferentes nombres de acciones.

d) Uso

Diálogo de Control muestra en sus desplegados los nombres de las acciones que ejecutará el operador al seleccionar algún elemento gráfico. Se inicializa con los valores de la base en PC y no se modificará durante la ejecución del sistema.

e) Atributos relacionados

Físicamente hay ligas del campo AP_ACCION de los registros de variables de control hacia NOMBRE_ACCION.

f) Entidades donde se le encuentra

ACCIONES

g) Tipo y Rango

Ocho caracteres (char[8]), el último contiene el terminador nulo.

4) NOMBRE DEL SISTEMA AL QUE PERTENECE LA VARIABLE

a) Identificador

NOMBRE_SISTEMA

b) Nombre

Nombre_sistema

c) Descripción

Contiene las cadenas de los nombres de los sistemas en que se divide la planta.

d) Uso

Sirve para construir menús e identificar listados de variables ordenadas por sistema y subsistema, es de poco uso ya que el identificador de cada variable trae de por sí asociado a que lugar pertenece en los tres primeros caracteres.

e) Atributos relacionados

Físicamente hay ligas de AP_SISTEMA de los registros de variables hacia estos campos.

f) Entidades donde se le encuentra

SISTEMAS

g) Tipo y Rango

Treinta y dos caracteres (char[32]), el último contiene el terminador nulo.

5) NOMBRE DEL SUBSISTEMA AL QUE PERTENECE LA VARIABLE

a) Identificador

NOMBRE_SUBSISTEMA

b) Nombre

Nombre_sistema

c) Descripción

Contiene las cadenas de los nombres de los subsistemas en que se divide la planta.

d) Uso

El campo se presenta en menús y listados de variables ordenadas por sistema y subsistema, es de poco uso ya que el identificador de cada variable trae de por sí asociado a que lugar pertenece en los tres primeros caracteres.

e) Atributos relacionados

Físicamente hay ligas del campo AP_SUBSISTEMA de los registros-variable.

f) Entidades donde se le encuentra

SUBSISTEMAS

g) Tipo y Rango

Treinta y dos caracteres (char[32]), el último contiene el terminador nulo.

6) INDICADOR DE SISTEMAS VALIDOS

a) Identificador

SISTEMAS_VALIDOS

b) Nombre

Sistemas_validos

c) Descripción

Cada bit de este campo corresponde a un sistema, de modo que para un registro tipo SUBSISTEMAS específico, cuando el bit de un sistema está en uno, indica que dicho sistema es válido en ese subsistema.

d) Uso

Se ocupa en los desplegados donde el operador debe seleccionar un sistema y un subsistema (p. ej. en la FP Lista de Variables); primeramente se muestran todos los sistemas, el operador selecciona uno de ellos y luego aplicando una operación OR sobre SISTEMAS_VALIDOS se listan únicamente los susbsistemas que dicho sistema tiene. El campo una vez inicializado no cambia.

e) Atributos relacionados

Se asocia con registros del tipo SISTEMAS.

f) Entidades donde se le encuentra

SUBSISTEMAS

g) Tipo y Rango

Cuatro bytes (unsigned int), (0 a 0xFFFFFFFF)

7) NOMBRE DEL METODO DE LINEARIZACION

a) Identificador

NOMBRE_LINEARIZACION

b) Nombre

Nombre_linearizacion

c) Descripción

Contiene las cadenas de los nombres de los diferentes métodos de linearización que se aplican a las variables. De hecho se trata de la descripción breve de la función apuntada por FUNCION_LINEARIZACION.

d) Uso

Se mostrará solo cuando el método empleado no sea recta. Se trata de un campo muy poco utilizado cuyo contenido no cambia desde su inicialización.

e) Atributos relacionados

Se asocia conceptualmente con FUNCION_LINEARIZACION.

f) Entidades donde se le encuentra

LINEARIZACIONES

g) Tipo y Rango

Ocho caracteres (char[8]), el último contiene el terminador nulo.

8) APUNTADOR A LA FUNCION QUE EFECTUA LA LINEARIZACION

a) Identificador

FUNCION_LINEARIZACION

b) Nombre

Funcion_linearizacion

c) Descripción

Contiene un apuntador a funcion que deberá proporcionar el usuario y que con objeto de acordar sintaxis, deberá aceptar el valor adquirido de la variable, sus límites de operación alto y bajo y un apuntador a flotante donde dejar el valor calculado:

```
funcion (float *resultado, unsigned short VALOR_VARIABLE,
         float LIM_OP_ALTO, float LIM_OP_BAJO);
```

d) Uso

El programador deberá verificar si el valor de la variable que se va a desplegar requiere de un tipo de linearización especial y en su caso llamar a la función apuntada con la sintaxis indicada.

e) Atributos relacionados

Se relaciona con NOMBRE_LINEARIZACION, y con los parámetros de función; VALOR_VARIABLE, LIM_OP_ALTO, LIM_OP_BAJO.

f) Entidades donde se le encuentra

LINEARIZACIONES

g) Tipo y Rango

Cuatro bytes, apuntador a caracter (char *)

9) NOMBRE DEL PROGRAMA EN DISCO

a) Identificador

NOMBRE_PROGRAMA

b) Nombre

Nombre_programa

c) Descripción

Contiene los nombres de la imagen ejecutable que se conserva en disco de algunos programas que se deben arrancar bajo ciertas circunstancias relacionadas con acciones de control.

d) Uso

Solo los programas de control accessan este campo para ejecutar los programas como procesos hijos.

e) Atributos relacionados

Se relaciona con NUMERO_PROGRAMA.

f) Entidades donde se le encuentra

PROGRAMAS

g) Tipo y Rango

Treinta y dos caracteres (char[32]), El último con terminador nulo.

10) NUMERO DEL PROGRAMA EN DISCO

a) Identificador

NUMERO_PROGRAMA

b) Nombre

Numero_programa

c) Descripción

Da un identificador único a cada programa en disco de modo que actualizando el campo NOMBRE_PROGRAMA se puede cambiar el nombre de la imágen residente en disco sin tener que cambiar nada en el programa en si.

d) Uso

Los programas que lanzan acciones de control accesan este tipo de registros. El campo NUMERO_PROGRAMA no cambiará, de hecho es un atributo a cuyo valor los programadores deben apegarse.

e) Atributos relacionados

Se relaciona con NOMBRE_PROGRAMA de manera biunívoca.

f) Entidades donde se le encuentra

PROGRAMAS

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0xFFFF)

11) FACTOR DE INTEGRACION

a) Identificador

FACTOR_INT

b) Nombre

Factor_int

c) Descripción

Especifica el factor que se debe aplicar al valor de la variable apuntada por AP_VAR_ANALOGICA (VALOR_ANALOGICO convertido en flotante) antes de sumarlo a la CUENTA_CORRIENTE_INT, esto es especialmente útil cuando el tiempo de muestreo (periodicidad con que se monitorea la variable) y el sensor del valor de la variable están dados en diferentes unidades de tiempo. La fórmula para obtener el factor es:

FACTOR_INT= TIEMPO_MUESTREO_INT / Ps

Donde 'Ps' es el valor del periodo en que está dado el instrumento de medición.

Por ejemplo, supongamos que se va a integrar una variable cuyo sensor nos da 'litros por minuto' pero queremos que nuestro tiempo de muestreo sea cada segundo, entonces, FACTOR_INT= 1s / 60s = 1/60

d) Uso

Es de uso exclusivo en el módulo de Cálculos Especiales y se debe alterar únicamente cuando cambie TIEMPO_MUESTREO_INT o el sensor en campo.

e) Atributos relacionados

Directamente con TIEMPO_MUESTREO_INT.

f) Entidades donde se le encuentra

INTEGRACIONES

g) Tipo y Rango

Cuatro bytes (float), (0 a 0xFFFFFFFF)

12) TIEMPO DE MUESTREO DE LA INTEGRACION
13) TIEMPO DE INTEGRACION TRANSCURRIDO
14) TIEMPO MAXIMO DE INTEGRACION

a) Identificadores

TIEMPO_MUESTREO_INT,
TIEMPO_ACTUAL_INT,
TIEMPO_MAXIMO_INT

b) Nombres

Tiempo_muestreo_int,
Tiempo_actual_int,
Tiempo_maximo_int

c) Descripción

Con estos tres campos se controlan las integraciones desde el punto de vista tiempo, TIEMPO_MUESTREO_INT indica cada cuando debe tomarse el valor de la variable apuntada por AP_VAR_ANALOGICAparaactualizarCUENTA_CORRIENTE_INT;TIEMPO_ACTUAL_INT contiene el tiempo que ha transcurrido desde el inicio de la integración actual y TIEMPO_MAXIMO_INT da el valor máximo que puede alcanzar TIEMPO_ACTUAL_INT de manera que se de por terminada la integración.

d) Uso

Son de uso exclusivo por el módulo de Cálculos Especiales y a menores tiempos de muestreo se tendrán que accesar con mayor frecuencia. A excepción de TIEMPO_ACTUAL_INT que es de uso interno, los otros campos se llegan a modificar para variar las características de la integración.

e) Atributos relacionados

Se relacionan entre ellos y TIEMPO_MUESTREO_INT con FACTOR_INT ya que sirve para calcularlo.

f) Entidades donde se le encuentra

INTEGRACIONES

g) Tipo y Rango

Cuatro bytes (int), (0 a 0xFFFFFFFF)

15) CUENTA CORRIENTE DE INTEGRACION
16) CUENTA MAXIMA DE INTEGRACION
17) ULTIMA CUENTA DE INTEGRACION

a) Identificadores

CUENTA_CORRIENTE_INT,
CUENTA_MAXIMA_INT,
CUENTA_ULTIMA_INT

b) Nombres

Cuenta_corriente_int,
Cuenta_maxima_int,
Cuenta_ultima_int

c) Descripción

Con estos tres campos se controlan las integraciones desde el punto de vista valor, CUENTA_CORRIENTE_INT es donde se acumulan las integraciones en curso y no puede superar el valor de CUENTA_MAXIMA_INT. En cuanto a CUENTA_ULTIMA_INT, se actualiza cada vez que TIEMPO_ACTUAL_INT alcanza el valor de TIEMPO_MAXIMO_INT o bien si CUENTA_CORRIENTE_INT alcanza el valor de CUENTA_MAXIMA_INT.

d) Uso

El campo de CUENTA_ULTIMA_INT solo se accesa cuando las integraciones se reinician; los otros dos, serán accesados con tanta frecuencia como bajo sea el tiempo máximo de integración. Solo CUENTA_MAXIMA_INT es modificable por el operador.

e) Atributos relacionados

Se relacionan entre ellos y con el VALOR_ANALOGICO, LIM_OP_ALTO, LIM_OP_BAJO de la variable analógica referenciada por AP_VAR_ANALOGICA.

f) Entidades donde se le encuentra

INTEGRACIONES

g) Tipo y Rango

Cuatro bytes (float), (0 a 0xFFFFFFFF)

18) APUNTADOR A LA VARIABLE ANALOGICA

a) Identificador

AP_VAR_ANALOGICA

b) Nombre

Ap_var_analogica

c) Descripción

Contiene un apuntador tipo "reference"[17] a un registro de tipo ANALOGICAS que debe ser la variable de adquisición sobre la que se desea hacer la integración.

d) Uso

El módulo de cálculos especiales ocupa este apuntador para obtener en un momento dado el VALOR_ANALOGICO y los límites de operación de una variable (necesarios para efectuar la conversión a valor flotante). Y asi actualizar CUENTA_CORRIENTE_INT.

e) Atributos relacionados

CUENTA_CORRIENTE_INT

f) Entidades donde se le encuentra

INTEGRACIONES

g) Tipo y Rango

Dos bytes (unsigned short), (0 a 0xFFFF)

---

[17]Vid sección 2.1.2, *Requerimientos Generales*

19) CANASTA PRIMARIA O DE RESPALDO
20) CANASTA EN SERVICIO O FUERA DE SERVICIO
21) CANAL OPERANDO 'A' O 'B'
22) CANASTA REAL O FICTICIA

a) Identificadores

CANASTA_PRIMARIA,
CANASTA_EN_SERVICIO,
CANASTA_CANAL,
CANASTA_REAL

b) Nombres

Canasta_primaria,
Canasta_en_servicio,
Canasta_canal,
Canasta_real

c) Descripción

Estos atributos describen el estado operativo de una canasta.

d) Uso

Del conocimiento de la dirección hardware y de las propiedades de una canasta puede el programador en un momento dado determinar si una variable es de campo o interna (recuérdese que las variables internas tienen direcciones ficticias) y en Tolerancia a Fallas cambiar de canasta primaria a respaldo cuando la primaria queda fuera de servicio.

e) Atributos relacionados

Se relacionan entre ellos y con la dirección hardware de las variables (CANASTA, TARJETA, BIT).

f) Entidades donde se le encuentra

CANASTAS

g) Tipo y Rango

Un bit, (0 ó 1)

2.3. ESTADISTICAS DE LA INFORMACION A MANEJAR

En la sección 2.2.1. (*Entidades*), reagrupamos los tipos de variables de acuerdo a características comunes para formar entidades de variables y determinamos las entidades auxiliares necesarias, posteriormente, cada una será convertida en un tipo de registro. Hacer un estudio sobre la cantidad de registros y la frecuencia de altas y bajas es indispensable en la definición de cualquier base de datos y especialmente necesario en nuestro caso donde el cimiento de la arquitectura, que propondremos en el siguiente capítulo, está en un "archivo-esquema" en el que estos datos deben contemplarse.

Fuera del punto de vista estructural, la información sobre las variables de la Central de Ciclo Combinado de Gómez Palacio tiene su origen en una base en PC usada para proyectos anteriores relacionados con centrales de generación eléctrica similares, en la estructura de dicha base se contemplan campos de interés tanto para la IHM como para alambrado y canastas, es depurada constantemente y para transferirla a VAX se genera un archivo de texto con los campos adecuados. De esta base tomamos los datos presentados en la siguiente tabla:

ENTIDAD	TIPO	PARCIAL	SUMA	HOLGURA	TOTAL
ANALOGICAS	V	1066	1066	134	1200
LOGICAS	L	986	1314	136	1450
	S	328			
ANALOGICAS_CALC	A	60	217	23	240
	H	23			
	F	62			
	N	23			
	E	49			
LOGICAS_CALC	D	8	57	8	65
	I	49			
CONSTANTES	K	634	634	66	700
DIAGNOSTICOS	Z	518	544	56	600
	Y	26			
CONFIGURACIONES	Q	192	192	23	215
T O T A L: - - - - - - - - - -			4024	446	4470

En la columna 'ENTIDAD' están los nombres de las siete entidades de variables mencionadas en 2.2.1 *Entidades*, en 'TIPO' el código del tipo de variable que corresponde al

segundo caracter del identificador; en 'PARCIAL' y 'SUMA' tenemos la cantidad por tipo de variable y por entidad respectivamente. Una vez instalado el sistema es difícil que sea necesario agregar o eliminar registros pero permitiremos cierta libertad dejando un diez porciento de holgura.

En cuanto a las entidades auxiliares, los valores para SISTEMAS y SUBSISTEMAS están completamente determinados por la división ya existente en planta; las ocurrencias de UNIDADES, ESTADOS, y ACCIONES tienen valores prácticamente constantes y en la base de datos original (la base en PC) no forman un archivo de base de datos independiente sino que se encuentran embebidas como campos dentro de la base de variables a pesar de la redundancia que ello implica; filtrando los valores repetidos encontramos la cantidad real.

Para LINEARIZACIONES tomamos en cuenta que en el proyecto de la CCC de Dos-Bocas, Veracruz, solamente en tres casos fué necesario recurrir a un método alternativo al de linearización por una sola recta. La cantidad de PROGRAMAS que se planea lanzar por comandos de control es inicialmente de 15, y el número de ocurrencias para la entidad INTEGRACIONES será cuando mucho igual a la de variables analógicas ya que son las únicas que pueden tener asociada una tabla de integración. CANASTAS depende del hardware que se intalará, es decir, 22 canastas. Como en el caso anterior para las entidades auxiliares también dejamos aproximadamente un diez porciento de holgura.

ENTIDAD	CANTIDAD	HOLGURA	TOTAL
UNIDADES	44	6	50
ESTADOS	32	18	50
ACCIONES	4	12	15
SISTEMAS	15	5	20
SUBSISTEMAS	78	12	90
LINEARIZACIONES	4	11	15
PROGRAMAS	15	5	20
INTEGRACIONES	217	33	250
CANASTAS	22	8	30

REFERENCIAS

Slavko V. Tripkovic
A Fast Access Method for Real-Time Database
International Conference on Industrial Electronics
Control and Instrumentation
Tokyo, Japón
1984

Charles J. Conrad
Issues in Manufacturing Floor Data and Control
Control Engineering
Marzo, 1991

Robert Bucher
Database Integration
Boosts Plantwide Quality Control
Control Engineering
Marzo, 1991

Dennis L. Brandl, John McGehee
Relational Database Extensions for
Real-Time Process Control
Proceedings of the Control West Conference
Johnson City, TN, USA
1985

Philip Off P.
Human Interface Techniques for Real-Time
Global Database Access
Advances in Instrumentation Conference
Anaheim, CA, USA
1987

B. J. Schroer, Ernst Goss P.
Design Considerations for a State Level Energy
Database Management System
Computers & Industrial Engineering
1988, Vol. 14

CAPITULO 3

DISEÑO DE LA BASE DE DATOS

3.1. ARQUITECTURA

La arquitectura que a continuación se propone fue planificada para que respondiera a los requerimientos expuestos anteriormente y aunque el esquema resultante depende por completo de la aplicación en cuestión, fueron elaboradas algunas herramientas que permiten generar automáticamente el código de las estructuras tanto de alto como de bajo nivel, de esta manera, es posible implantar el mismo concepto no solo para sistemas de presentación de centrales termoeléctricas similares sino en general, para cualquier aplicación en tiempo real que requiera un grado considerable de información almacenada en RAM y estructurada como base de datos.

En cuanto a los objetivos globales tenemos que en los procedimientos directos se minimizan los parámetros críticos que presenta cualquier aplicación en tiempo real: tiempo de respuesta y actualización de la información[1]. Tomando en cuenta que la falla en alguno de estos puntos no significa simplemente deterioro en el desempeño sino pérdida del control sobre equipos de la planta, se decidió proporcionar al programador con métodos de acceso directos. La Figura 3.1 muestra de un modo gráfico los conceptos aplicados en la arquitectura de la base:

[1]Tripkovic, Slavko V., *A Fast Access Method for Real-Time Database*

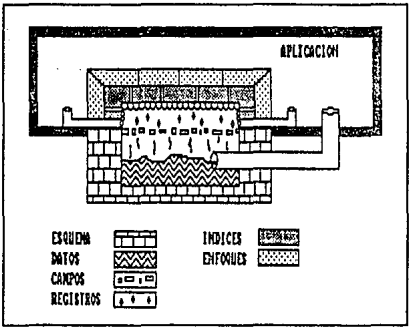


Figura 3.1
Arquitectura general

A partir de un *"archivo-esquema"* escrito en un *Lenguaje de Descripción de Esquema* semejante en cuanto a propósito al DDL de los sistemas de red[2] y cuyas reglas de sintaxis serán expuestas mas adelante, se obtiene código fuente al aplicar una herramienta CASE[3] (Computer Aided Software Engineering) desarrollada con ese propósito; el código producido contiene la declaración y definición de las estructuras de datos que sirven de cimiento a la base, dado que se trata de código fuente, una vez armada la aplicación, es imposible modificar dichas estructuras a menos que se vuelva a aplicar la herramienta, se recompile y ligue la aplicación, es decir que el programador bien puede realizar una serie de operaciones sobre los datos en lo que podríamos llamar un lenguaje de comandos, pero no asi crear o alterar estructuras definidas desde un principio. Esto se representa en el diagrama mediante un líquido (datos) que toma la forma del recipiente que lo contiene (esquema).

Como puede observarse, todo el código de la aplicación se construye sobre el sustento del esquema, existiendo una conexión directa entre este y los datos. Sobre los datos y a modo de criba tenemos descriptores de campos que en un nivel intermedio llevan a la formación de tipos de registros representados como burbujas; el programador puede también trabajar desde este nivel con la información y en el dispone de procedimientos para manejarla fácilmente al tenerla agrupada por campos y registros.

Los registros se acumulan y forman estructuras ordenadas sobre los índices los cuales al entrar en contacto (asignarse) con enfoques permiten a un bloque-enfoque *"ver"* todos los campos

---

[2]Vid sección 3.5.2, *Historia del Modelo Reticular*

[3]Vid sección 3.1.2, *Esquema*

de un registro encadenado a un índice; a este nivel el programador cuenta con una serie de procedimientos encaminados principalmente para el manejo de los registros en índices.

En la sección 3.1.1 Se expondrá el procedimiento de modelado conceptual y su normalización llevados a cabo para esta aplicación en particular atendiendo a los requerimientos y características de la información determinados en el capítulo anterior. Posteriormente, se explicarán en detalle los elementos constructivos de la arquitectura haciendo notoria su flexibilidad.

3.1.1. Modelado

Modelo conceptual gráfico

1) Entidad "VARIABLES"

Siguiendo las convenciones y simbología establecida en la sección 1.4.2 (*Modelado Conceptual*) y considerando que la cantidad de atributos es grande, tomamos la tabla entidad-atributo dada al final de la descripción de atributos de variables[4] y agrupamos los atributos en 9 tipificaciones que se convertirán en conjuntos del tipo de las hipergráficas dentro de los cuales está presente el producto cartesiano de sus elementos:

A) Dirección hardware

```
DIR_HW=     CANASTA, TARJETA, BIT
```

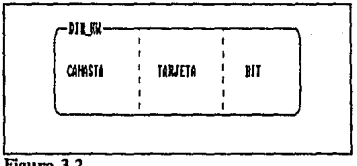


Figura 3.2
Entidad DIR_HW

B) Atributos globales

```
GLOBALES=  PUNTO_BLOQUEADO, SCAN_INH, VALOR_DADO,
           IDENTIFICADOR, DESCRIPCION, PRIORIDAD,
           AP_SISTEMA, AP_SUBSISTEMA
```

[4]Vid sección 2.2.2, *Atributos de Variables*

C) Atributos generales de alarmas

```
ALAR_GLOB= ALARMA_INH, VAR_ALARMADA, EVENTO_INH,
           ALARMA_NO_RECONOCIDA, ZUMBA_ALARMA,
           IMPRIME_ALARMA, ALARMAR_DIAGRAMA, COLOR_VAR
```

D) Atributos de límites analógicos

```
LIM_ANA=   ESTADO_FUNCIONAL, BANDA_MUERTA,
           BANDA_MUERTA_PUESTA, ALARMA_RETORNO,
           LIM_DINAMICO, LIM_DINAMICO_PUESTO,
           PORC_BANDA_MUERTA, PORC_LIM_DINAMICO,
           LIM_CRITICO_ALTO, LIM_CRITICO_BAJO,
           LIM_PRECRITICO_ALTO, LIM_PRECRITICO_BAJO,
           AP_LINEARIZACION, AP_INTEGRACION,
           LIM_OP_ALTO, LIM_OP_BAJO,
           RANGO_GRAF_INFERIOR, RANGO_GRAF_SUPERIOR
```

E) Atributos específicos de variables lógicas

```
ESP_LOG=   ESTADO_NORMAL, ALARMA_0A1, ALARMA_1A0,
           AP_ESTADO_NORMAL, AP_ESTADO_ANORMAL,
           VALOR_LOGICO
```

F) Atributos de control

```
ACC_CONT=  ACCION_INH, ACCION_EN_PROGRESO, NO_RETAL,
           AP_ACCION, DURACION_ACCION, RETRASO_RETAL,
           AP_RETAL, VALOR_RETAL_ESPERADO
```

G) Atributos de analógicas adquiridas

```
ANA_ADQ=   CALIDAD_DATO, GANANCIA, TENDENCIA
```

H) Atributo de valor analógico

```
VALOR_ANA= VALOR_ANALOGICO
```

I) Atributo de Apuntador a Unidades de Ingeniería

```
UNID_ING=  AP_UNIDAD
```

De acuerdo a lo anterior, la hipergráfica para las entidades de variables es:

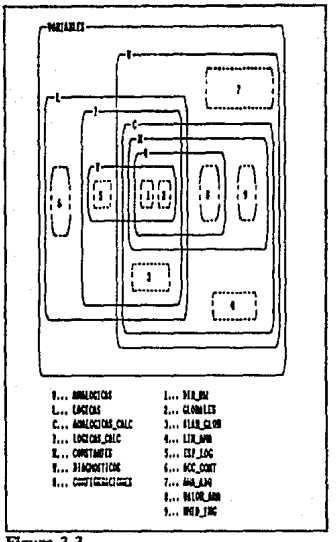


Figura 3.3
Entidad VARIABLES

Observemos dos aspectos en la Figura 3.3 que hacen visibles las ventajas del diagramado por hipergráficas: Dentro de VARIABLES encontramos dos clases de conjuntos de acuerdo con el tipo de línea que delimita su perímetro; a los de línea contínua se aplica la operación OR, es decir:

```
VARIABLES= ANALOGICAS  ||  LOGICAS  ||
           ANALOGICAS_CALC  ||  LOGICAS_CALC  ||
           CONSTANTES  ||  DIAGNOSTICOS  ||
           CONFIGURACIONES
```

Esto es por la diferencia entre las hipergráficas y los diagramas de Venn[5], por ejemplo, según la teoría de conjuntos tradicional, ANALOGICAS_CALC sería un subconjunto de ANALOGICAS; en nuestro caso solo indicamos que existen parámetros comunes a ambas, llendo más lejos, en el diagrama está implicito que un registro de la entidad VARIABLES será de uno y solamente un tipo (ANALOGICAS, LOGICAS, etc.).

Por otro lado, los conjuntos delimitados por líneas entrecortadas representan producto cartesiano, por ejemplo, las entidades del tipo CONSTANTES son resultado de aplicar producto cartesiano entre los grupos de atributos DIR_HW, GLOBALES, VALOR_ANA, UNID_ING:

```
CONSTANTES= DIR_HW, GLOBALES, VALOR_ANA, UNID_ING
```

Un último aspecto que nos hace falta considerar sobre esta entidad es que sus ocurrencias (las variables) se deben identificar de manera única por la *dirección hardware*, es decir que:

```
DF[DIR_HW]= GLOBALES, ALAR_GLOB, LIM_ANA,
            ESP_LOG, ACC_CONT, ANA_ADQ,
            VALOR_ANA, UNID_ING
```

En la Figura 3.4 queda representada la dependencia funcional y al mismo tiempo subiendo un nivel de abstracción para conseguir cierta simplificación gráfica la entidad VARIABLES se muestra únicamente con los nombres de grupos de atributos.

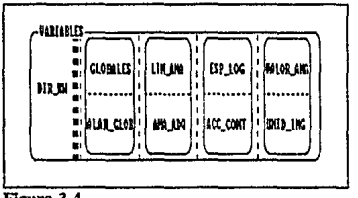


Figura 3.4
Gráfica simplificada de la entidad VARIABLES

[5]Vid sección 1.4.2, *Simbología*

2) Entidades Auxiliares

A) UNIDADES= NOMBRE_UNIDAD

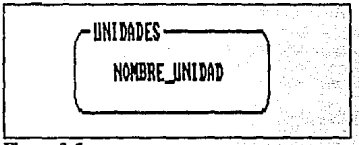


Figura 3.5
Entidad UNIDADES

B) ESTADOS = NOMBRE_ESTADO

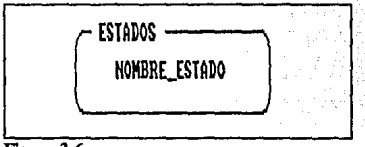


Figura 3.6
Entidad ESTADOS

C) ACCIONES = NOMBRE_ACCION

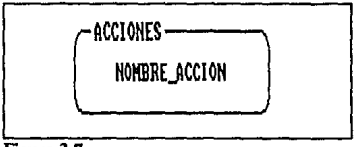


Figura 3.7
Entidad ACCIONES

D) SISTEMAS = NOMBRE_SISTEMA

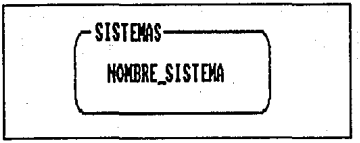


Figura 3.8
Entidad SISTEMAS

E) SUBSISTEMAS = NOMBRE_SUBSISTEMA, SISTEMAS_VALIDOS

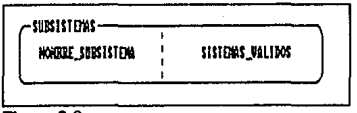


Figura 3.9
Entidad SUBSISTEMAS

F) LINEARIZACIONES = NOMBRE_LINEARIZACION,
FUNCION_LINEARIZACION

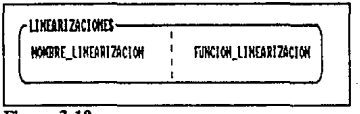


Figura 3.10
Entidad LINEARIZACIONES

G) PROGRAMAS = NOMBRE_PROGRAMA, NUMERO_PROGRAMA

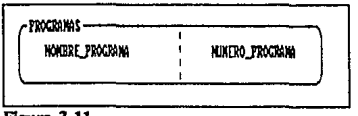


Figura 3.11
Entidad PROGRAMAS

```
H) INTEGRACIONES = FACTOR_INT, TIEMPO_MUESTREO_INT,
                   TIEMPO_ACTUAL_INT, TIEMPO_MAXIMO_INT
                   CUENTA_CORRIENTE_INT, CUENTA_MAXIMA_INT,
                   CUENTA_ULTIMA_INT, AP_VAR_ANALOGICA
```

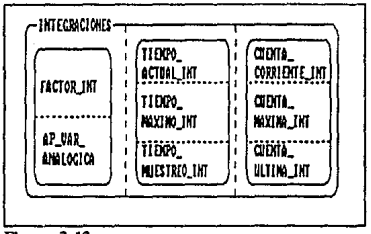


Figura 3.12
Entidad INTEGRACIONES

```
I) CANASTAS = CANASTA_PRIMARIA, CANASTA_EN_SERVICIO,
              CANASTA_CANAL, CANASTA_REAL
```

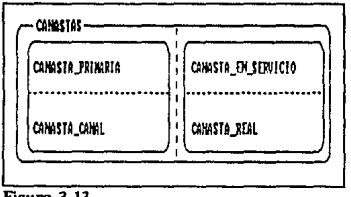


Figura 3.13
Entidad CANASTAS

3) Relaciones entre entidades

Ya en el último paso de este diagrama conceptual sin normalizar, representamos las relaciones entre entidades de variables y auxiliares; el modelo se construye a partir de las siguientes características:

A) Todas las variables pertenecen a un SISTEMA y un SUBSISTEMA específico, a su vez, un SISTEMA tiene varios SUBSISTEMAS y estos se pueden encontrar en varios SISTEMAS.

```
VARIABLES <<---------> SISTEMAS
VARIABLES <<---------> SUBSISTEMAS
SISTEMAS  <<-------->> SUBSISTEMAS
```

B) Para el cálculo de valores flotantes usando VALOR_ANALOGICO y los límites de operación, se aplica un método de LINEARIZACION.

```
VARIABLES <<-------->> LINEARIZACIONES
```

C) Dentro de las características de variables lógicas, se tienen dos ESTADOS posibles (para 0 y para 1).

```
VARIABLES <<-------->> ESTADOS
```

D) Las características y condiciones de funcionamiento de la canasta a la que está asignada una variable se definen en la entidad CANASTAS.

```
VARIABLES <<---------> CANASTAS
```

E) Las variables analógicas como flujos, y consumos a las que se aplican integraciones establecen relación uno a uno con INTEGRACIONES, entidad que a su vez se relaciona con la variable analógica de la que se obtiene el valor de campo.

```
VARIABLES      <----------> INTEGRACIONES
INTEGRACIONES <----------> VARIABLES
```

F) Las variables de control tienen la descripción de su acción correspondiente en ACCIONES; además, comunmente existe relación con una variable de retroalimentación de la cual se espera el resultado de la acción.

```
VARIABLES <----------> ACCIONES
VARIABLES <----------> VARIABLES
```

G) Las variables analógicas tienen su valor flotante expresado en ciertas UNIDADES de ingeniería.

```
VARIABLES <<---------> UNIDADES
```

Antes de presentar el modelo conceptual profundizaremos un nivel de abstracción mas, especificando en las siete condiciones anteriores los atributos a través de los que se establecen las relaciones; recordemos que en 2.1.2 (3) *Requerimientos generales* se indicó la necesidad de establecer un tipo de campo "apuntador" de una entidad 'x' a un **registro o campo** de otra entidad

'y'. De esta forma, usaremos la notación ENTIDAD.ATRIBUTO para las entidades auxiliares y ENTIDAD(GRUPO).ATRIBUTO para las entidades de variables, donde 'GRUPO' es una de las nueve tipificaciones hechas al principio de esta sección; en los casos donde no se apunta a un campo específico sino a un registro, reemplazaremos 'ATRIBUTO' por un asterisco.

A) Variables, Sistemas y Subsistemas

```
VARIABLES(GLOBALES).AP_SISTEMA     <<---------> SISTEMAS.NOMBRE_SISTEMA
VARIABLES(GLOBALES).AP_SUBSISTEMA  <<---------> SUBSISTEMAS.NOMBRE_SUBSISTEMA
SISTEMAS.*                         <<--------->> SUBSISTEMAS.SISTEMAS_VALIDOS
```

B) Variables y Linearizaciones

```
VARS(LIM_ANA).AP_LINEARIZACION <<--------->> LINEARIZACIONES.FUNCION_LINEARIZACION
```

C) Variables y Estados

```
VARIABLES(ESP_LOG).* <<--------->> ESTADOS.NOMBRE_ESTADO
```

D) Variables y Canastas

```
VARIABLES(DIR_HW).CANASTA <<---------> CANASTAS.*
```

E) Variables e Integraciones

```
VARIABLES(LIM_ANA).AP_INTEGRACION <----------> INTEGRACIONES.*
INTEGRACIONES.AP_VAR_ANALOGICA    <----------> VARIABLES().*
```

F) Variables y Acciones

```
VARIABLES(ACC_CONT).AP_ACCION <----------> ACCIONES.NOMBRE_ACCION
VARIABLES(ACC_CONT).AP_RETAL  <----------> VARIABLES(ESP_LOG).VALOR_LOGICO
```

G) Variables y Unidades

```
VARIABLES(UNID_ING).AP_UNIDAD <<---------> UNIDADES.NOMBRE_UNIDAD
```

Modelo Conceptual sin Normalizar

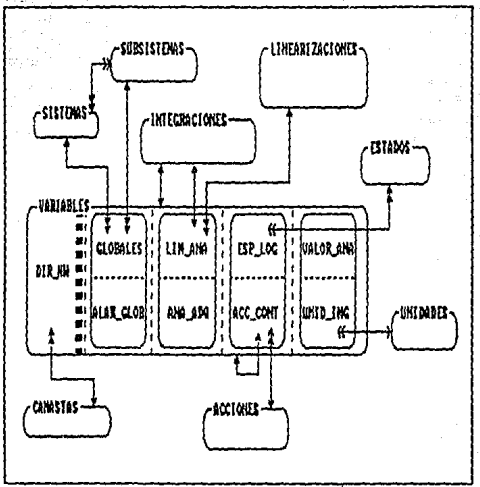


Figura 3.14
Modelo conceptual para la base de datos (sin normalizar)

Normalización

El modelo obtenido en la sección anterior adolece de algunas deficiencias, mismas que corregimos siguiendo el proceso de normalización[6].

Primera forma normal

Debido a los elementos gráficos y reglas de construcción nuestro modelo ya está en la 1FN (*FN*: Forma Normal).

---

[6]Vid sección 1.4.2, *Normalización*

Segunda forma normal

Dada la dependencia funcional total de los campos respecto a *"Dirección Hardware"* anotada en **3.1.1**, el modelo cumple con las condiciones de 2FN.

```
DF[DIR_HW]= GLOBALES, ALAR_GLOB, LIM_ANA,
            ESP_LOG, ACC_CONT, ANA_ADQ,
            VALOR_ANA, UNID_ING
```

Tercera forma normal y BCNF (Boyce-Codd Normal Form)

Para llegar a este nivel debemos eliminar las dependencias transitivas, con objeto de evitar redundancia; recordemos[7] que en la tercera formal no deben existir dependencias transitivas entre atributos no-rectores y que en la forma BCNF no debe existir ningún tipo de dependencia transitiva lo cual, se consigue separando los atributos afectados en una entidad auxiliar. En el diagrama observamos que existe una dependencia de un atributo del grupo (ACC_CONT) hacia VARIABLES sin embargo, se trata de una relacion uno a uno y por tanto no habrá problemas de redundancia, además, resulta imprácticó crear otra entidad VARIABLES a la que se apunte ya que significaría duplicación de la base.

Cuarta forma normal

En este punto se eliminan las dependencias funcionales multivaluadas y las relaciones varios a varios entre entidades. Por ese lado la relación entre ESTADOS y VARIABLES(ESP_LOG) es varios a varios ya que para cada variable lógica hay dos ESTADOS posibles, el problema se resuelve separando los apuntadores, este mismo método lo aplicamos a la relación entre SISTEMAS y SUBSISTEMAS donde cada bit del campo SISTEMAS_VALIDOS incluye (1) o excluye (0) al subsistema en cuestión dentro de un sistema específico[8].

```
VARIABLES(ESP_LOG).AP_ESTADO_NORMAL  <-------->> ESTADOS.NOMBRE_ESTADO
VARIABLES(ESP_LOG).AP_ESTADO_ANORMAL <-------->> ESTADOS.NOMBRE_ESTADO

SISTEMAS.* <<-------->> SUBSISTEMAS.SISTEMAS_VALIDOS:0
SISTEMAS.* <<-------->> SUBSISTEMAS.SISTEMAS_VALIDOS:1
 - - -          - -        - -        - -        - -
SISTEMAS.* <<-------->> SUBSISTEMAS.SISTEMAS_VALIDOS:15
```

Con estas previsiones, proponemos el modelo conceptual normalizado y como ya se definieron los atributos válidos para cada entidad, por simplicidad gráfica solo presentamos

---

[7]Vid *Tercera forma normal* en secc 1.4.2

[8]Vid sección 2.2.2, *Indicador de Sistemas Válidos*

aquellos nombres de atributos que relacionan entidades, los demás atributos de cada entidad son reemplazados con un asterisco.

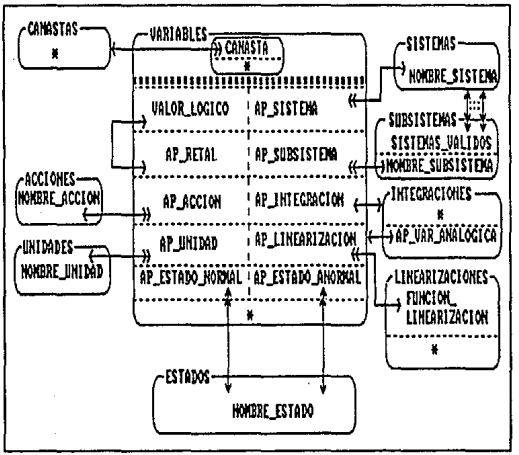


Figura 3.15
Modelo conceptual normalizado

Adecuación

Antes de traducir el modelo conceptual a un archivo-esquema, se aclararán algunos puntos sobre el modo en que manejaremos las relaciones entre entidades adecuándolas a los elementos específicos que el lenguaje-esquema nos proporciona; más adelante trataremos al detalle las herramientas que permiten definir la estructura de la base de datos a través del lenguaje-esquema y su sintaxis completa pero, por el momento, basta con mencionar que las estructuras de datos ideadas y sus relaciones permiten aplicar el enfoque de red al modelo conceptual. Retomando el concepto *Conjunto de Tipos*[9] que establece relación uno a varios entre registro propietario y registros miembro vamos a redibujar la gráfica presentada en *1.5.4 Análisis Comparativo* con los elementos que nuestro lenguaje-esquema permite declarar:

[9]Vid sección 1.5.2, *Enfoque de Red*

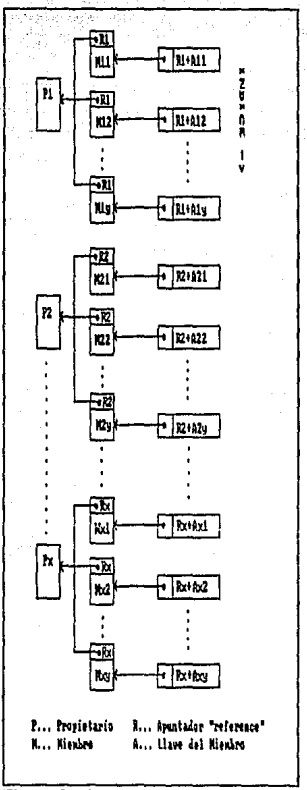


Figura 3.16
Creación de tipos de conjunto

En la Figura 3.16 centremos nuestra atención en los elementos "reference" y el índice; en una relación de la forma PROPIETARIO <-----> > MIEMBRO el enlace de un MIEMBRO a un PROPIETARIO se establece con el apuntador "reference" por ejemplo, la relación del registro

miembro 'M22' al propietario 'P2' es a través de 'R2'. El enlace en sentido inverso es a través del índice, de modo que el viaje en sentido inverso del propietario 'P2' al miembro 'M22' se hace buscando en el índice la dirección de 'P2' mas la llave que corresponde a 'M22', es decir, 'R2+A22'.

A diferencia del DDL desarrollado por CODASYL que automáticamente declara ambos apuntadores, buscando ahorro de memoria e independencia nuestro lenguaje-esquema permite declarar por separado estos elementos, por ello, debemos analizar las relaciones del tipo uno a varios para determinar si los usuarios requieren enlaces en ambos sentidos.

En general, en todas las relaciones uno a varios y uno a uno los programadores necesitan un enlace limpio del miembro al propietario lo cual se logra con apuntadores "reference" hacia el registro propietario y opcionalmente hacia un campo específico del mismo. En cuanto a las relación entre VARIABLES y CANASTA no necesitamos de un campo tipo "reference" especial con el valor del apuntador ya que de por si el campo CANASTA de VARIABLES se usará para indicar al mismo tiempo el número de canasta y el registro de la entidad CANASTAS correspondiente. Lo mismo aplicamos a SISTEMAS_VALIDOS de SUBSISTEMAS donde el estado de cada bit responde a la presencia o ausencia del subsistema específico en un sistema. En cuanto al enlace múltiple de propietario a miembros en las relaciones:

```
AP_ACCION           <------>>  NOMBRE_ACCION
AP_UNIDAD           <------>>  NOMBRE_UNIDAD
AP_ESTADO_NORMAL    <------>>  NOMBRE_ESTADO
AP_ESTADO_ANORMAL   <------>>  NOMBRE_ESTADO
AP_SISTEMA          <------>>  NOMBRE_SISTEMA
AP_SUBSISTEMA       <------>>  NOMBRE_SUBSISTEMA
AP_LINEARIZACION    <------>>  FUNCION_LINEARIZACION
```

Aunque conceptualmente existe, el programador no las ocupa en ambos sentidos, por ejemplo, no necesitará saber para ningún algoritmo o Función de Presentación las VARIABLES que apunten al mismo NOMBRE_ACCION; sin embargo la relación:

```
SUBSISTEMAS.SISTEMAS_VALIDOS   <------>>  SISTEMAS
```

Se requiere establecer completamente, en ambos sentidos, ya que diversas actividades se basan en la selección de un sistema y posteriormente de uno de sus subsistemas, para ello, crearemos un índice que ligue ordenadamente los subsistemas de acuerdo con su subsistema.

Por último, en cuanto a la entidad PROGRAMAS esta será tratada por separado, no establece relación alguna con VARIABLES ya que solo los algoritmos del módulo de Cálculos Especiales la usan y su proposito es hacerlos independientes del nombre del programa en disco.

3.1.2. Esquema

Proceso para la integración de sistemas

Ya diseñado el modelo conceptual, el primer paso hacia la creación del sistema es trasladarlo a un archivo-esquema, el cual sirve como entrada única a la herramienta (un programa que produce código en C) que interpreta el lenguaje y *"genera"* las declaraciones y definiciones de las estructuras de datos tanto de bajo como de alto nivel, además crea un programa para inicializar en blanco los archivos de registros en disco que contendrán la información de cada tipo de registro. Antes de ejecutar el programa GENERA, deben configurarse los archivos encabezado (Consultar sección 3.1.2, *Encabezados de Configuración*).

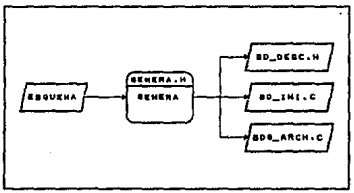


Figura 3.17
Traducción del archivo-esquema

ESQUEMA
: Archivo escrito en lenguaje-esquema

GENERA
: Programa intérprete del esquema

GENERA.H
: *Encabezado* de configuración de GENERA.C

BD_DESC.H
: *Encabezado* con declaración de estructuras como "extern"

BD_INI.C
: Declaración y definición de estructuras y función de inicialización de las mismas

BDG_ARCH.C
: Programa para inicializar archivos de registros

El siguiente paso es la creación en blanco de los archivos de registro (su tamaño y nombre se especifica en el archivo-esquema) para la base compilando, ligando y corriendo el programa BDG_ARCH en el disco y subdirectorio donde residirán tales archivos:

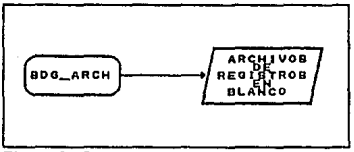


Figura 3.18
Inicialización de archivos de datos

Por último, compilamos y ligamos todos los módulos de programas de aplicación agregando el módulo con los procedimientos de alto nivel del manejador de la base de datos (BD_FUNC.C), el de inicialización de estructuras (BD_INI.C) y el de las funciones de usuario para cálculo de llaves de índices y acceso de bajo nivel:

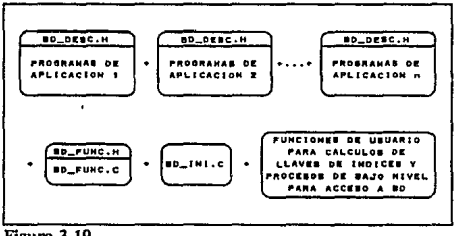


Figura 3.19
Compilación y ligado

Como ejemplo, los siguientes son comandos en VMS para ligar un sistema que corrió bajo VAXELN incluyendo el programa de aplicación "ejeln.c":

```
define decw$include dka100:[eln.dwinclude]
cc/define=(vms,vmseln,vaxelnc) ejeln+eln$:vaxelnc/lib
cc/define=(vms,vmseln,vaxelnc) bd_ini+eln$:vaxelnc/lib
cc/define=(vms,vmseln,vaxelnc) bd_func+eln$:vaxelnc/lib
link/exec=ejeln/nosyslib-
      ejvax2,bd_ini,bd_func, -
      eln$:elndecw_xlib/library,-
      eln$:crtlobject/library,-
      eln$:rtlobject/library,-
      eln$:crtlshare/library,-
```

```
        eln$:rtlshare/library,-
        eln$:rtl/library
purge/nolog/noconfirm *.*
deassign decw$include
```

Estructuras de datos

La herramienta *"GENERA"* de acuerdo a lo especificado en el archivo-esquema produce código en lenguaje C pero ciertas porciones del mismo son indepedientes de la aplicación y dan el soporte básico, de acuerdo a esto podemos clasificar las estructuras en dos tipos: "Predefinidas" o "Definibles"

1) Tipos simples (predefinidos)

Para la declaración de tipos de campos e índices se propuso un grupo de tipos válidos, este es un subconjunto del total de tipos que permite el lenguaje C y además, incluye otros construidos a partir de los básicos. Las tablas muestran el identificador **#define** del tipo, el valor numérico con que es reemplazado por el preprocesador del C, su correspondiente tipo en C y la palabra con que es reconocido en el archivo-esquema.

TIPOS DE CAMPOS

#define	VALOR	TIPO (C)	ESQUEMA
BD_SHORT	1	short	short
BD_INTEGER	2	int	int
BD_FLOAT	4	float	float
BD_DOUBLE	8	double	double
BD_CHAR	16	char	char
BD_STRING	256	char[]	string
BD_POINTER	512	char *	pointer
BD_REFERENCE	1024	unsigned short	reference
BD_NBITS	2048	unsigned :	bit
BD_UNSIGNED	4096	unsigned	u

TIPOS DE INDICES

#define	VALOR	TIPO (C)	ESQUEMA
BD_INTEGER	2	- - -	int
BD_STRING	256	- - -	string
BD_ENUM	11	- - -	enum
BD_USER	10	- - -	user

Los tipos de dato válidos para campos son: short, int, float, double, char, string, pointer, reference, bit. Además podemos combinar algunos de estos tipos con unsigned (u) formando las siguientes palabras válidas para el lenguaje-esquema: ushort, uint, ufloat, udouble, uchar. En cuanto a los tipos bit y string adicionalmente debemos indicar su longitud.

Los índices solo pueden ser de cuatro tipos: "int", "string", "enum" y "user"; ya que no existen tipos equivalentes en lenguaje C se implementaron como tipos compuestos. La estructura básica o celular usada para construir índices de cualquier tipo es el "nodo" que gráficamente consta de dos elementos de enlace: apuntador a un nodo en el siguiente nivel y apuntador a un nodo dentro del mismo nivel.

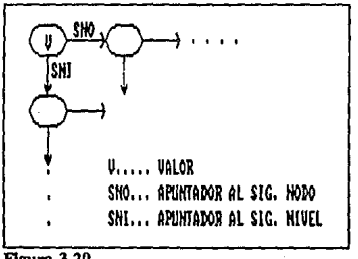


Figura 3.20
Nodo

En lenguaje C se declara como:

```
struct bd_nodo {
  unsigned char   valor;
  struct bd_nodo  *sig_nodo;
  struct bd_nodo  *sig_nivel;
}
```

2) Indices tipo "enum" (predefinidos)

Este tipo de índice es el más simple y lo representamos conceptualmente de la siguiente forma:

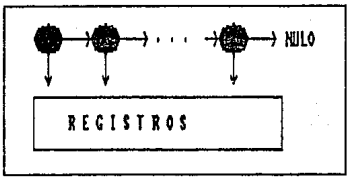


Figura 3.21
Indice tipo "enum"

Las características de los índices tipo enum son:

* Un solo nivel de nodos que apunta directamente a registros.
* Nodo final apuntando a NULO.
* Carencia de llave de acceso.
* Búsqueda de registros lineal y en un solo sentido (comportamiento semejante a una lista ligada).

3) Indices tipo "string" (predefinidos)

En estos índices existen varios niveles, uno por cada byte de la llave de acceso, su estructura es muy semejante a un índice jerárquico y el movimiento dentro de ellos combina procedimientos aleatorios y lineales, por esa razón ambos tipos de acceso son posibles.

Entre las características de los índices string están:

* Búsqueda multi-nivel en un sentido.
* El último nodo de cada nivel apunta a NULO.
* Existen tantos niveles como caracteres tenga la llave (sin contar el terminador nulo) de manera que en el nivel 'n' se tienen todos los valores que ha presentado el 'n-ésimo' byte de la llave.
* El orden de acomodo de los nodos dentro de un nivel es por el valor de su byte asociado en la llave y tiene que ser ascendente o descendente.
* El último nivel de nodos apunta directamente a registros.

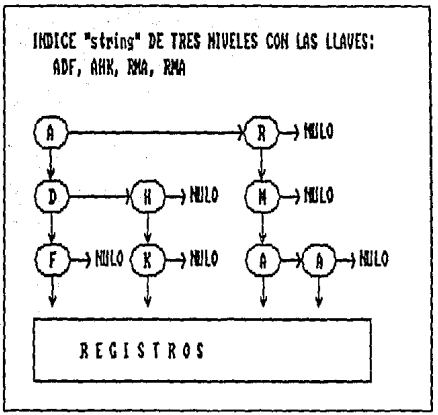


Figura 3.22
Ejemplo de índice tipo "string"

En el ejemplo de la Figura 3.22 es de notar como aunque en un índice exista duplicidad de llaves, dentro de una misma cadena de nodos no pueden existir nodos con valores repetidos salvo en el último nivel, esto queda reflejado en un sustancial ahorro de memoria y velocidad de acceso al hacer menos largas las cadenas.

4) Indices tipo "int" (predefinidos)

Los índices tipo "int" son una variante de los "string", permiten valores de llave desde 0 hasta 65535, es decir, el rango de dos bytes (unsigned short); después de un estudio sobre optimización de los índices (que expondremos en 3.3 *Evaluación*) se concluyó que para ese rango la cantidad de niveles más conveniente es de 4 con 8 nodos en cada uno ($8^4 = 4096$) para una distribución uniforme de llaves; pero como un short es representado de manera interna en dos bytes, las funciones de manejo de la base de datos al operar sobre índices "int" parten cada byte en dos grupos de 4 bits, de modo que internamente el primer nivel tiene los cuatro bits más significativos de una llave. Fuera de esta conversión de 2 a 4 bytes, este tipo de índice tiene las mismas características que el "string".

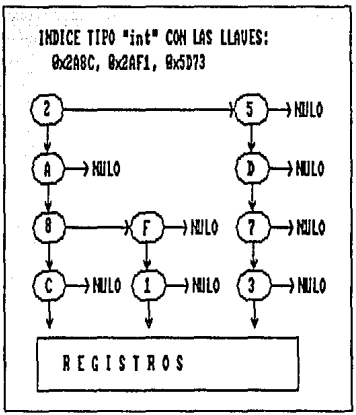


Figura 3.23
Ejemplo de índice tipo "int"

5) Indices tipo "user" (definibles)

Para que el usuario pueda diseñar sus propios métodos de acceso directos a la base y con objeto de llenar requerimientos de alto desempeño especialmente en aplicaciones de tiempo real, se deja abierta la posibilidad para que el programador aproveche este tipo declarado pero no definido llenándolo con sus estructuras y pueda incluso en un momento dado asociarlo a un enfoque.

6) Estructuras de registros (definibles)

El intérprete del archivo-esquema además de declarar estructuras descriptivas (*descriptores*) de información, declara para la organización y sostenimiento de la misma, arreglos de estructuras por cada tipo de registro en los cuales efectivamente se guardan los datos de la base, tales arreglos tienen la forma:

```
struct nombre_registro {
  char      tipo_reg;
  tipo  nombre_campo1;
  tipo  nombre_campo2;
    . .       . . .
    . .       . . .
  tipo  nombre_campoN;
}nombre_arreglo_registro;
```

Todas las estructuras-registro tienen como primer miembro a **tipo_reg**, por medio de este byte las funciones del manejador y el programador pueden reconocer de que tipo de registro se trata ya que por norma, contendrá el valor del identificador #define asociado, siendo positivo si su información es válida o negativo si el registro está vacío.

7) Identificadores (predefinidos y definibles)

Dentro de los identificadores #define tenemos ambos tipos:

Predefinidos:

* Los de tipos de datos de campos e índices (BD_SHORT, BD_INTEGER, BD_USER, etc.)
* 4 Identificadores que equivalen a la cantidad de elementos definidos por descriptor:
  BD_NO_CAMPOS, BD_NO_REGISTROS,
  BD_NO_INDICES, BD_NO_ENFOQUES

Definibles:

* Identificadores de campos, índices, registros y enfoques, p.ej:

```
/* Identificadores de los campos */
  #define PUNTO_BLOQUEADO              1
  #define SCAN_INH                     2
  #define ALARMA_INH                   3
  #define VAR_ALARMADA                 4
    .  .    .  .  .                    .
    .  .    .  .  .                    .
  #define CANASTA_EN_SERVICIO         76
  #define CANASTA_CAÑAL               77
  #define CANASTA_REAL                78
```

8) Estructura de protección (predefinida)

Si la aplicación ha de ejecutarse bajo VAXELN, es incluido un arreglo de estructuras con MUTEX para controlar escrituras a registros e índices, el manual[10] recomienda para garantizar el alineamiento de este objeto declararlo como primer elemento de una estructura:

```
struct bd_struct_prot {
  MUTEX  mutex;
}bd_obj_mutex[BD_NO_REGISTROS+BD_NO_INDICES+1];
```

[10]VAXELN C Runtime Library Reference Manual, p65-67

9) Elementos lógicos y sus descriptores (predefinidos)

Por cada tipo de elemento lógico de la arquitectura (campos, tipos de registro, índices y enfoques) se define un arreglo de descriptores que contendrá las características de los elementos de ese tipo y cuya inicialización corre a cargo de la función bd_inicia_bd() del módulo BD_INI.C que es uno de los archivos automáticamente "generados". A continuación explicamos los elementos de cada descriptor, en las porciones de código en C mostradas, los arreglos char declarados como:

```
char   arreglo [?];
```

Indican que el programa "GENERA" calcula el elemento de mayor tamaño a ser almacenado y reemplaza ese valor en ?.

a) Descriptor de campos

```
struct bd_desc_campo {
  BD_INT  tipo;
  BD_INT  longitud;
  BD_INT  reg_ref;
  BD_INT  campo_ref;
  BD_BIT  indexado:1;
}bdd_campo[BD_NO_CAMPOS+1];
```

Se declara un arreglo de descriptores con tantos elementos como número de campos y de cada uno guardamos su **tipo** (el valor correspondiente al nombre #define, p.ej. BD_STRING... 256), y su **longitud** en bytes. En caso de que **tipo** == BD_REFERENCE, **reg_ref** contendrá el identificador del tipo de registro al que apunta el campo descrito y **campo_ref** el identificador del campo apuntado.

b) Descriptor de registros

```
struct bd_desc_registro {
  struct bd_struct_prot  *prot;
  char    archivo[?];
  int     ap_arch;
  BD_INT  longitud;
  BD_INT  cantidad;
  char    *arreglo;
  BD_INT  libres;
  BD_INT  despl[BD_NO_CAMPOS+1];
  char    mascara[BD_NO_CAMPOS+1];
}bdd_registro[BD_NO_REGISTROS+1];
```

El primer miembro de la estructura es un apuntador a una estructura de protección que utilizan los procedimientos del manejador de la base para evitar que dos procesos escriban simultáneamente al mismo tipo de registro, solo es incluido en las declaraciones si en el encabezado de configuración de "GENERA" se contempló la ejecución del sistema bajo VAXELN.

Por cada tipo de registro conservamos la siguiente información: nombre de su correspondiente **archivo** en disco y en **ap_arch** el apuntador a archivo asignado durante la operación de apertura; en **longitud** el largo total en bytes de la estructura de datos propia del tipo de registro, y en **cantidad** el número de elementos (registros) del arreglo de estructuras apuntado por ***arreglo**; un BD_INT **libres** para el identificador de índice "enum" que mantiene ligados los registros que no tienen información (vacíos o libres).

Dos arreglos, **despl[]** y **mascara[]** que contienen respectivamente el número de byte donde comienza cada campo dentro de la estructura (considerando que el primero es cero) y la máscara que se debe aplicar (si el campo es tipo bit); por ejemplo, si de la lista completa de campos:

```
#define     TIPO     LONGITUD

CAMPO1      char        1
CAMPO2      char        1
CAMPO3      char        1
CAMPO4      bit         1
CAMPO5      bit         1
CAMPO6      bit         1
```

Solo son válidos para el tipo de registro REG1 los campos CAMPO1, CAMPO3, CAMPO4, CAMPO5; entonces "GENERA" declararía una estructura con los siguientes elementos:

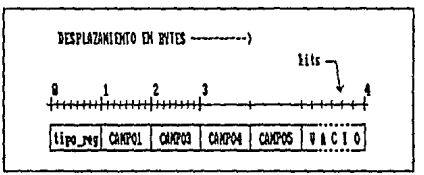


Figura 3.24
Estructura de un registro tipo REG1

Nótese como el primer byte es **tipo_reg** que como habíamos explicado se encuentra en todas las estructuras y contiene el identificador del tipo de registro, ahora bien, de acuerdo con lo explicado, despl[CAMPO2] y despl[CAMPO6] se inicializarán en cero ya que no son válidos en ese registro; mascara[CAMPO4] tendrá $0001_{binario}$ y mascara[CAMPO5] $0010_{binario}$ pues aplicando una operación AND entre el tercer byte y estas máscaras obtenemos el estado del bit correspondiente.

```
despl[CAMPO1]= 1        mascara[CAMPO1]= 0
despl[CAMPO2]= 0        mascara[CAMPO2]= 0
despl[CAMPO3]= 2        mascara[CAMPO3]= 0
despl[CAMPO4]= 3        mascara[CAMPO4]= 1
despl[CAMPO5]= 3        mascara[CAMPO5]= 2
despl[CAMPO6]= 0        mascara[CAMPO6]= 0
```

c) Descriptor de índices

```
struct bd_desc_indice {
  struct bd_struct_prot  *prot;
  BD_INT   tipo;
  BD_INT   no_niveles;
  BD_BIT   orden:1;
  BD_BIT   llave_unica:1;
  struct bd_nodo   *raiz;
  int      (*func)();
  BD_INT   campo_parte;
  char     campo_llave[BD_NO_CAMPOS+1];
}bdd_indice[BD_NO_INDICES+1];
```

Al igual que en los descriptores de registros, el primer miembro es el apuntador a la estructura de protección y su objetivo e inclusión siguen los mismos lineamientos.

De cada índice se mantiene: **tipo** (BD_STRING, BD_INT, BD_ENUM o BD_USER), **no_niveles** (vale 1 para los de tipo "enum", 4 para los "int" y tendrá el largo de la llave para los "string"), "**orden**" (1 para ascendente, 0 descendente), **llave_unica** que indica en índices "string" o "int" si está permitido (1) o prohibido (0) el duplicar llaves, un apuntador ***raiz** al primer nodo del índice, un apuntador ***func** a la función que el usuario debe proporcionar y que los procedimientos del manejador de la base de datos llaman automáticamente para obtener la llave correspondiente a un registro, tal función debe tener la siguiente sintaxis:

```
nombre_funcion (llave,reg)
   unsigned char  *llave;
   char           *reg;
```

El programador deberá hacer *tantas* funciones como índices "string" o "int" declare en el archivo-esquema y en ellas debe copiar al área apuntada por ***llave** la cadena o el entero de dos bytes que corresponde como llave al registro ***reg**, en la escritura de estas funciones puede usar los procedimientos de alto nivel que en 3.2 *El Manejador de la Base de Datos* abordaremos; cabe señalar que si bien es posible crear o calcular llaves a partir de varios campos, en la mayoría de los casos la llave de acceso es exactamente igual a un campo de la base, siendo así, la función puede ser tan simple como:

```
identif_kw(llave,reg)
  unsigned char  *llave;
  char  *reg;
{
  bd_ccamp_reg(llave,reg,IDENTIFICADOR);
}
```

d) Descriptor de enfoques

```
struct bd_desc_enfoque {
  BD_INT  no_indice;
  char    *registro;
  char    filtro_activo;
  unsigned char          filtro[?];
  unsigned char          llave[?];
  struct bd_desc_indice  *indice;
  struct bd_nodo         *posicion;
}bdd_enfoque[BD_NO_ENFOQUES+1];
```

Los *"enfoques"* son las estructuras base para los procedimientos de alto nivel, un proceso puede tener tantos enfoques como sea necesario pero los debe usar de manera exclusiva ya que no tienen protección de acceso a escritura concurrente con objetos MUTEX como sucede en el caso de registros e índices de la base.

Un enfoque puede tener asignado un índice, en cuyo caso **no_indice** será diferente de cero y equivalente al identificador #define del mismo, además, el ***indice** apuntará al descriptor correspondiente a ese índice.

En todo momento consideraremos que el *"registro actual"* es el apuntado por ***registro**; varias funciones del manejador leen o modifican este apuntador (bd_salta, bd_busca, etc.). Para facilitar y agilizar el movimiento secuencial dentro de un índice ***posicion** conserva un apuntador al nodo que apunta al registro actual.

La **llave** del registro actual se conserva en **llave[]** y existen procedimientos que activan (haciendo filtro_activo= 1) un filtrado de los registros del índice asignado de manera que solo pasan aquellos cuya llave sea igual a **filtro[]**; en el caso de los índices tipo "string" el filtro puede ser una cadena de menor tamaño que la llave, y solo serán filtrados los primeros bytes, por ejemplo, si las llaves de un grupo de registros indexados en un índice tipo "string" son:

```
"1L1020"
"1L2345"
"1L2598"
"1L4609"
"2V4533"
"2V4982"
```

Y si **filtro[]** contiene "1L2" entonces en el índice parecerán existir solamente los registros con las llaves "1L2345" y "1L2598".

Sintaxis del lenguaje descriptor de esquemas

Para los diagramas de sintaxis emplearemos por su claridad la notación gráfica de Wirth que frecuentemente se usa en los textos sobre lenguaje Pascal, sus elementos gráficos son:

Círculo etiquetado:
Contiene elementos propios del lenguaje.

Rectángulo etiquetado:
Representa otros diagrama sintácticos.

Grafo dirigido:
Liga círculos y/o rectángulos indicando el camino a seguir para la construcción de sentencias.

Punto de Derivación:
Marca la posibilidad de tomar dos o más caminos durante el flujo de la construcción sintáctica.

Adicionalmente:
Ya que nuestro lenguaje-esquema se basa en tablas ordenadas por columnas, en la descripción sintáctica de cada una, las columnas se marcarán en la parte superior de la descripción con etiquetas delimitadas por líneas punteadas.

1) Comentarios

Cuando el traductor detecta el caracter ';', ignora todos los que le sigan hasta encontrar fin de línea ('\n').

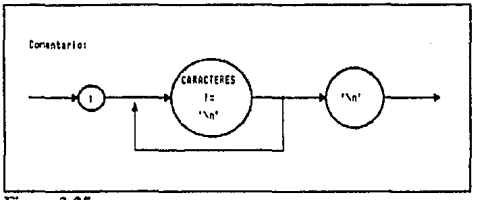


Figura 3.25
Comentarios

2) Inicio y fin de tablas

El archivo-esquema debe tener cinco tablas: CAMPOS, REGISTROS, INDICES, ENFOQUES e INCLUIR; en las primeras cuatro se describen los elementos lógicos de la arquitectura y en la última indicamos el o los archivos con las funciones de usuario; el traductor reconoce el inicio y fin de estas tablas por:

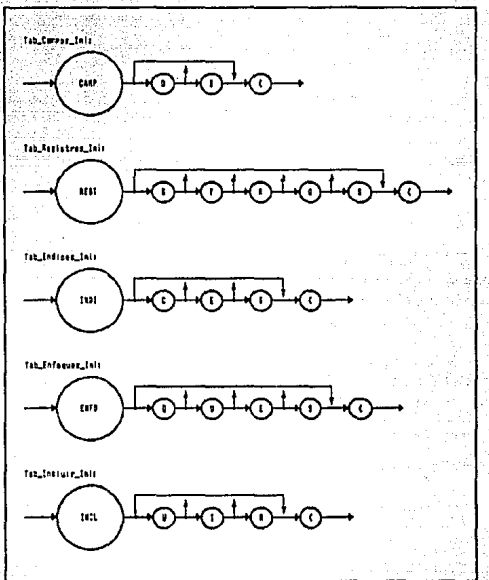


Figura 3.26
Identificación del inicio de las tablas

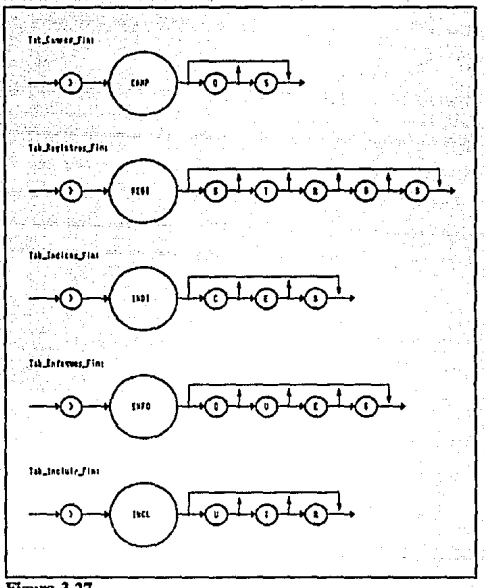


Figura 3.27
Identificación del fin de las tablas

3) Palabras

Durante la traducción del archivo-esquema en muchos casos buscamos "palabras", si bien no hay regla que impida mezclar letras mayúsculas y minúsculas, es recomendable ocupar un solo tipo de letra para los siguientes tipos de palabras:

MAYUSCULAS	MINUSCULAS
Identificadores	Nombres
Anillo_libres	Arreglos
Campo_perte	Archivos
	Funciones

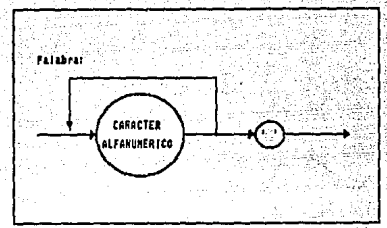


Figura 3.28
Palabras

4) Enteros

Para que el traductor reconozca valores numéricos (solo enteros) debe seguirse la sintaxis expuesta en la figura 3.29.

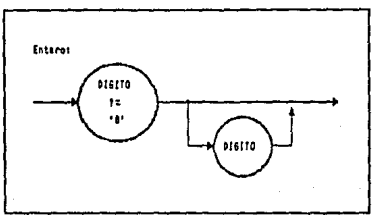


Figura 3.29
Enteros

5) Tabla "CAMPOS"

En la Figura 3.30 queda expresado que para los campos de tipo "string" y "bit" debe proporcionarse su longitud en bytes o bits respectivamente; si el tipo del campo es "reference" no es necesaria la longitud pero si lo es el identificador del registro y del campo apuntado. Para los otros tipos debe indicarse '-' en las columnas Longitud, Campo_ref y Reg_ref.

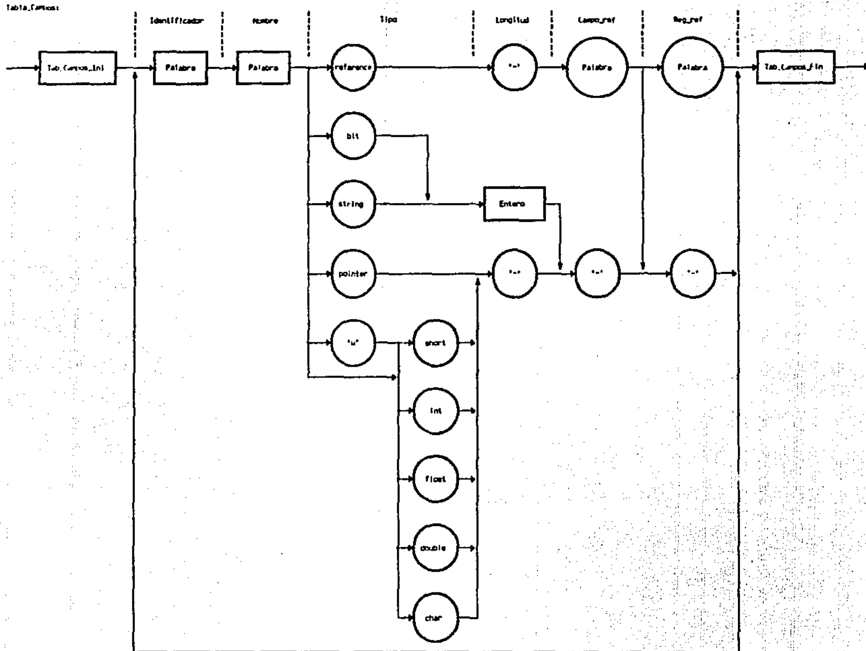

Tabla_Campos:
Identificador
Nombre
Tipo
Longitud
Campo_ref
Reg_ref
Tab_Campos_Ini
Palabra
Palabra
reference
Palabra
Palabra
Tab_Campos_Fin
bit
string
Entero
pointer
short
int
float
double
char

Figura 3.30

6) Tabla "REGISTROS"

Observemos como dentro de esta tabla (Figura 3.31) se busca por el inicio y fin de la tabla CAMPOS sin embargo los campos, aunque deben su existencia al hecho de ligarse con registros, son hasta cierto punto independientes de ellos y ya fueron definidos en su tabla, por ello solo se deben indicar los campos válidos para cada registro en lo que podríamos llamar la subtabla REGISTROS:CAMPOS.

7) Tabla "INDICES"

En la Figura 3.32 vemos como la sintaxis está íntimamente ligada a las características descritas en la sección 3.1.2 (*Estructuras de Datos*). En este caso encontramos dos subtablas: INDICES:CAMPOS e INDICES:REGISTROS en la primera hacemos referencia a los identificadores de campo que al ser modificados pueden alterar la ubicación o pertenencia de un registro al índice, en otras palabras se incluyen los campos que forman parte de la llave, por tanto, solo en los índices tipo "string" e "int" se tomarán en cuenta dichos identificadores; en la subtabla INDICES:REGISTROS se especifican los identificadores de aquellos registros cuya indexación dentro del índice es válida, el traductor del archivo-esquema declarará automáticamente en cada registro indicado un campo adicional, **Campo_perte**.

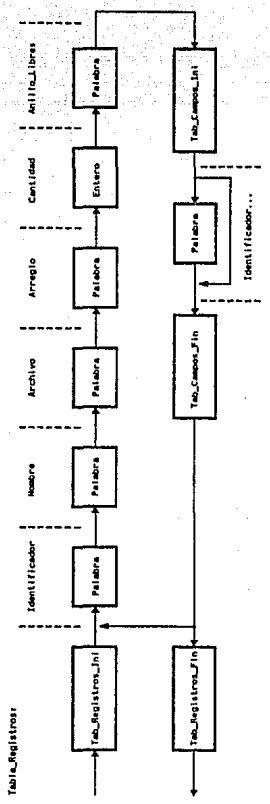

Tabla_Registros
Tab_Registros_Ini
Tab_Registros_Fin
Identificador
Palabra
Nombre
Palabra
Archivo
Palabra
Arreglo
Palabra
Cantidad
Entero
Anillito_Libres
Palabra
Tab_Campos_Ini
Palabra
Identificador...
Tab_Campos_Fin

Figura 3.31

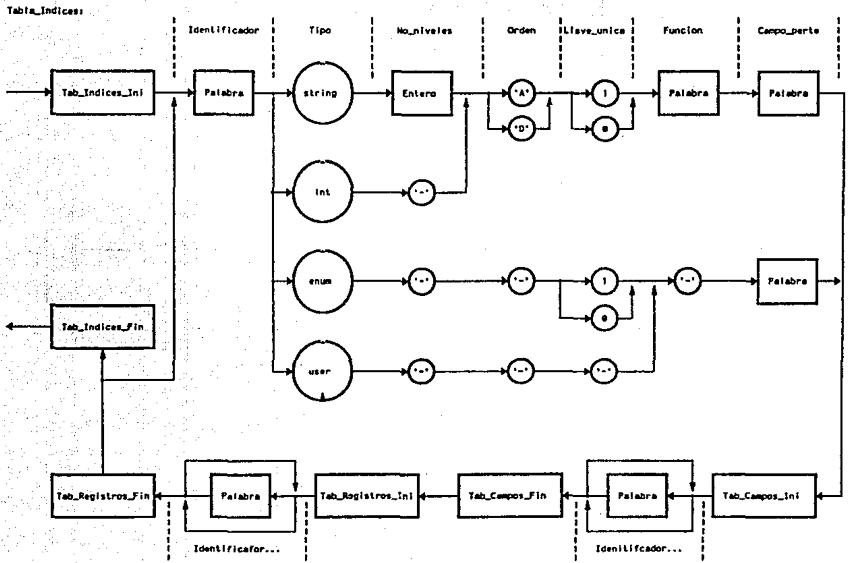

Tabla_Indices:
Identificador
Tipo
No_niveles
Orden
Llave_unica
Funcion
Campo_perte
Tab_Indices_Ini
Palabra
string
Entero
"A"
"D"
1
0
Palabra
Palabra
int
'_'
enum
'_'
'_'
1
0
'_'
Palabra
user
'_'
'_'
'_'
Tab_Indices_Fin
Tab_Registros_Fin
Palabra
Tab_Registros_Ini
Tab_Campos_Fin
Palabra
Tab_Campos_Ini
Identificafor...
Idenitifcador...

Figura 3.32

8) Tabla "ENFOQUES"

En esta tabla solo incluimos los identificadores de cada enfoque ya que estos elementos lógicos no tienen características configurables, nótese como al menos debe existir el enfoque SISTEMA, mismo que está reservado para algunos procedimientos del manejador.

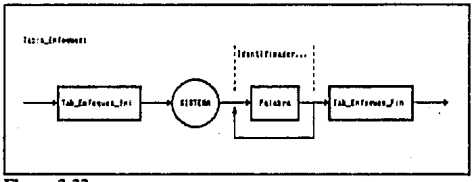


Figura 3.33
Tabla ENFOQUES

9) Tabla "INCLUIR"

Se deben indicar los nombres de archivos que contienen las funciones para el cálculo de las llaves de índices y de acceso de bajo nivel.

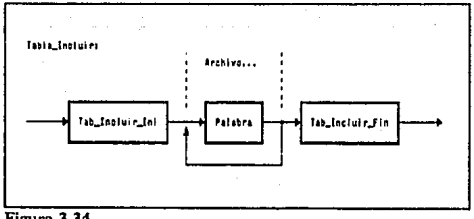


Figura 3.34
Tabla INCLUIR

10) Archivo-Esquema

Por último, dados los anteriores diagramas, el esquema es resultado de reunir las cinco tablas:

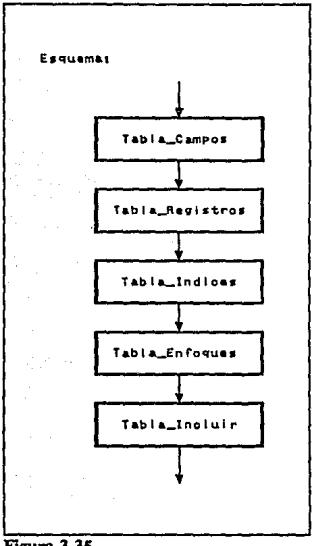


Figura 3.35
ESQUEMA

Encabezados de configuración

1) GENERA.H

Aprovechando la directiva de preprocesador #include, las características del traductor se pueden y deben ajustar para trabajar en el ambiente y con el archivo-esquema en particular editando el encabezado GENERA.H; un ejemplo de tal archivo es:

```
/* Archivo GENERA.H (configuración para GENERA.C) */

/* QC
#define  BD_OPEN_RDWR  (0x0002|0x8000) */
/* TC
#define  BD_OPEN_RDWR  (0x0004|0x8000) */
/* VMS */
#define  BD_OPEN_RDWR  (01000|002)
/* VAXELN */
#define  BD_VAXELN  1

#define  MAX_CAR_ID      25
#define  MAX_CAR_ARCH    15
#define  MAX_CAMPOS     100
#define  MAX_REGISTROS   18
#define  MAX_INDICES      7
#define  MAX_ENFOQUES     8
```

En las primeras líneas tenemos distintos valores para la opción de apertura de archivo en los tres compiladores que actualmente soporta la versión de GENERA.C, además cuando vamos a correr la aplicación en VAXELN, se debe descomentar la línea correspondiente para que el programa agregue objetos MUTEX a las declaraciones de los descriptores de registros.

En las últimas 6 líneas se especifican características propias del archivo-esquema a traducir:

MAX_CAR_ID:	Cantidad máxima de caracteres que pueden tener los identificadores y nombres ya sea de campos, registros, índices o enfoques.
MAX_CAR_ARCH:	Longitud máxima que puede tener el nombre de los archivos en disco.
MAX_CAMPOS:	Cantidad máxima de campos e índices que se van a declarar en el archivo-esquema.
MAX_REGISTROS:	Cantidad máxima de tipos de registros declarados.
MAX_INDICES:	Cantidad máxima de índices (considerando todos los tipos, "string", "int", "enum" y "user") declarados.
MAX_ENFOQUES:	Cantidad máxima de enfoques que se van a usar.

2) BD_FUNC.H

Tres características del manejador de la base se determinan en este encabezado, resulta claro que una vez compilado el módulo "bd_func.c" se mantendrán:

* Debug: Es recomendable que durante el desarrollo se active, ya que son compilados en las funciones filtros de verificación de validez en los parámetros pasados como identificadores de enfoques, campos, etc. En la versión depurada no debe estar presente pues afecta negativamente la velocidad de respuesta.

* Mensajes: Aunque todas las funciones del manejador informan sobre el resultado de su actividad, suele ser necesaria durante el desarrollo información detallada; activando mensajes y controlando el nivel de depuración con la función bd_debug() se obtienen paso a paso en pantalla los códigos de error y su descripción.

* VAXELN: Ya que la protección por objetos MUTEX implica llamada a funciones exclusivas de la biblioteca del VAXELN, para evitar errores durante el ligado de la aplicación en otros ambientes debe desactivarse.

El siguiente es un listado-ejemplo del archivo encabezado BD_FUNC.H:

```
/* Archivo BD_FUNC.H (para configurar BD_FUNC.C) */

#define  MAXLONGLLAVE  10

/* Para eliminar Debug, comentar la siguiente línea */
#define  DEBUG  1

/* Para desactivar mensajes de error,
   comentar las dos siguientes líneas */
#define  BUGMES  1
#define  rtn(coderr)  return(bd_debug(coderr))

#ifndef  BUGMES
  #define  rtn(valor)   return(valor)
#endif

/* VAXELN */
#define  BD_VAXELN  1
```

Algoritmo general del traductor de esquemas

Conociendo la sintaxis del lenguaje-esquema los detalles de funcionamiento del programa *GENERA* son transparentes durante el diseño de la base de datos, sin embargo, esta herramienta aunque de caracter general fue originalmente ideada para automatizar el paso de esquema lógico a implantación física de la base de datos para la IHM de la CCC Gómez Palacio; por ello mostramos en esta sección los algoritmos involucrados en el traductor empleando diagramas de cajas[11](también conocidos como *diagramas de Nassi-Schneiderman*); sus elementos gráficos se muestran en la Figura 3.36.

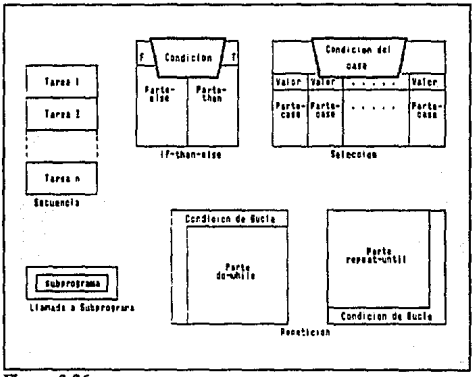


Figura 3.36
Diagramas de cajas: Elementos gráficos

Las estructuras de datos fundamentales que contiene el programa *GENERA* reflejan directamente la constitución de las tablas en archivos-esquema, cada columna de cada tabla pasa a ser un elemento dentro de su estructura correspondiente:

```
struct {
  char    id[MAX_CAR_ID];
  char    nombre[MAX_CAR_ID];
  char    tipo;
  int     longitud;
  char    reg_ref[MAX_CAR_ID];
  char    campo_ref[MAX_CAR_ID];
}campo[MAX_CAMPOS];
```

[11]Pressman, R.S., *Ingeniería del Software*, p270

```
struct {
  char    id[MAX_CAR_ID];
  char    nombre[MAX_CAR_ID];
  char    archivo[MAX_CAR_ARCH];
  char    arreglo[MAX_CAR_ID];
  int     cantidad;
  char    libres[MAX_CAR_ID];
  char    campos[MAX_CAMPOS][MAX_CAR_ID];
}registro[MAX_REGISTROS];

struct {
  char    id[MAX_CAR_ID];
  char    tipo;
  char    no_niveles;
  char    orden;
  char    llave_unica;
  char    func[MAX_CAR_ID];
  char    campo_perte[MAX_CAR_ID];
  char    campos[MAX_CAMPOS][MAX_CAR_ID];
  char    registros[MAX_REGISTROS][MAX_CAR_ID];
}indice[MAX_INDICES];

char  enfoque[MAX_ENFOQUES][MAX_CAR_ID],
      incluir[MAX_ENFOQUES][MAX_CAR_ID];
```

En un primer nivel *GENERA* se describe según la Figura 3.37.

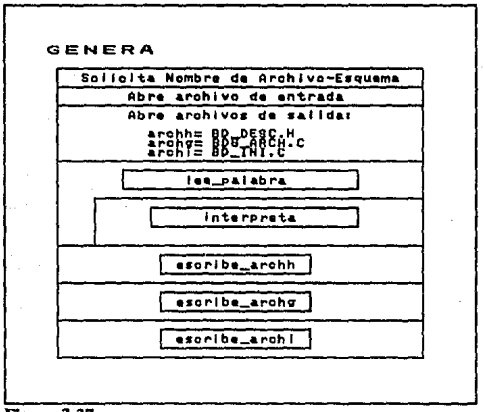


Figura 3.37
Diagrama general de GENERA

Las Figuras 3.38 a 3.41 muestran en niveles de detalle más bajos cada módulo.

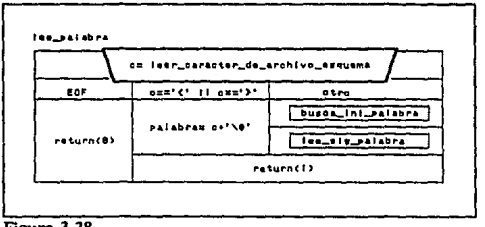


Figura 3.38
lee_palabra

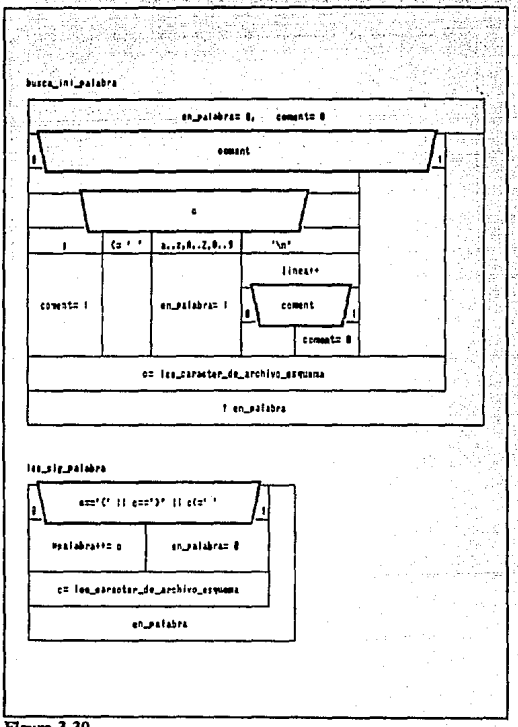


Figura 3.39
Diagrama para busca_ini_palabra y lee_sig_palabra

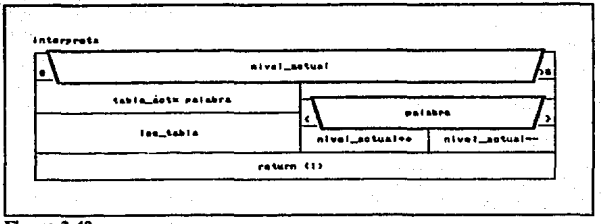


Figura 3.40
interpreta

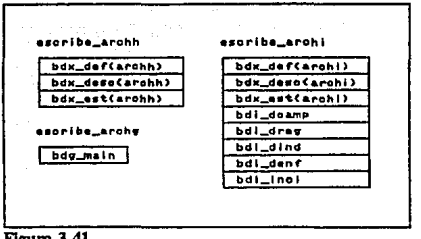


Figura 3.41
Creación de archivos

Por último damos la descripción textual de las funciones involucradas:

lee_tabla(): Según el valor de "tabla_act" se leen tantas palabras como elementos tenga la estructura respectiva, de modo que cada columna corresponda a un elemento. por ejemplo, si tabla_act == TAB_CAMP se leerán 6 palabras para llenar los elementos campo[n].id, campo[n].nombre, campo[n].tipo, campo[n].longitud, campo[n].reg_ref, campo[n].campo_ref

bdg_main(): Escribe en el archivo BDG_ARCH.C la función main() en la cual se abren tantos archivos como tipos de registro con el nombre especificado en el esquema y se escriben en cada uno "registro[n].cantidad" registros en blanco.

bdx_def(arch): Escribe en el archivo "arch" los #define tanto de tipo predefinido como definibles[12]

[12]Vid sección 3.1.2, *Identificadores*

bdx_desc(arch): Escribe en "arch" las declaraciones de descriptores[13] ajustando el tamaño de los arreglos de acuerdo a la cantidad de elementos (campos, registros, índices, etc) declarada en el esquema.

bdx_est(arch): En "arch" escribe las declaraciones adecuadas para los tipos de registro[14] tomando en cuenta los campos válidos en cada uno.

bdi_dcamp(),bdi_dreg(),bdi_dind(),bdi_denf(): Escriben en el archivo BD_INI.C la inicialización de cada tipo de descriptor, por ejemplo, para los descriptores de campos los valores de inicialización de cada uno se toman de la estructura campo[n].id, campo[n].nombre, etc.

bdi_incl(): En el archivo BD_INI.C se escriben líneas de la forma:

```
#include %s
```

Donde "%s" se reemplaza por los nombres de archivo declarados en la tabla INLCUIR del archivo esquema.

Esquema propuesto

El listado completo del archivo-esquema se encuentra en el Apéndice A; la tabla CAMPOS se construyó a partir de la información recabada en 2.2.2 (*Atributos*), asi mismo, la tabla REGISTROS resulta de los aspectos considerados en la sección 2.2.1 (*Entidades*).

En cuanto a los índices, se declararon cinco pero cabe señalar que el mismo nivel de flexibilidad que permite el lenguaje-esquema para manejar campos, lo tenemos en la creación de índices, en un futuro es posible agregar otros sin alterar realmente el desempeño del sistema.

```
NOMBRE          TIPO

IND_DIRHW       user
IND_IDENTIF     string
IND_CRONOAL     enum
IND_PRIORAL     int
IND_SISYSUB     string
```

El acceso principal a la información es a través del índice de usuario IND_DIRHW (tipo "user") exhaustivo construido en base a un arreglo de apuntadores a estructuras declarado de la siguiente forma:

```
char  *dirhw[23,16,64];
```

---

[13] Vid sección 3.1.2, *Elementos Lógicos y sus Descriptores*

[14] Vid sección 3.1.2, *Estructuras de Registros*

En este arreglo la primera dimensión corresponde a número de canasta, la segunda a ranura y la última a señal; es inicializado por medio de una rutina cuyo pseudo-código es el siguiente:

```
Igualar todo elemento de dirhw[] a NULL;
Activar índice IND_IDENTIF;
Salto al primer elemento del índice;
REPETIR
   Canasta= valor del campo CANASTA del elemento actual;
   Ranura=  valor del campo RANURA  del elemento actual;
   Senal=   valor del campo SENAL   del elemento actual;
   dirhw[Canasta,Ranura,Senal]= &(elemento actual);
   Siguiente elemento del índice;
MIENTRAS fin de índice sea falso;
```

Como vemos, la correcta inicialización depende de la existencia de otro índice, IND_IDENTIF en el cual se asignan todos los registros de variables del sistema y quedan ordenados por su campo IDENTIFICADOR, pero como no todas las direcciones hardware posibles tienen una variable relacionada, la asignación previa de NULL a todos los elementos de dirhw[] proporciona un método para identificar aquellas que son inválidas.

En cuanto IND_IDENTIF, tenemos que se trata de un índice tipo "string" de seis niveles (precisamente la longitud del campo IDENTIFICADOR sin tomar en cuenta el terminador nulo), su procedimiento de cálculo de llaves llama a una función del manejador de la base (las funciones serán tratadas al detalle en 3.2 *El Manejador de la Base*) que simplemente copia el valor de IDENTIFICADOR al área apuntada por 'llave'.

```
identif_kw(llave,reg)
  unsigned char  *llave;
  char  *reg;
{
  bd_ccamp_reg(llave,reg,IDENTIFICADOR);
}
```

Los índices IND_CRONOAL e IND_PRIORAL se usan en los desplegados de la función de presentación Diálogo de Alarmas para mostrar las variables ordenadas ya sea cronológicamente (conforme se han alarmado) o por prioridad. Ambos índices contendrán a lo más 100 elementos que es la longitud máxima para la lista de variables en los desplegados de alarmas. IND_CRONOAL no requiere de llave ya que es de tipo "enum" y simplemente liga los elementos a un contínuo, comportándose para la inserción igual que una pila. El índice IND_PRIORAL coloca sus elementos de acuerdo al valor para la llave que calcula el siguiente procedimiento:

```
prioral_kw(llave,reg)
  unsigned char  *llave;
  char  *reg;
{
  bd_ccamp_reg(llave,reg,PRIORIDAD);
}
```

En este procedimiento como también usamos un solo campo para el cálculo de la llave, simplemente se copia su valor al área apuntada por 'llave'.

El índice IND_SISYSUB es empleado por diversas funciones de presentación cuyo acceso a los registros de variables se hace por sistema y subsistema, el cálculo de su llave requiere de la lectura de tres campos; AP_SISTEMA, AP_SUBSISTEMA e IDENTIFICADOR.

```
sisysub_kw(llave,reg)
  unsigned char  *llave;
  char  *reg;
{
  llave[0]= (char)(bd_apunt_reg(reg,AP_SISTEMA)+1);
  llave[1]= (char)(bd_apunt_reg(reg,AP_SUBSISTEMA)+1);
  bd_ccamp_reg(llave+2,reg,IDENTIFICADOR);
}
```

En la función anterior se obtiene el valor de los apuntadores a los registros tipo SISTEMAS y SUBSISTEMAS, es decir, tomamos la posición dentro del arreglo de estructuras del sistema y subsistema asociado con el registro 'reg' (que corresponde al número de sistema o subsistema menos uno) y la pasamos como un byte (char) ya que la cantidad de sistemas y subsistemas es mucho menor a 255; después incluimos el IDENTIFICADOR y, como las funciones del manejador de la base hacen un riguroso ordenamiento tomando en cuenta el código ASCII en índices tipo "string" el resultado es la ordenación por SISTEMA en primer lugar, SUBSISTEMA en segundo e IDENTIFICADOR en último lugar.

En cuanto a los enfoques declarados además de BD_SISTEMA que es obligatorio para actividades propias del manejador de la base[15] se asigna uno por cada función de presentación (FP) y programa de infraestructura (PI) a excepción del Diálogo de Alarmas para el cual se asignan dos para mantener en cada uno los índices IND_CRONOAL e IND_PRIORAL y el de Arranque del Sistema que no tiene ninguno pues no se hace manejo sobre las variables en ese módulo:

```
ENFOQUE          Propietario (FP o PI)

SISTEMA          Manejador de la Base
PI_ADENDEME      Adquisición de Señales
PI_ALARMAS       Alarmas
PI_MSE           Monitor de Secuencia de Eventos
PI_HISTORIAS     Historias
PI_CALCULOS      Cálculos Especiales
PI_SPOOLER       Spooler
PI_FALLAS        Tolerancia a Fallas
PI_ADMONFP       Administrador de las FP
FP_LISVAR        Lista de Variables
```

[15]Vid 3.1.2 *Sintaxis del Lenguaje Esquema: Tabla Enfoques*

```
ENFOQUE          Propietario (FP o PI)

FP_DIALAR_C      Diálogo de Alarmas (IND_CRONOAL)
FP_DIALAR_P      Diálogo de Alarmas (IND_PRIORAL)
FP_GUIAS         Guías en Caso de Alarmas
FP_REPORTES      Reportes
FP_ARRYPAR       Arranque y Paro
FP_GRACOOR       Gráficas de Coordenadas
FP_DIAFLU        Diagramas de Flujo
FP_BARRAS        Diagramas de Barras
FP_DIATEN        Diagramas de Tendencias
FP_GRAPAP        Gráficas en Papel
FP_DIACON        Diálogo de Control
FP_DIASER        Diálogo de Servicios
```

3.1.3. Evaluación

En esta sección evaluaremos la arquitectura propuesta desde el punto de vista estructural, sin considerar aspectos como requerimientos de almacenamiento y velocidad de respuesta, temas que serán tratados en 3.3 (*Evaluación General de Desempeño*).

Los conceptos del diseño dirigido a objetos[16] influyeron notablemente en la concepción de la arquitectura, llevando a un tratamiento peculiar de las relaciones entidad-atributos. Tradicionalmente los lenguajes de descripción de datos (DDL) ligan fuertemente los registros con sus campos; para cada registro debemos describir detalladamente sus campos indicando nombre, tipo, longitud, etc., con objeto de ilustrar el problema que encierra este enfoque consideremos los registros R1 y R2 definidos conceptualmente como:

```
R1= C1, C2, C3
R2= C1, C2, C4
```

En algunos casos (p.ej. COBOL) la relación entre el registro y sus campos es tan estrecha que aunque dos registros tengan un atributo en común, nos vemos obligados a declarar nombres de campo diferentes para evitar repetición de símbolos con el consecuente error de compilación:

```
R1= C1_R1, C2_R1, C3_R1
R2= C1_R2, C2_R2, C4_R2
```

En otras ocasiones (p.ej. Clipper) el problema se aborda desde un punto de vista jerárquico, permitiendo en la declaración de la estructura de la base la repetición de nombres de campo en registros diferentes; gracias a ello la cantidad de identificadores con los que tiene que trabajar el programador se reduce considerablemente, sin embargo, en nuestro ejemplo la referencia a R1.C4 resultaría en un error en tiempo de ejecución.

---

[16] Pressman, R.S., *Ingeniería del Software*, p370-405

Estos problemas contrastan con el hecho de que en los lenguajes de manejo de datos (DML) se aplica el concepto de "registro actual", es decir, una vez encontrado un registro, el usuario puede solicitar al sistema los valores de sus campos sin necesidad de buscarlo cada vez; agreguemos a esto que en la cada vez más empleada técnica del diseño orientado a objetos, con el fin de lograr código flexible y reutilizable se prefieren métodos generales aplicables a diferentes objetos de una misma clase. Entonces, retomando el ejemplo, si el programador desea implementar una función o procedimiento general que afecte a los campos C1...C4 tendrá que dividir su código en dos partes, cada una de acuerdo al tipo de registro que reciba y si en un futuro quisiera hacer aplicable el código a otro tipo de registro tendría que modificarlo.

Las consideraciones anteriores desembocaron en la creación de la clase independiente "CAMPOS" que en el archivo-esquema se manifiesta como una tabla, comparando nuestro lenguaje esquema contra otros, tenemos que además de reducir la cantidad de identificadores definidos disminuye el monto de lenguaje descriptor necesario, se garantiza el que dos o más registros con atributos semejantes estén ligados a objetos (campos) con las mismas características y, lo más importante, permite escribir rutinas generales para el tratamiento de atributos. En el caso específico de bases para sistemas en tiempo real con tipos de registro de variables que comparten conjuntos de atributos como lo es la base para la IHM estudiada, la reducción en cantidad de código descriptor de la base disminuye hasta un 20% cuando menos.

Al separar la descripción de campos de sus registros resultó viable una opción que no encontramos en los sistemas de manejo de base de datos tradicionales; normalmente los objetos (registros) que podemos ligar a un índice tienen que ser del mismo tipo, deben tener la misma estructura (campos). La arquitectura propuesta permite definir métodos de cálculo de llave basados en valores de campos, así que un mismo índice puede directamente ligar toda una variedad de registros siempre que estos compartan los campos llave. Desafortunadamente, para obtener estas ventajas es necesario aprender el lenguaje-esquema y lidiar con las herramientas automatizadas para integrar el sistema pero el resultado final bien vale la pena.

Por otro lado, en cuanto a las relaciones entre entidades hay que reconocer que la definición de *Tipos de Conjunto* indispensable en cualquier enfoque de red aunque lograda por la aplicación conjunta de apuntadores "reference" e índices no queda explícitamente indicada en el lenguaje-esquema y si bien esto permite definir únicamente la relación de miembro a propietario o viceversa, cuando establecemos ambas, el enlace no resulta obvio; este problema será resuelto en versiones posteriores del lenguaje-esquema agregando una tabla descriptora de Tipos de Conjunto.

3.2. EL MANEJADOR DE LA BASE

Las herramientas que generan código en lenguaje C servirían de poco sin un conjunto de procedimientos estandarizados (manejador de la base de datos) para manipular la información, garantizando consistencia en la organización de los datos dentro de las estructuras declaradas; tales

procedimientos son las funciones del manejador de la base de datos que se concentran en el archivo "BD_FUNC.C"; para identificarlas el programador debe tener presente que por regla sus nombres empiezan con el prefijo 'bd_' mas cinco caracteres que detallan su propósito p.ej. bd_salta(), bd_xinde(), etc. y en algunos casos tres más indicando el objeto sobre el que actúan p.ej. bd_reemp_reg(), bd_ccamp_reg(), etc.

Como características generales del manejador tenemos:

1) Compatibilidad con las estructuras definidas por el traductor de archivos-esquema "GENERA".

2) Código fuente en C con llamada a funciones de la biblioteca estándar, probado en PC para los compiladores Turbo C 2.0, Quick C 1.0 y en VAX C; alto grado de portabilidad.

3) Inclusión opcional de objetos para la protección de escritura concurrente para aplicaciones que corran bajo el ambiente VAXELN.

4) Un solo conjunto de códigos de error, válido para todas las funciones.

5) Detección e impresión de mensajes de error y advertencia en tres niveles; apagado, encendido y errores de sistema.

6) Sin importar el nivel de detección de errores, todas las funciones informan si su conclusión fue exitosa.

7) Compilación opcional de algoritmos de validación de parámetros, útil durante el periodo de desarrollo.

Para su estudio, agrupamos las funciones en cuatro clases que corresponden a diferentes actividades del manejo de la base: Respaldo y Mecanismos de Recuperación, Organización de la Información, Operaciones de Consulta y Operaciones de Actualización. Unicamente la función relacionada con la detección de errores sale de este contexto y por tanto la explicamos a continuación:

ERRORES

Resumen:

Ajustar el nivel de detección de errores e imprimir el significado de los códigos de error.

Funciones: bd_debug()

Sintaxis:

```
BD_INT bd_debug(coderr)
  BD_INT  coderr;
```

Descripción:

Los valores que *coderr* puede tomar están definidos en el archivo de encabezado BDERR.H (un listado del mismo se encuentra en el Apéndice C); el comportamiento del sistema difiere dependiendo si al momento de compilación en el archivo de encabezado estaba definido BUGMES o no, esto es útil durante el periodo de desarrollo ya que asi, antes de regresar de cualquier función automáticamente será llamada **bd_debug()** con el código de error producido si es que hubo alguno. Con la misma función el programador puede cambiar el nivel de detección de errores pasándole como parámetro BD_BUGON, BD_BUGOFF o BD_BUGSYS con lo cual cambiamos respectivamente a impresión de todos los errores, ninguno o solo errores importantes como falla de solicitud dinámica de memoria, errores en el manejo de archivos, identificadores desconocidos, etc. Para optimizar el desempeño del sistema final no basta con evitar impresión de errores ajustando el nivel a BD_BUGOFF, es necesaria la recompilación sin definir BUGMES ni DEBUG[17].

Ejemplos:

```
    . . . .
    . . . .
bd_debug(BD_BUGON);    /* Activa impresión de errores */
    . . . .
bd_debug(BD_BUGOFF);   /* Desactiva impresión de errores */
    . . . .
```

[17]Vid sección 3.1.2, *Encabezados de Configuración*

3.2.1. Respaldo y mecanismos de recuperación

En un sistema de adquisición de datos la permanencia de la información tiende a los extremos: o rara vez se actualiza (límites, unidades, descripciones, etc.) o bien está sometida a cambios continuos (valores analógicos y lógicos adquiridos); la información adquirida y que por tanto cambia constantemente no requiere respaldarse pues de ser necesario el sistema solicitaría los valores más recientes, por otro lado los atributos de variables cuyo valor casi no cambia, pueden actualizarse (activando en una estación de trabajo la FP Diálogo de Servicios) de manera inmediata en disco, enviando un mensaje a todos los nodos y como en cada uno existe una copia de la base, la información queda suficientemente respaldada por redundancia, es decir que con la pérdida de un nodo por falla o mantenimiento en los nodos restantes no habrá pérdida de información.

Sin embargo existen algunos datos altamente dinámicos que no son adquiridos sino obtenidos en base a la historia del comportamiento de terceros, tal es el caso de variables calculadas de la base de datos de la CCC de Gómez Palacio que tienen asociada alguna tabla de integraciones; dado lo anterior el problema más importante desde el punto de vista de respaldo se ubica en el universo de datos manejado por el módulo de infraestructura "Cálculos Especiales", estos registros son actualizados contínuamente y deben conservarse, para ello los registros de variables calculadas (ANALOGICAS_CALC) y las estructuras de integración (INTEGRACIONES) serán escritos a sus archivos en disco cada 30 segundos.

Resumiendo, tenemos tres niveles de respaldo de acuerdo con el tipo de atributos:

1) Atributos Adquiridos: No hay respaldo, son adquiridos cada vez que cambian o por solicitud a canastas.

2) Atributos Constantes: Modificaciones ocasionales, respaldo a disco inmediato.

3) Atributos Históricos: Comportamiento dinámico, respaldo cada 30 segundos.

La carga de la base de datos en un nodo toma uno de dos caminos de acuerdo con el estado de la red: si el sistema en conjunto está fuera (arrancar todos los nodos) el nodo efectuará un arranque en frío, los atributos del tipo (2) y (3) bajarán a la base en RAM de los archivos en su disco y los del tipo (1) serán solicitados a las canastas; si el sistema está funcionando el nodo realizará un arranque en caliente, situación en la que los atributos del tipo (1) serán también solicitados a las canastas pero los del tipo (2) y (3) serán recuperados de la información existente en los archivos de otro nodo.

Para llevar a cabo las actividades antes descritas el manejador de la base de datos proporciona herramientas para lectura completa de todos o un tipo de registro y escritura de todos un tipo o incluso un registro específico.

ARCHIVOS

Resumen:

Transferencia de información entre disco y RAM.

Funciones:

bd_carg00, bd_carg10, bd_cier00, bd_cier10, bd_escr00, bd_escr10, bd_escr20

Sintaxis:

```
BD_INT bd_carg0()

BD_INT bd_carg1(id_reg)
  BD_INT  id_reg;

BD_INT bd_cier0()

BD_INT bd_cier1(id_reg)
  BD_INT  id_reg;

BD_INT bd_escr0()

BD_INT bd_escr1(id_reg)
  BD_INT  id_reg;

BD_INT bd_escr2(id_e,reg)
  BD_INT  id_e;
  char    *reg;
```

Descripción:

La función **bd_carg10** carga del archivo en disco adecuado el tipo de registro especificado en *id_reg* a memoria RAM en su arreglo de estructuras correspondiente, si el archivo es mayor que 16 Kbytes la transferencia se realiza en bloques de 16K, al efectuar la carga cada registro es ligado automáticamente al (o los) índice(s) donde se encontraba en la última operación de cierre de archivo; es decir que el archivo de datos queda abierto con objeto de acelerar posteriores accesos al mismo y solo la llamada explícita a **bd_cier00** o a **bd_cier10** libera los buffers asociados. La función **bd_carg00** carga todos los tipos de registro especificados en el archivo-esquema mediante llamadas a la función anterior.

bd_cier10 cierra el archivo correspondiente a *id_reg* mientras que **bd_cier00** cierra todos los archivos de datos de los tipos de registro declarados en el archivo-esquema.

Una vez abiertos los archivos son posibles tres niveles de actualización de la información en disco; en el primero está la función bd_escr2() que escribe únicamente el registro apuntado por *reg* en el archivo adecuado, si *reg* es NULL entonces toma el registro actual del identificador de enfoque *id_e*. Por otro lado, la función bd_escr1() actualiza completamente el archivo del tipo de registro *id_reg* transfiriendo en bloques de 16K su arreglo de estructuras; en último lugar tenemos a bd_escr0() que llama a la función anterior tantas veces como tipos de registro se hayan declarado para actualizar todos los archivos de registros.

Estas funciones detectan el tipo de registro automáticamente y reconocen su archivo asociado por el nombre guardado en bdd_registro[*id_reg*].archivo, donde *id_reg* es el identificador del registro (que corresponde al valor absoluto del primer byte en cualquier registro); para realizar escritura o lectura en otros nodos de la red el programador debe alterar esta cadena a fin de que incluya en la ruta el nombre del nodo correspondiente.

Ejemplos:

A continuación mostramos algunas porciones de código de una aplicación sencilla; empezamos por inicializar el ambiente para trabajar con la base e inmediatamente se cargan todos los registros de disco, proceso durante el cual son generados automáticamente los índices conteniendo los registros que estaban indexados en la última escritura. Líneas abajo si es necesario recuperar los registros de variables análogicas calculadas llamamos a la función **bd_carg1()**, posteriormente si la FP Diálogo de Servicios modifica un registro se llama a bd_escr2() para actualizar la información en disco y, por último se graban en disco los registros de variables calculadas, se cierran todos los archivos y damos fin al ambiente de la base.

```
    . . . .
    . . . .
  bd_inicia_bd();  /* Inicialización */
  bd_carg0();      /* Lee todos los registros de disco */
    . . . .
    . . . .
/* Lee de disco únicamente los registros
   tipo 'análogicas_calculadas'          */
  if (recuperar_calc)
    bd_carg1(ANALOGICAS_CALC);
    . . . .
    . . . .
    . . . .
/* Guarda en disco el registro apuntado por 'regis' */
  if (diaser_mod)
    bd_escr2(FP_DIASER,regis);
    . . . .
    . . . .
/* Guarda en disco las variables calculadas */
  bd_escr1(ANALOGICAS_CALC);
  bd_escr1(LOGICAS_CALC);
    . . . .
  bd_cier0();    /* Cierra todos los archivos */
  bd_fin_bd();   /* Concluye uso de la base */
    . . . .
    . . . .
```

3.2.2. Organización de la información

Los registros quedan contenidos en sus respectivos arreglos de estructuras y se organizan por medio de índices cuya manipulación tiene tres niveles: automática, programable de tipo preconstruido y programable de tipo "user"; en el modo automático las operaciones sobre índices son completamente transparentes para el programador y no hay indicio de llamada directa a ninguna función, como ejemplo tenemos los procedimientos de agregado y eliminación de registros (**bd_agreg()**, **bd_borra()**) que internamente desindexan o indexan registros del índice tipo "enum" que mantiene ligados los registros libres de cada tipo, también las funciones para actualizar valores de atributos (**bd_reemp()**, **bd_copia()**) modifican la posición o pertenencia del registro a índices preconstruidos cuando las alteraciones a campos llave asi lo requieren.

En la manipulación programable de tipo preconstruido (índices tipo "string", "int" o "enum") podemos agregar o eliminar registros a un índice y activar filtros,[18] además, como posteriormente veremos en 3.2.3 *Operaciones de Consulta* existen funciones para recuperar registros aleatoria o secuencialmente (**bd_busca()**, **bd_salta()**).

En el manejo programable de tipo "user" **UNICAMENTE** la función que asigna un índice a un enfoque (**bd_asign()**) es válida, es decir que para ligar un registro a este tipo de índices el programador debe hacer sus propias funciones, en nuestro caso el índice IND_DIRHW es inicializado con el algoritmo descrito en 3.1.2 *Esquema Propuesto*; igualmente actividades como desligar registros, filtrar o reorganizar deben definirse y programar en una biblioteca por separado. Es importante hacer notar que la manipulación automática de índices **NO** trabaja sobre los de tipo "user"; por ejemplo, si cambiamos en la dirección hardware de una variable el valor de SENAL, aun llamando a la función proporcionada con ese propósito (**bd_reemp()**), solamente los índices de tipo preconstruido que usen ese campo para calcular su llave serían actualizados mientras que IND_DIRHW debe reorganizarse con procedimientos específicos.

En contraste, si el programador recurriendo a procedimientos de bajo nivel, entra por los arreglos de estructuras para modificar directamente el valor de un campo llave, **NINGUN** índice es actualizado siendo su responsabilidad actualizar los de tipo "user" con procedimientos propios y los preconstruidos ya sea con llamadas a **bd_index()** y **bd_xinde()** o aplicando la función **bd_perte()**.

---

[18] Vid sección 3.1.2, *Elementos logicos y sus Descriptores*

RELACION ENTRE INDICES Y ENFOQUES

Resumen:

Asignar un índice a un enfoque, activar filtrado de llaves.

Funciones: bd_asign(), bd_filtr()

Sintaxis:

```
BD_INT bd_asign(id_e,id_i)
  BD_INT  id_e;
  BD_INT  id_i;

BD_INT bd_filtr(id_e,llave)
  BD_INT  id_e;
  unsigned char  *llave;

BD_INT bd_llave(id_e,llave)
  BD_INT  id_e;
  unsigned char  *llave;

char *bd_regis(id_e)
  BD_INT  id_e;
```

Descripción:

bd_asign() asigna el índice *id_i* al enfoque *id_e*, en todo momento la última llamada a la función sobre un enfoque determinado es la única válida, es decir que a cada enfoque puede asignársele **SOLAMENTE** un índice pero por el contrario, un mismo índice puede estar asignado a varios enfoques y en cada uno presentar diferentes filtros y posiciones de registro actual, recordemos que los programadores pueden usar de manera exclusiva uno o más enfoques. Internamente **bd_asign()** altera algunos elementos de la estructura del descriptor de enfoque para relacionarla con el índice:

```
bdd_enfoque[id_e].no_indice= id_i;
  /* Número de índice */
bdd_enfoque[id_e].indice= &bdd_indice[id_i];
  /* Apuntador al descriptor del índice */
bdd_enfoque[id_e].posicion= ?;
bdd_enfoque[id_e].registro= ?;
bdd_enfoque[id_e].llave= ?;
  /* Para índices tipo "string" e "int" los miembros registro
       posición y llave se inicializan con los siguientes valores:
       apuntador al primer registro indexado, apuntador al nodo de
       dicho registro y llave de acceso
     Para índices "enum" se inicilizan con cero */
```

La función **bd_filtr()** caracteriza una relación entre enfoque e índice previamente establecida sin alterar realmente a este último, con ella el programador enmascara los elementos de acuerdo a su llave. Solo es aplicable a índices tipo "int" y "string"; cuando el índice asociado al enfoque *id_e* es de tipo "int" todos los registros cuya llave sea diferente a lo que apunta *llave*, serán -filtrados-, aparentemente desindexados, efecto igualmente obtenido si el índice es de tipo "string" y la longitud de la cadena apuntada por *llave* es igual al número de niveles del índice, cuando su longitud es mayor la función regresa un código de error y si es menor el filtrado se hace únicamente sobre la cantidad de caracteres que contenga la cadena (ver ejemplo en 3.1.2, *Elementos logicos y sus Descriptores*). Por otro lado para desactivar el filtro basta con pasar NULL en el parámetro llave.

Este método lo ocupan algunos programas como Lista de Variables donde para consultar las variables de un sistema y subsistema específico se filtran los primeros dos caracteres del índice IND_SISYSUB.

Las funciones **bd_llave()** y **bd_regis()** informan respectivamente la llave y la dirección en memoria del primer byte de la estructura del registro actual para el enfoque *id_e*.

Ejemplos:

El siguiente segmento de código asigna a los enfoques de la FP Lista de Variables y del PI (programa de infraestructura) Adendeme el índice de variables por identificador (IND_IDENTIF); observemos que si bien un enfoque solo puede tener asignado un índice, a varios enfoques se les puede asignar el mismo índice y cada uno de ellos tendrá su particular posición dentro del mismo. (es decir, cada uno tendrá su propio "registro actual") Posteriormente, se activa el filtro adecuado en el enfoque de Lista de Variables según sea una variable lógica o analógica de TG2; nótese que este filtrado solo afecta al enfoque FP_LISVAR, PI_ADENDEME sigue "viendo" todos los registros. Líneas abajo reasignamos el enfoque FP_LISVAR al índice de variables por sistema y subsistema, probablemente para generar un listado ordenado por esos campos. Por último, imprimimos la llave del registro actual del enfoque PI_ADENDEME.

```
    . . . .
    . . . .
  char  llave_actual[10];  /* Area para capturar llaves */
    . . . .
    . . . .
/* Asignación de índices a enfoques */
  bd_asign(FP_LISVAR,IND_IDENTIF);
  bd_asign(PI_ADENDEME,IND_IDENTIF);
    . . . .
    . . . .
/* Filtrado del índice por identificador de FP_LISVAR */
  if (var_tg2) {
    if (var_logica)
      bd_filtr(FP_LISVAR,"2L2");
    else
      bd_filtr(FP_LISVAR,"2V2"); }
    . . . .
    . . . .
/* Reasignación al índice por sistemas y subsistemas */
  bd_asign(FP_LISVAR,IND_SISYSUB);
    . . . .
    . . . .
/* Captura e impresión de la llave actual */
  bd_llave(PI_ADENDEME,llave_actual);
  printf("(ADENDEME) ");
  printf("Llave del registro actual: %s\n",llave_actual);
    . . . .
    . . . .
```

RELACION ENTRE INDICES Y REGISTROS

Resumen:

Indexar, desindexar y verificar pertenencia de registros a un índice.

Funciones: bd_index(), bd_xinde(), bd_xindx(), bd_perte()

Sintaxis:

```
BD_INT bd_index(id_e,reg,modo_normal)
  BD_INT  id_e;
  char    *reg;
  BD_INT  modo_normal;

BD_INT bd_xinde(id_e,reg,modo_normal)
  BD_INT  id_e;
  char    *reg;
  BD_INT  modo_normal;

BD_INT bd_xindx(id_e)
  BD_INT  id_e;

BD_INT bd_perte(id_e,reg)
  BD_INT  id_e;
  char    *reg;
```

Descripción:

El par de funciones bd_index() y bd_xinde() ligan o desligan respectivamente un registro *reg* al índice asociado con el enfoque *id_e* siempre y cuando su tipo sea "int", "string" o "enum". Recordemos que la herramienta traductora de archivos-esquema declara automáticamente en la estructura de los registros un campo por cada índice donde es válido ligar el registro ('campo_perte al índice') en el cual se indica si tal registro pertenece al índice, llamar estas funciones con *modo_normal* en uno garantiza la consistencia de esa información y por tanto es la forma como se deben usar, el llamado con *modo_normal* en cero lo ocupa únicamente la función **bd_copia()** (consúltese la documentación sobre esta función mas adelante) y esto es para agilizar la modificación del valor de varios campos sin actualizar con cada cambio los índices pero, al mismo tiempo, conservar la indexación original. Las diferencias en el llamado con *modo_normal* en uno o cero son:

modo_normal= 1

bd_index(): Indexa el registro y pone su campo_perte al índice en uno.

bd_xinde(): Desindexa el registro y pone su campo_perte en cero.

modo_normal= 0

bd_index(): Indexa el registro **SOLO** si su campo_perte es uno.

bd_xinde(): Desindexa el registro y **NO** altera su campo_perte.

La función **bd_xindx()** deja vacío el índice asociado con el enfoque *id_e*, todos sus nodos son liberados; ya que este procedimiento es altamente destructivo, para que trabaje es necesario que el índice en cuestión no esté asignado a ningún otro enfoque además de *id_e*, internamente esta función llama tantas veces a **bd_xinde()** como nodos tenga el índice.

Cuando el programador altera valores de campos llave sin recurrir a las funciones provistas por el manejador de la base, se corre el riesgo de corromper los índices; llamando a la función **bd_perte()** la relación entre el registro *reg* y el índice asociado con el enfoque *id_e* se normaliza. **bd_perte()** tiene otra utilidad ya que regresa cero cuando el registro *reg* está contemplado en el índice asociado pudiendo así determinarse si ya fue indexado, lo cual es especialmente útil cuando el buscar la llave de un registro aplicando la función **bd_busca()** no es un método seguro para ubicarlo tal como sucede en los índices de tipo "enum" porque se ordenan cronológicamente y no por el valor de alguna llave y en los "string" e "int" declarados para aceptar llaves repetidas.

Ejemplos:

En estas líneas de código se asignan índices a los enfoques de la FP Diálogo de Alarmas a continuación, si llega una variable alarmada se agrega a ambos índices y en las siguientes líneas, cuando el operador reconoce y cancela algún aviso de alarma, el registro adecuado se saca del índice respectivo.

```
      . . . .
      . . . .
/* Asignación de índices a enfoques */
   bd_asign(FP_DIALAR_C,IND_CRONOAL);
   bd_asign(FP_DIALAR_P,IND_PRIORAL);
      . . . .
      . . . .
/* Llega variable alarmada,
    'regis' debe apuntar a la variable alarmada */
   if (var_alarmada) {
     bd_index(FP_DIALAR_C,regis,1);
     bd_index(FP_DIALAR_P,regis,1); }
      . . . .
      . . . .
/* Si el operador cancela aviso, se desindexa,
    'regis' debe apuntar a la variable a cancelar */
   if (cancela_aviso) {
     if (pantalla == CRONOLOGICA)
       bd_xinde(FP_DIALAR_C,regis,1);
     if (pantalla == PRIORIDAD)
       bd_xinde(FP_DIALAR_P,regis,1);
      . . . .
      . . . .
```

3.2.3. Operaciones de consulta

Los registros quedan contenidos en sus respectivos arreglos de estructuras y comunmente el acceso a ellos es a través de índices ya sea de tipo preconstruido o "user" asi que el proceso de consulta de información de la base de datos sigue frecuentemente las siguientes fases:

1) Asignar un índice al enfoque con **bd_asign()**.
2) Llegar al registro adecuado ya sea con **bd_salta()** o con **bd_busca()**.
3) Consultar el o los campos.

ACCESO SECUENCIAL Y ALEATORIO A INDICES

Resumen:

Cambiar la posición dentro de un índice.

Funciones:

bd_salta(), bd_busca()

Sintaxis:

```
BD_INT bd_salta(id_e,cantidad)
  BD_INT  id_e;
  BD_INT  cantidad;

BD_INT bd_busca(id_e,llave)
  BD_INT  id_e;
  unsigned char  *llave;
```

Descripción:

Una vez asignado un registro a un índice el concepto de "registro actual" es aplicable, inicialmente el enfoque apuntará al primer registro (siempre y cuando el índice no sea tipo "user") y posteriormente si el índice asociado es tipo "int", "string" o "enum" es posible avanzar secuencialmente con la función **bd_salta()**, en la cual *cantidad* indica el número de registros a avanzar; si este parámetro es cero el puntero es colocado al principio del índice. El acceso aleatorio por medio de la función **bd_busca()** solo es posible en índices tipo "int" y "string" ya que los "enum" no tienen llave, cabe señalar que si en el índice están permitidas las llaves repetidas **bd_busca()** colocará el puntero en la primera que encuentre, para obtener los demás registros que tengan la misma llave debemos usar **bd_salta()**.

Ejemplos:

En el siguiente ejemplo, se asigna el índice por identificador a la FP Lista de Variables y se permite el paso de aquellas variables cuyo identificador empiece con "1L1"; después saltamos al principio del índice y entramos a un ciclo en el que con cada iteración visitamos las variables. Finalmente, se busca si existe el identificador "1L1830", notemos que se compara lo que regresa la función bd_busca() con el código de error BDE_BUSC (llave no encontrada).

```
     . . . .
     . . . .
/* Asignación y filtrado del índice */
  bd_asign(FP_LISVAR,IND_IDENTIF);
  bd_filtr(FP_LISVAR,"1L1");
     . . . .
/* Salta al principio del índice */
  bd_salta(FP_LISVAR,0);
/* Entra a un ciclo visitando todos los registros
   que empiezan con "1L1" */
  do {
     . . . .
     . . . .
  } while (!bd_salta(FP_LISVAR,1));
     . . . .
     . . . .
/* Busca el identificador "1L1830" */
  if (bd_busca(FP_LISVAR,"1L1830") == BDE_BUSC)
    printf("La variable 1L1830 no existe\n");
     . . . .
     . . . .
```

CONSULTA A CAMPOS

Resumen:

Obtención de valores y apuntadores a campos de un registro específico.

Funciones:

bd_ccamp(), bd_ccamp_reg(), bd_campo(), bd_campo_reg()

Sintaxis:

```
BD_INT bd_ccamp(id_e,dato,id_c)
  BD_INT  id_e;
  char    *dato;
  BD_INT  id_c;

BD_INT bd_ccamp_reg(dato,reg,id_c)
  char    *dato;
  char    *reg;
  BD_INT  id_c;

char *bd_campo(id_e,id_c)
  BD_INT  id_e;
  BD_INT  id_c;

char *bd_campo_reg(reg,id_c)
  char    *reg;
  BD_INT  id_c;
```

Descripción:

Para la consulta de campos del "registro actual" entran en acción las funciones que estamos estudiando: **bd_ccamp()** que toma el registro actual del enfoque *id_e* para copiar el valor del campo cuyo identificador es *id_c* al área apuntada por *dato* y **bd_ccamp_reg()** cuyo propósito es el mismo que la anterior pero trabaja sobre el registro *reg*, en ambas la cantidad de bytes copiada depende de la longitud del campo solicitado; también contamos con el par de funciones **bd_campo()** y **bd_campo_reg()** que en vez de copiar la información regresan un apuntador al campo, estas funciones son especialmente útiles para ciertos procedimientos del SL-GMS[19] en los que es necesario un apuntador a la información en memoria a fin de ligarla con algún objeto gráfico, es importante que para un funcionamiento adecuado el programador aplique a la función un operador "cast" de acuerdo al tipo del campo, por ejemplo:

---

[19]Vid 2.1.2, *Necesidades de los Módulos de la IHM*

```
    . . . .
    . . . .
float  *lim_alto;
    . . . .
    . . . .
lim_alto= (float *)bd_campo(FP_BARRAS,LIM_OP_ALTO);
    . . . .
    . . . .
```

Estas cuatro funciones detectan la presencia de campos tipo "reference" y automáticamente toman en cuenta el campo apuntado por ejemplo, considerando los identificadores declarados en nuestro archivo-esquema (consultar Apéndice A) para copiar el nombre de las unidades de ingeniería del registro actual en 'unid_ing' basta con pasar a la función el identificador del campo que contiene el apuntador a un registro del tipo UNIDADES:

```
    . . . .
    . . . .
char  unid_ing[9];
    . . . .
    . . . .
bd_ccamp(FP_BARRAS,unid_ing,AP_UNIDAD);
    . . . .
    . . . .
```

APUNTADORES TIPO "REFERENCE"

Resumen:

Obtener apuntadores contenidos en campos tipo "reference", calcular el apuntador "reference" correspondiente a un registro específico y obtener un registro a partir de un apuntador "reference".

Funciones:

bd_apunt(), bd_apunt_reg(), bd_regap(), bd_apreg()

Sintaxis:

```
BD_INT bd_apunt(id_e,id_c)
  BD_INT  id_e;
  BD_INT  id_c;

BD_INT bd_apunt_reg(reg,id_c)
  char    *reg;
  BD_INT  id_c;

BD_INT bd_apreg(id_e,reg)
  BD_INT id_e;
  char   *reg;

char *bd_regap(id_reg,apunt)
  BD_INT  id_reg;
  BD_INT  apunt;
```

Descripción:

Las funciones estudiadas en la sección anterior detectan automáticamente campos tipo "reference" y toman en cuenta el campo del registro apuntado pero, si es necesario obtener el **valor real** del campo, es decir el valor del apuntador "reference" entonces debemos aplicar **bd_apunt()** y **bd_apunt_reg()** las cuales difieren de **bd_campo()** y **bd_campo_reg()** en que solo aceptan campos tipo "reference" y en lugar de regresar un 'char *', regresan un BD_INT con el valor del apuntador, comparemos el ejemplo de la sección anterior con este que también copia en 'unid_ing' el nombre de las unidades de ingeniería:

```
    . . . .
    . . . .
  char    unid_ing[9];
  BD_INT  ap_reference;
    . . . .
    . . . .
  ap_reference= bd_apunt(FP_BARRAS,AP_UNIDAD);
  strcpy(unid_ing,tab_unidades[ap_reference].Nombre_unidades);
    . . . .
    . . . .
```

Para la actualización de campos tipo "reference" necesitamos primero ubicar el registro al que se desea apuntar y segundo, calcular su apuntador "reference" asociado que es precisamente lo que regresa la función **bd_apreg()**; en la cual si el parámetro *reg* es NULL, se toma el registro actual del enfoque *id_e*. La función **bd_regap** actúa de modo contrario, regresa un apuntador al registro de tipo *id_reg* que corresponde al apuntador "reference" especificado en *apunt*; en la siguiente porción de código tomamos la variable "1L1256" y hacemos que su campo AP_RETAL apunte a la variable cuyo identificador es "1L1089".

```
    . . . .
    . . . .
  BD_INT  ap_reference;
    . . . .
    . . . .
/* Asigna índice */
  bd_asign(PI_MSE,IND_IDENTIF);
    . . . .
    . . . .
/* Busca y obtiene el apuntador "reference"
   correspondiente a "1L1256" */
  bd_busca(PI_MSE,"1L1256");
  ap_reference= bd_apreg(PI_MSE,NULL);
/* Actualiza el campo AP_RETAL de la variable "1L1089" */
  bd_busca(PI_MSE,"1L1089");
  bd_reemp(PI_MSE,AP_RETAL,p_reference);
    . . . .
    . . . .
```

TIPOS

Resumen:

Obtener el tipo de índices, registros y campos.

Funciones:

bd_ind(), bd_idreg(), bd_idtip()

Sintaxis:

```
BD_INT bd_idind(id_e)
  BD_INT  id_e;

BD_INT bd_idreg(reg)
  char  *reg;

BD_INT bd_idtip(id_c)
  BD_INT  id_c;
```

Descripción:

Todas estas funciones regresan un BD_INT que contiene el valor numérico relacionado con un tipo[20], en el caso de la función **bd_idind()** esta regresa el tipo del índice asociado al enfoque *id_e*. **bd_idreg()** regresa el valor numérico del identificador del tipo de registro al que perenece *reg*.

[20]Vid sección 3.1.2, *Tipos Simples*

Ejemplos:

A continuación mostramos segmentos de código donde se imprimen los valores numéricos de los tipos:

```
     . . . .
     . . . .
  BD_INT  tipo;
     . . . .
     . . . .
/* Prueba de bd_idind() */
  tipo= bd_idind(FP_LISVAR);
  printf("FP_LISVAR tiene asignado al índice ");
  if (tipo == IND_IDENTIF)
    printf("IND_IDENTIF: %d\n",tipo);
  else
    printf("número %d\n",tipo);
     . . . .
     . . . .
/* Prueba de bd_idreg(), 'regis' apunta a un registro */
  tipo= bd_idreg(regis);
  printf("El tipo del registro 'regis' es ");
  if (tipo == ANALOGICAS)
    printf("ANALOGICAS: %d\n",tipo);
  else
    printf("%d\n",tipo);
     . . . .
     . . . .
/* Prueba de bd_idtip() */
  tipo= bd_idtip(NOMBRE_ESTADO);
  printf("El tipo del campo NOMBRE_ESTADO es ");
  if (tipo == BD_STRING)
    printf("BD_STRING: %d\n",tipo);
  else
    printf("%d\n",tipo);
     . . . .
     . . . .
```

3.2.4. Operaciones de actualización

Las fases descritas para el proceso de consulta también son válidas aqui, el conjunto de funciones para actualización es pequeño pero permite aplicar tanto el concepto de "registro actual" en las modificaciones como trabajar sobre registros específicos.

ACTUALIZACION DE CAMPOS

Resumen:

Actualizar el valor de un campo y copiar campos entre registros.

Funciones:

bd_reemp(), bd_reemp_reg(), bd_copia()

Sintaxis:

```
BD_INT bd_reemp(id_e,id_c,dato)
  BD_INT  id_e;
  BD_INT  id_c;
  char    *dato;

BD_INT bd_reemp_reg(reg,id_c,dato)
  char    *reg;
  BD_INT  id_c;
  char    *dato;

BD_INT bd_copia(id_e,reg_destino,reg_fuente)
  char  *reg_destino,*reg_fuente;
```

Descripción:

Las funciones **bd_reemp()** y **bd_reemp_reg()** copian respectivamente al registro actual del enfoque *id_e* y al registro *reg* tantos bytes como tenga el campo cuyo identificador es *id_c* a partir del área apuntada por *dato*; en el caso de campos tipo "reference" no es válido intentar cambiar el valor del campo apuntado, por ejemplo si quisiéramos actualizar las unidades de una variable a "°C" el siguiente código NO es válido:

```
   . . . .
   . . . .
bd_reemp(FP_DIASER,AP_UNIDAD,"°C")
   . . . .
   . . . .
```

El procedimiento adecuado es buscar en los registros de tipo UNIDADES aquel que contenga la cadena "°C", obtener su apuntador "reference" ya sea con la función **bd_apreg()** o directamente por su posición en el arreglo y entonces usar **bd_reemp()** para actualizar AP_UNIDAD con dicho apuntador.

```
    . . . .
    . . . .
BD_INT  i;
    . . . .
    . . . .
for (i= 0;
     strncmp(tab_unidades[i],"°C",2);
     i++);
bd_reemp(FP_DIASER,AP_UNIDAD,&i)
    . . . .
    . . . .
```

En algunos casos es necesario pasar todos los campos de una variable a otra; gracias a la forma de ligado entre un registro y sus campos fue posible la implementación de la función **bd_copia()** la cual a través del enfoque *id_e* copia campos entre dos registros: de *reg_fuente* a *reg_destino*. Una característica sobresaliente de la función es que los registros pueden ser de diferentes tipos y solo toma en cuenta los campos comunes a ambos. El mantenimiento de los índices está garantizado y se hace de modo automático, es decir, mediante llamadas internas a **bd_xinde()** y **bd_index()** de acuerdo al siguiente algoritmo.

Desindexar:

El registro es desligado, el 'campo_perte' correspondiente a cada índice se mantiene en uno.

Copiar campos:

No hay actualización de los índices.

Indexar registro:

El registro es colocado en su nueva posición únicamente dentro de los índices donde originalmente estaba.

** Las llamadas a **bd_xinde()** y **bd_index()** son con modo_normal en cero (consulte estas funciones).

ALTA Y BAJA DE REGISTROS

Resumen:

Agregar y eliminar registros.

Funciones:

bd_agreg(), bd_borra()

Sintaxis:

```
char *bd_agreg(id_e,id_reg)
  BD_INT  id_e;
  BD_INT  id_reg;

BD_INT bd_borra(id_e,reg)
  BD_INT  id_e;
  char    *reg;
```

Descripción:

Recordemos que existe un arreglo de estructuras para contener los elementos de cada tipo de registro, esto limita la cantidad máxima de registros que pueden existir; sin embargo el manejador de la base de datos distingue entre registros válidos (que contienen información) e inválidos (disponibles) por dos características: primero, todo registro disponible está ligado en un índice "enum" (declarado automáticamente por la herramienta GENERA) asociado con su tipo de registro y cuyo identificador es igual pero con el prefijo 'LIB_' y segundo, el tipo de un registro inválido es igual en valor al de los válidos pero de signo negativo.

Con este método debemos dejar cierta holgura previendo altas de registros; la función **bd_agreg()** a través del enfoque *id_e* libera un registro de tipo *id_reg* de su índice de registros libres, pero no lo agrega a ningúno otro; trabajando en modo inverso **bd_borra()** liga el registro *reg* (si *reg* es NULL, se toma el registro actual del enfoque) junto con los demás disponibles previa desindexación de cualquier otro índice, es decir que la llamada a **bd_borra()** efectúa un mantenimiento automático de índices.

Ejemplos:

En esta porción de código buscamos y eliminamos la variable "1V4639" y agregamos la "4V5702" dándola de alta en los índices por identificador, por sistema y subsistema y por dirección hardware:

```
    . . . .
    . . . .
  char  *regis;
  char    campo_char;
  BD_INT  campo_ref;
    . . . .
    . . . .
/* Asignación de índice */
  bd_asign(FP_DIASER,IND_IDENTIF);
    . . . .
    . . . .
/* Busca y elimina la variable "1V4639" */
  if (!bd_busca(FP_DIASER,"1V4639"))
    bd_borra(FP_DIASER,NULL);
    . . . .
    . . . .
/* Agrega un registro tipo Analógicas */
  regis= bd_agreg(FP_DIASER,ANALOGICAS);
/* Asigna ciertos valores a algunos de sus campos */
  campo_char= 3;
  bd_reemp_reg(regis,CANASTA,campo_char);
  campo_char= 34;
  bd_reemp_reg(regis,TARJETA,campo_char);
  campo_char= 4;
  bd_reemp_reg(regis,BIT,   campo_char);
  bd_reemp_reg(regis,IDENTIFICADOR,"4V5702");
  campo_ref= 12;
  bd_reemp_reg(regis,AP_SISTEMA,campo_ref);
  campo_ref= 2;
  bd_reemp_reg(regis,AP_SUBSISTEMA,campo_ref);
/* Indexa el registro */
  bd_asign(FP_DIASER,IND_DIRHW);
  bd_index(FP_DIASER,regis,1);
  bd_asign(FP_DIASER,IND_SISYSUB);
  bd_index(FP_DIASER,regis,1);
  bd_asign(FP_DIASER,IND_IDENTIF);
  bd_index(FP_DIASER,regis,1);
    . . . .
    . . . .
```

3.3. EVALUACION GENERAL DE DESEMPEÑO

Como el sistema final para la Central de Ciclo Combinado de Gómez Palacio debe correr bajo el ambiente VAXELN, para la "entonación" del mismo debemos conocer acerca de cada proceso parámetros relacionados con cantidad de memoria requerida y velocidad de respuesta (tiempo que mantienen ocupado al procesador, etc.); con esta información se asigna a cada uno el número de páginas adecuado y su prioridad.

3.3.1. Almacenamiento

Las estructuras que sustentan la base de datos son de tres tipos de acuerdo a su función; unas contienen la información en si (los arreglos de registros), otras describen sus características (descriptores) y otras la mantienen organizada (índices). El programa "GENERA" para escribir las declaraciones de tales estructuras en el encabezado "BD_DESC.H" lee la información del archivo-esquema; en algunas estructuras (en los descriptores) solo varía la cantidad declarada[21], otras cambian no solo en número sino además en sus elementos (arreglos de registros) y para los índices, sin importar cuantos y de que tipo sean "GENERA" siempre declara una sola estructura básica (el nodo) ya que estos son inicializados (por lo común al momento de la carga de la base) y mantenidos dinámicamente.

Para calcular el tamaño de las estructuras se hizo un pequeño sistema en el que la única actividad (previa inicialización de la base) consiste en aplicar el operador "sizeof" a cada una de ellas e indicar en pantalla su respectiva longitud en bytes; la cantidad de registros (dejando suficientes entradas para nuevos) fue tomada de 2.3 *Estadísticas de la Información a Manejarse*; la tabla resume los tamaños de registros declarados de acuerdo al archivo-esquema propuesto (consultar el Apéndice A):

[21] Vid *Estructuras "Predefinidas" y "Definibles"* en la sección 3.1.2

NOMBRE	TAMAÑO [Bytes]	CANTIDAD	PARCIAL [KBytes]
Reg_logicas	71	1450	102.95
Reg_analogicas	91	1200	109.20
Reg_logicas_calc	59	65	3.84
Reg_analogicas_calc	89	240	21.36
Reg_constantes	52	700	36.40
Reg_diagnosticos	53	600	31.80
Reg_configuraciones	50	215	10.75
Reg_unidades	9	50	0.45
Reg_estados	9	50	0.45
Reg_acciones	10	15	0.15
Reg_sistemas	34	20	0.68
Reg_subsistemas	38	90	3.42
Reg_linearizaciones	14	15	0.21
Reg_programas	36	20	0.72
Reg_integraciones	32	100	3.20
Reg_canastas	2	30	0.06
T O T A L [KBytes]:- - - - - - - -			325.64

A continuación indicamos tamaño y cantidad de cada una de las estructuras descriptoras:

NOMBRE	TAMAÑO [Bytes]	CANTIDAD	PARCIAL [Bytes]
bd_desc_campo	9	84	756
bd_desc_registro	281	17	4777
bd_desc_indice	103	22	2266
bd_desc_enfoque	33	23	759
bd_struct_prot	6+128	38	5092
bd_nodo	9	0	0
T O T A L [Bytes]:- - - - - - - - -			13650

Acerca de los índices, aunque en el archivo de encabezado "BD_DESC.H" solo definimos la estructura "bd_nodo" y no se declara de hecho ningún "nodo", la creación de los mismos es por solicitud de memoria dinámica al momento de la ejecución, siendo por tanto de especial interés e importancia calcular la cantidad máxima de memoria requerida para evitar fallas durante la indexación; para los índices este parámetro se calcula directamente de la cantidad máxima de nodos que se espera puedan contener pero, como esto es variable debido al tipo de índice y a su distribución de llaves, los analizaremos uno a uno:

Indice IND_CRONOAL: Lo ocupa la Función de Presentación de Alarmas para el desplegado cronológico y como máximo podrá contener 100 elementos, al ser un índice de tipo "enum", la cantidad de nodos por elemento es constante e igual a uno; asi que por parte de este índice serán solicitados al sistema hasta 100 nodos.

Indices LIB_*: Se trata de 16 índices (uno para cada tipo de registro) cuyo propósito es mantener ligados los registros disponibles o vacios, suponiendo que se cargara la base completamente en blanco (lo que no es usual), la cantidad de nodos que estos índices solicitarían equivale al total de registros; y resulta de sumar la columna marcada como "CANTIDAD" de la tabla de tamaños de registros: 1450 + 1200 + ... + 30= 4860 registros.

Indice IND_PRIORAL: Este índice lo necesita Alarmas para su desplegado de variables alarmadas por prioridad, recordando la estructura de los indices tipo "int" descrita en 3.1.2 *Estructuras de Datos* y considerando que el rango del campo PRIORIDAD es[22] de 1 a 9, resulta claro que los primeros tres niveles del índice solo contendrán un nodo con valor cero, mientras que en el último nivel estarán acumuladas todas las llaves, siendo asi, la cantidad máxima de nodos para este índice es de 103.

Indice IND_IDENTIF: Sirve para mantener ordenados por identificador los registros de variables y accesarlos de esta forma en operaciones no-críticas, es decir, sin relación directa con procesos de tiempo real. Dado que la distribución de llaves (valores del campo IDENTIFICADOR) no es homogénea, para obtener un valor realista se hizo un estudio sobre los identificadores de una base de datos en PC no depurada que contenía 5456 variables. El primer paso para determinar teóricamente la cantidad de nodos consistió en encontrar cuantos valores de llave diferentes se podían encontrar en cada nivel, por ejemplo, si tomamos los 6 caracteres del campo, como no hay identificadores repetidos tendremos precisamente 5456 llaves distintas, en el otro extremo, considerando solo un caracter del campo encontramos que este podía tomar uno de seis valores: { 0, 1, 2, 5, 7, 9 }, es decir, que en el primer nivel tendremos seis nodos.

NO_CARACTERES	NO_LLAVES_DIFERENTES
1	6
2	37
3	93
4	253
5	911
6	5456

Ahora bien, atendiendo a la propiedad de los índices tipo "string" donde para cada caracter hay un nivel, resulta claro que la cantidad de nodos por nivel corresponde directamente al número de llaves diferentes y el total de nodos es la suma de 6 + 37 + ... + 5456= 6756 nodos. Actualmente la base consta de 4024 registros tipo variable y sigue en proceso de depuración, sin embargo tomaremos como válido el cálculo anterior para dejar cierta libertad.

---

[22]Vid sección 2.2.2 *Atributo PRIOPRIDAD*

Indice IND_SISYSUB: Este índice tiene como parte final los seis caracteres de IDENTIFICADOR y en los dos primeros niveles contiene el número de sistema y subsistema sin embargo, como en el identificador[23] están de alguna manera implícitos ambos, podemos predecir que a pesar de agregar dos niveles, la cantidad de nodos no se incrementará en mucho pero, nos aseguraremos repitiendo el procedimiento anterior.

NO_CARACTERES	NO_LLAVES_DIFERENTES
1	15
2	121
3	151
4	297
5	406
6	607
7	1595
8	5456

El total de nodos necesarios para construir IND_SISYSUB en esta base no depurada es 15 + 121 + ... + 5456= 8648, (8648/6756 - 1) = 28% más que en el índice IND_IDENTIF.

Indice IND_DIRHW: Es un índice tipo "user" y por tanto no se construyó a base de elementos-nodo sino a través de un arreglo de apuntadores[24] de tres dimensiones correspondiendo cada una a CANASTA, RANURA y SENAL con un total de 23 * 16 * 64= 23552 elementos cada uno de cuatro bytes.

En la tabla se concentran los datos relacionados con índices indicando el nombre de cada uno, la cantidad máxima de nodos o elementos solicitada y en "PARCIAL" su consumo de memoria calculado a partir de que cada nodo requiere 19 bytes; nueve para su estructura "bd_nodo" y diez para el encabezado o "header" que el sistema agrega a fin de controlar cualquier bloque solicitado dinámicamente y cada apuntador tiene 4 bytes de longitud.

INDICE	NO_NODOS/ELEMENTOS	PARCIAL
LIB_*	4860	92.34
IND_CRONOAL	100	1.90
IND_PRIORAL	103	1.96
IND_IDENTIF	6756	128.36
IND_SISYSUB	8648	164.31
IND_DIRHW	23552	94.21
T O T A L (KBytes): - - - - -		483.08

[23] Vid sección 2.2.2 *Atributo IDENTIFICADOR*

[24] Vid sección 3.1.2, *Esquema Propuesto*

Finalmente obtenemos el consumo de memoria máximo sumando los totales debido a registros, descriptores e índices:

ELEMENTO	CONSUMO (KBytes)
Registros	325.64
Descriptores	13.65
Indices	483.08
Total:	822.37

3.3.2. Velocidad de acceso

La alta velocidad de acceso a la información, necesaria en procesos directamente relacionados con operaciones en tiempo real se consigue en esta aplicación por medio del índice de usuario IND_DIRHW el cual, proporciona un método que sin duda es el mas veloz ya que no solo la estructura del índice y la información residen en memoria RAM sino además, la recuperación de un registro implica únicamente obtener el apuntador guardado como un elemento de arreglo. **Usar IND_DIRHW es casi tan rápido como accesar al valor de una variable que el programador hubiera declarado en su módulo.** Para comprobarlo y tener una referencia temporal, se midió el tiempo que tardó un ciclo "for" de 10 millones de iteraciones recuperar en cada una dos elementos del arreglo asociado con IND_DIRHW: tan solo 38 segundos, es decir, que por segundo fueron accesados más de 525,000 elementos.

En algunos procesos, especialmente en las Funciones de Presentación la secuencia de recuperación (el orden o indexado) es más importante que la velocidad, para ello usamos los índices tipo "enum", "int" y "string"; en los índices con llave ("int" y "string") la actividad de recuperación de elementos denota tres pasos básicos:

1. Recuperar el valor asociado con el nodo.

2. Comparar contra el caracter de la llave adecuado.

3. Saltar al siguiente nodo del mismo nivel si la comparación resultó negativa o al siguiente nivel si resultó positiva.

Esto indica que la visita a un nodo es de por si tres veces más lenta que el acceso a un elemento de arreglo, de ahi que sea punto central la obtención y minimización del número de comparaciones promedio durante la búsqueda de llaves específicas: *Sea el índice multinivel 'IND' que contiene 'K' llaves organizadas en un sistema de 'x' niveles con un promedio de 'y' nodos en cada nivel,*

entonces, la cantidad promedio 'N' de comparaciones necesarias en un mismo nivel para dar con un valor determinado viene dada por:

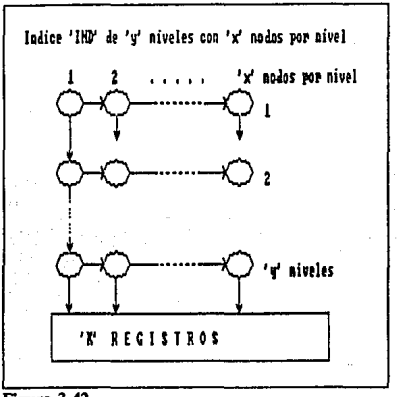


Figura 3.42
Ejemplo: Indice multinivel 'IND'

$$N = (y + 1) / 2 \qquad \ldots\ldots(1)$$

Y dado que tenemos 'x' niveles, entonces, el promedio total 'P' de comparaciones es:

$$P = x * N = x * (y + 1) / 2 \qquad \ldots\ldots(2)$$

Analizando la ecuación anterior vemos que se pueden obtener diferentes valores de 'P' con distintas distribuciones 'x', 'y' que den la misma cantidad 'K' de llaves:

$$K = y^x \qquad \ldots\ldots(3)$$

Obviamente en todo caso el propósito es minimizar 'P' y asi ganar en velocidad, para obtener la pareja 'x','y' ideal conjuntamos (3) y (2) en (5) y aplicamos la teoría de máximos y mínimos al resultado:

Despejando 'y' de (3):

$$y = K^{(1 / x)} \qquad \ldots\ldots(4)$$

Sustituyendo (4) en (2):

$$P = x/2 * (K^{(1/x)} + 1) \qquad ....(5)$$

Derivando (5):

$$P' = 1/2 * (K^{(1/x)} + 1) + x/2 * (K^{(1/x)})'$$

$$P' = 1/2 * (K^{(1/x)} + 1) - K^{(1/x)} * Ln(K) / (2*x)$$

Haciendo P'= 0 para encontrar el punto mínimo:

$$x*(K^{(1/x)} + 1) = K^{(1/x)} * Ln(K)$$

Igualando a cero:

$$0 = K^{(1/x)} * (Ln(K) - x) - x \qquad ....(6)$$

Finalmente a la ecuación (6) podemos aplicar el método iterativo de Newton-Raphson para encontrar la 'x' ideal y usando (4) calcular su correspondiente 'y'; para ello definimos la función f(x) y su derivada f'(x) tales que:

$$x_{(i+1)} = x_i - f(x) / f'(x) \qquad ....(7)$$

Donde de (6):

$$f(x) = K^{(1/x)} * (Ln(K) - x) - x$$

Y su derivada f'(x):

$$f'(x) = \frac{-K^{(1/x)} * Ln(K)}{x^2} * (Ln(K) - x) - K^{(1/x)} - 1$$

Ahora vamos a un ejemplo práctico, tomando en cuenta la base original no depurada en IND_IDENTIF se indexan 5456 registros[25] es decir que el universo 'K' es de 5456 llaves, la aplicación iterativa de (7) lleva a un valor de 6.73 para 'x' y aplicando (4) $y = K^{(1/x)} = 3.59$; para esta combinación usando (2) el promedio de comparaciones 'P' es 15.44, verificaremos numéricamente que esa pareja es la mejor haciendo una tabla con algunos valores posibles de 'x', 'y' para los que se cumple K= 5456:

[25] Vid sección 3.3.1, *Almacenamiento*

x	y	P
1	5456.00	2728.50
2	73.86	74.86
3	17.60	27.91
4	8.59	19.19
5	5.59	16.47
6	4.19	15.57
6.7	3.61	15.45
7	3.42	15.46
8	2.93	15.73
9	2.60	16.21
10	2.36	16.82

Afortunadamente, la longitud de llave para el índice IND_IDENTIF es de 6 caracteres asi que el acceso se hará prácticamente con la mayor agilidad posible (6.73 sería el ideal para K= 5456 pero el numero de niveles 'x' debe ser entero), dentro de las limitaciones impuestas por la propia estructura. Abordando aspectos comparativos respecto a la indexación por arreglo de apuntadores, consideremos la pareja x= 6, y= 4.19, si tomamos en cuenta que es tres veces más costosa la visita a un nodo, es de esperarse una disminución en velocidad del orden de tres veces 'P', es decir, 3 * 15.57= 46.71 al menos.

El análisis expuesto adolece de un defecto que a primera vista resulta grave y es que **presupone igual distribución de nodos en todos los niveles** pero, vamos a demostrar como este supuesto no nos aleja mucho de la realidad si la distribución es más o menos uniforme; volvamos a la tabla que relaciona 'número de caracteres' con 'número de llaves diferentes' del índice IND_IDENTIF de la sección 3.3.1, de un número de niveles x= 6 y suponiendo una distribución uniforme de y= 4.19 por nivel encontramos P= 15.57; si ahora calculamos la **distribución real** por nivel dividiendo sucesivamente los renglones de 'número de llaves diferentes' tenemos que:

Nivel	NLNA	NLEN	DPR
1	-	6	6 / 1 = 6.000
2	6	37	37 / 6 = 6.166
3	37	93	93 / 37 = 2.514
4	93	253	253 / 93 = 2.720
5	253	911	911 / 253 = 3.601
6	911	5456	5456 / 911 = 5.989

*NLNA: Número de Llaves en el Nivel Anterior
*NLEN: Número de Llaves en Este Nivel
*DPR: Distribución Promedio Real

Resulta claro que en el peor de los casos durante la búsqueda de un nodo en un nivel tendríamos que recorrerlo en su totalidad, en esas condiciones se visitarían 6 + 6.166 + ... + 5.989= 27 nodos, por otro lado, en el mejor de los casos tendríamos que visitar al menos un nodo por cada nivel es decir, seis nodos; promediando ambos extremos resulta:

```
P= (6 + 27) / 2 = 16.5
```

Valor que es muy aproximado al calculado con el supuesto de distribución uniforme y que nos da una disminución en velocidad de respuesta de (3 * P) 49.5 Con objeto de comprobar prácticamente las apreciaciones anteriores, se dispuso la misma prueba de velocidad aplicada a recuperación de elementos de arreglos pero para elementos del índice IND_IDENTIF, llamando en cada iteración del ciclo "for" dos veces a la función "bd_busca" con diferentes parámetros, el resultado fue de 100,000 iteraciones en 19 segundos; más de 10,500 elementos recuperados por segundo. Comparando 10,500 contra 525,000 obtenemos una relación en tiempo entre ambas pruebas de 1:50, es decir que resulta cincuenta veces mas costoso, hablando en tiempo, accesar a la base de datos por el índice de identificadores que por la dirección hardware mediante el arreglo de apuntadores, claro que intentar este último método para ordenar los registros de variables por sus identificadores no es viable pues consumiría varios megabytes de memoria RAM.

En cuanto a los índices tipo "int", recordemos que su llave de acceso proviene de enteros de 2 bytes es decir que se pueden tener hasta 65536 diferentes, sin embargo al menos en nuestro sistema no usamos índices de este tipo con más de 100 llaves y podemos esperar que aun cuando se necesitaran otros, la cantidad de llaves no excederá la cantidad total de registros de variables que en la base depurada es de 4470[26], veamos los siguientes valores de 'P' para algunas 'K'

K	P
100	3.6
4096	6.5
65536	8.7

Decidimos configurar la cantidad de niveles en cuatro por un lado para cubrir el rango de 100 a 4096 aceptablemente, en especial el punto K = 100 y además, porque el procedimiento de conversión de 2 bytes a 4 es un algoritmo pequeño que requiere poco tiempo de procesador lo que lo hace ideal para ser llamado en cada búsqueda.

[26]Vid sección 3.3.1 *Almacenamiento*

REFERENCIAS

Andrew W. Kramer
Power Plant Primer
Technical Publishing Company
Barrington , Illinois
Enero, 1979
16p

H. L. Harkins
Combined Cicles: Today and Tomorrow
Energy Today
Julio, 1980

Prontuario de Datos Técnicos de
Central de Ciclo Combinado de Gómez Palacio
CFE
Octubre, 1989
126p

Ricardo Ortega
Planeación de la Interfaz Hombre-Máquina
del SCD para la CCC Gómez Palacio, Durango
IIE
Agosto 1991

Sergio Alvarez
Relación de Especificaciones Funcionales de
las Funciones de Presentación de
Interfaz Hombre-Máuina del SCD para
la CCC de DosBocas, Veracruz
IIE
Septiembre, 1990

Sergio Alvarez
Descripción de Campos de la Base de Datos
Definitiva en PC para el SCD para la CCC
de DosBocas, Veracruz
IIE
Septiembre, 1990

VAXELN Vol. 3:
Runtime Facilities Guide
Digital Equipment Corporation
Maynard, Massachusetts
Marzo, 1990

VAXELN Vol. 7:
C Runtime Library Reference Manual
Digital Equipment Corporation
Maynard, Massachusetts
Marzo, 1990

CONCLUSIONES

A través de este trabajo hemos documentado la investigación previa y el planteamiento de una respuesta a un problema muy específico: el diseño de la base de datos para la IHM del proyecto SCD para la CCC de Gómez Palacio, Durango. Sin embargo, a pesar de los requerimientos muy particulares del caso, intentamos dar una visión global apoyándonos en un amplio estudio teórico sobre los conceptos generales relacionados con bases de datos.

Los sistemas que incluyen elementos en tiempo real tan característicos en procesos de control industrial están ampliando su campo de aplicación a toda una diversidad de áreas; cada vez son mas apreciados los beneficios que proporciona la relación directa del usuario con información fresca. Los sistemas de cómputo ya no son la caja negra que alimentada por lotes de entradas produce listados, ahora realizamos interacción directa por medio de elaboradas IHM, las peticiones de los usuarios de un sistema ya no están programadas, ahora varian según las necesidades del momento; el diseño de un almacén de información (base de datos) organizado lógicamente del cual se extraen y actualizan datos y que presta sus servicios a un sistema de control donde una falla puede costar incluso vidas humanas, obliga la revisión teórica de cada aspecto.

Dentro de los diferentes elementos de hardware, la elección adecuada del medio de almacenamiento denota una planeación integral y prepara el escenario para llevar a buen término la implantación de una BD, ya desde este punto el analista debe responder a toda una serie de cuestiones técnicas que se desprenden de su análisis sobre los requerimientos y la naturaleza de la información a manejar tales como: capacidad, disponibilidad y velocidad de respuesta que se espera tenga el sistema.

La base de datos, siendo el centro de recolección y consulta de información de los diferentes módulos de un sistema debe estar ampliamente documentada con un diccionario de datos (DD) que establezca los términos con que usuarios, programadores, analistas y demás personas relacionadas con el proyecto deben comunicarse, en el DD encontraremos también la descripción de cada atributo y entidad lo que nos dará una vision particular y detallada sobre los elementos del sistema; en contraste, el modelo conceptual de la BD, cristalización de los requerimientos y primer paso en firme hacia un prototipo, funge como descripción global no solo de los datos sino que es el reflejo conceptual del sistema mismo.

Atraidos por la singular claridad y cualidades que presta la diagramación mediante hipergráficas en el modelado de sistemas de tiempo real, aplicamos varios de sus conceptos en el diagramado del modelo conceptual de la BD, aunque la idea no es del todo nueva, se extendió su alcance con la representación de dependencias funcionales obteniendo una metodología en mucho superior a los típicos diagramas E-R. De igual forma la notación matemática tradicional para expresar dependencias funcionales se amplió diferenciando las totales de las que no lo son; con esta simbología "extendida" intentamos presentar un humilde aporte a los métodos de diagramación de modelos conceptuales.

Aunque el origen del proceso de normalización está fuertemente ligado al desarrollo del enfoque relacional al grado que algunos autores proponen la normalización sobre un modelo del tipo relacional para su posterior mapeo a otros enfoques de ser necesario, nosotros aplicamos la normalización sobre el modelo conceptual indicando el procedimiento para desarrollarla gráfica y matemáticamente resultando de esto un modelo conceptual de mayor validez.

En el camino hacia la implantación física, atendiendo al modelo lógico que resulta del mapeo del modelo conceptual a algún enfoque y al medio de almacenamiento que contendrá cada elemento de la BD, viene la elección de la estructura de datos o tipo de archivo con que se dará sentido lógico a la disposición física de la información. Hemos demostrado, en el estudio hecho sobre las estructuras predefinidas de índices, como aun contando con la posibilidad de almacenamiento en memoria RAM la caracterización de una estructura de datos con los parámetros correctos implica diversos cálculos matemáticos, esto es extensivo y aun más riguroso si esperamos un buen desempeño por parte de medios de almacenamiento que presenten partes mecánicas.

Frecuentemente los productos de software a diferencia de los de hardware se construyen casi desde sus cimientos; pensemos en el diseño de un equipo electrónico digital, en nuestro trabajo planearemos las interconexiones entre diferentes circuitos consultando catálogos donde se describen las características operativas de cada uno. Hace años los diseñadores de software no contaban con nada semejante; para obtener un producto integral el camino era la fragmentación del problema en módulos para su posterior codificación, cuando era posible copiar o adecuar algunas rutinas de proyectos anteriores ya podían considerarse afortunados.

Ahora, con la especialización y estandarización de procedimientos el diseñador de software puede considerar la posibilidad de ligar su propio código con bibliotecas de editores gráficos, manejadores de base de datos, administradores de ventanas, etc. Tomando ese enfoque planeamos la definición y manejo de la base, haciendo de estos módulos elementos configurables e independientes, el primero por medio de la herramienta traductora de esquemas y el segundo con la biblioteca de funciones del manejador de la BD. Ciertamente, el desarrollo automatizado de un producto de software cualquiera a partir únicamente de su especificación de requerimientos está muy lejos, siquiera pensarlo resulta impráctico a menos que se contara con una interfaz en lenguaje natural pero, por el momento, las técnicas de cuarta generación ofrecen respuestas prometedoras a grupos de trabajo que se enfrentan a proyectos de dimensiones considerables.

La *"limpieza"* en la interacción entre un módulo reutilizable y el código escrito especificamente para una aplicación se manifiesta por la facilidad con que los problemas son

conceptualizados en los términos que impone la sintaxis de comandos del propio módulo; de poca utilidad sería, por ejemplo, un administrador de ventanas en el que para desactivar una ventana nos viéramos obligados a navegar por las estructuras internas propias del manejador. Creemos que la información contenida en cualquier base de datos declarada a través del lenguaje esquema propuesto puede manejarse eficazmente, los ejemplos, abundan en la descripción de las rutinas del manejador. Tan solo queda esperar que el empleo de estos módulos llegue a otros proyectos una vez asentado el precedente en el proyecto del *"Sistema de Control Distribuido para la Central de Ciclo Combinado de Gómez Palacio, Durango"*.

APENDICE A

ARCHIVO-ESQUEMA PROPUESTO

```
;ARCHIVO ESQUEMA PARA LA BASE DE DATOS DE GOMEZ-PALACIO, DURANGO:

; En los nombres de identificador se usaron las siguientes abreviaciones:
;    INH.........  INHIBIDO(A)
;    VAR.........  VARIABLE
;    LIM.........  LIMITE
;    PORC........  PORCENTAJE
;    AP..........  APUNTADOR
;    RETAL.......  RETROALIMENTACION
;    OP..........  OPERACION
;    GRAF........  GRAFICACION
;    INT.........  (DE / PARA) INTEGRACION

CAMPOS(
; Identificador          Nombre                 Tipo       Longitud  Reg_ref           Campo_ref
   PUNTO_BLOQUEADO        Punto_bloqueado        bit          1      -                 -
   SCAN_INH               Scan_inhibido          bit          1      -                 -
   ALARMA_INH             Alarma_inhibida        bit          1      -                 -
   VAR_ALARMADA           Bandera_alarma         bit          1      -                 -
   EVENTO_INH             Evento_inhibido        bit          1      -                 -
   ESTADO_FUNCIONAL       Estado_funcional       bit          1      -                 -
   VALOR_DADO             Valor_operador         bit          1      -                 -
   CALIDAD_DATO           Calidad_dato           bit          1      -                 -
   BANDA_MUERTA           Banda_muerta           bit          1      -                 -
   BANDA_MUERTA_PUESTA    Banda_puesta           bit          1      -                 -
   ALARMA_RETORNO         Alarma_retorno         bit          1      -                 -
   LIM_DINAMICO           Limite_dinamico        bit          1      -                 -
   LIM_DINAMICO_PUESTO    Limite_puesto          bit          1      -                 -
   ALARMA_NO_RECONOCIDA   Alarma_nreconocida     bit          1      -                 -
   ACCION_INH             Control_inhibido       bit          1      -                 -
   ACCION_EN_PROGRESO     Control_progreso       bit          1      -                 -
   NO_RETAL               No_retro               bit          1      -                 -
   ESTADO_NORMAL          Estado_normal          bit          1      -                 -
   ALARMA_0A1             Alarma_01              bit          1      -                 -
   ALARMA_1A0             Alarma_10              bit          1      -                 -
   IDENTIFICADOR          Identificador          string       7      -                 -
   DESCRIPCION            Descripcion            string      31      -                 -
   PRIORIDAD              Prioridad              char         -      -                 -
   PORC_BANDA_MUERTA      Porc_banda_muerta      char         -      -                 -
   PORC_LIM_DINAMICO      Porc_limite_dinamico   char         -      -                 -
   LIM_CRITICO_ALTO       Limite_crit_alto       ushort       -      -                 -
   LIM_CRITICO_BAJO       Limite_crit_bajo       ushort       -      -                 -
   LIM_PRECRITICO_ALTO    Limite_prec_alto       ushort       -      -                 -
   LIM_PRECRITICO_BAJO    Limite_prec_bajo       ushort       -      -                 -
   AP_LINEARIZACION       Ap_linearizacion       reference    -      LINEARIZACIONES   FUNCION_LINEARIZACION
   VALOR_ANALOGICO        Valor_variable         ushort       -      -                 -
   AP_UNIDAD              Unidades               reference    -      UNIDADES          NOMBRE_UNIDAD
   ZUMBA_ALARMA           Zumba_alarma           ushort       -      -                 -
   IMPRIME_ALARMA         Imprime_alarma         ushort       -      -                 -
   AP_ESTADO_NORMAL       Ap_estado_normal       reference    -      ESTADOS           NOMBRE_ESTADO
   AP_ESTADO_ANORMAL      Ap_estado_anormal      reference    -      ESTADOS           NOMBRE_ESTADO
   AP_ACCION              Accion_control         reference    -      ACCIONES          NOMBRE_ACCION
   DURACION_ACCION        Timer_control          int          -      -                 -
   RETRASO_RETAL          Timer_retro_control    int          -      -                 -
   AP_RETAL               Ap_retro               reference    -      LOGICAS           VALOR_LOGICO
   VALOR_RETAL_ESPERADO   Estado_control         bit          1      -                 -
   VALOR_LOGICO           Valor_logico           bit          1      -                 -
   AP_INTEGRACION         Ap_est_integracion     reference    -      INTEGRACIONES     -
   LIM_OP_ALTO            Limite_oper_alto       float        -      -                 -
   LIM_OP_BAJO            Limite_oper_bajo       float        -      -                 -
   RANGO_GRAF_INFERIOR    Rango_gra_papel_inf    float        -      -                 -
   RANGO_GRAF_SUPERIOR    Rango_gra_papel_sup    float        -      -                 -
   ALARMAR_DIAGRAMA       Alarmar_diagrama       bit          1      -                 -
   GANANCIA_SENAL         Ganancia               char         -      -                 -
   AP_SISTEMA             Numero_sistema         reference    -      SISTEMAS          NOMBRE_SISTEMA
   AP_SUBSISTEMA          Numero_subsistema      reference    -      SUBSISTEMAS       NOMBRE_SUBSISTEMA
   COLOR_VAR              Color                  char         -      -                 -
   TENDENCIA_VAR          Tendencia              char         -      -                 -
   CANASTA                Canasta                char         -      -                 -
   TARJETA                Tarjeta                char         -      -                 -
```

```
  BIT                   Bit                   char       -    -          -
  NOMBRE_UNIDAD         Nombre_unidades       string     7    -          -
  NOMBRE_ESTADO         Nombre_estado         string     7    -          -
  NOMBRE_ACCION         Nombre_accion         string     8    -          -
  NOMBRE_SISTEMA        Nombre_sistema        string    32    -          -
  NOMBRE_SUBSISTEMA     Nombre_subsistema     string    32    -          -
  SISTEMAS_VALIDOS      Sistemas_validos      uint       -    -          -
  NOMBRE_LINEARIZACION  Nombre_linearizacion  string     8    -          -
  FUNCION_LINEARIZACION funcion_linearizacion pointer    -    -          -
  NOMBRE_PROGRAMA       Nombre_programa       string    32    -          -
  NUMERO_PROGRAMA       Numero_programa       ushort     -    -          -
  FACTOR_INT            Factor_int            float      -    -          -
  TIEMPO_MUESTREO_INT   Tiempo_muestreo_int   int        -    -          -
  TIEMPO_ACTUAL_INT     Tiempo_actual_int     int        -    -          -
  TIEMPO_MAXIMO_INT     Tiempo_maximo_int     int        -    -          -
  CUENTA_CORRIENTE_INT  Cuenta_corriente_int  float      -    -          -
  CUENTA_MAXIMA_INT     Cuenta_maxima_int     float      -    -          -
  CUENTA_ULTIMA_INT     Cuenta_ultima_int     float      -    -          -
  AP_VAR_ANALOGICA      Ap_var_analogica      reference  -    ANALOGICAS -
  CANASTA_PRIMARIA      Canasta_primaria      bit        1    -          -
  CANASTA_EN_SERVICIO   Canasta_en_servicio   bit        1    -          -
  CANASTA_CANAL         Canasta_canal         bit        1    -          -
  CANASTA_REAL          Canasta_real          bit        1    -          -
)CAMPOS

REGISTROS(
; Identificador     Nombre                Archivo       Arreglo            Cantidad  Anillo_libres
;   Registros de variables
  LOGICAS           Reg_logicas           logic.dat     tab_logicas          1450    LIB_LOGICAS
    CAMPOS(
      PUNTO_BLOQUEADO       $CAM_INH              ALARMA_INH            VAR_ALARMADA
      EVENTO_INH            VALOR_DADO            ALARMA_NO_RECONOCIDA  ACCION_INH
      ACCION_EN_PROGRESO    NO_RETAL              ESTADO_NORMAL         ALARMA_OA1
      ALARMA_1AO            IDENTIFICADOR         DESCRIPCION           PRIORIDAD
      ZUMBA_ALARMA          IMPRIME_ALARMA        AP_ESTADO_NORMAL      AP_ESTADO_ANORMAL
      AP_ACCION             DURACION_ACCION       RETRASO_RETAL         AP_RETAL
      VALOR_RETAL_ESPERADO  VALOR_LOGICO          ALARMAR_DIAGRAMA      AP_SISTEMA
      AP_SUBSISTEMA         COLOR_VAR             CANASTA               TARJETA
      BIT
    )CAMPOS
  ANALOGICAS        Reg_analogicas        analog.dat    tab_analogicas       1200    LIB_ANALOGICAS
    CAMPOS(
      PUNTO_BLOQUEADO       $CAM_INH              ALARMA_INH            VAR_ALARMADA
      EVENTO_INH            ESTADO_FUNCIONAL      VALOR_DADO            CALIDAD_DATO
      BANDA_MUERTA          BANDA_MUERTA_PUESTA   ALARMA_RETORNO        LIM_DINAMICO
      LIM_DINAMICO_PUESTO   ALARMA_NO_RECONOCIDA  ACCION_INH            ACCION_EN_PROGRESO
      IDENTIFICADOR         DESCRIPCION           PRIORIDAD             PORC_BANDA_MUERTA
      PORC_LIM_DINAMICO     LIM_CRITICO_ALTO      LIM_CRITICO_BAJO      LIM_PRECRITICO_ALTO
      LIM_PRECRITICO_BAJO   AP_LINEARIZACION      VALOR_ANALOGICO       AP_UNIDAD
      ZUMBA_ALARMA          IMPRIME_ALARMA        AP_INTEGRACION        LIM_OP_ALTO
      LIM_OP_BAJO           RANGO_GRAF_INFERIOR   RANGO_GRAF_SUPERIOR   ALARMAR_DIAGRAMA
      GANANCIA_SENAL        AP_SISTEMA            AP_SUBSISTEMA         COLOR_VAR
      TENDENCIA_VAR         CANASTA               TARJETA               BIT
    )CAMPOS
  LOGICAS_CALC      Reg_logicas_calc      logic_c.dat   tab_logicas_calc       65    LIB_LOGICAS_CALC
    CAMPOS(
      PUNTO_BLOQUEADO       $CAM_INH              ALARMA_INH            VAR_ALARMADA
      EVENTO_INH            VALOR_DADO            ALARMA_NO_RECONOCIDA  ESTADO_NORMAL
      ALARMA_OA1            ALARMA_1AO            IDENTIFICADOR         DESCRIPCION
      PRIORIDAD             ZUMBA_ALARMA          IMPRIME_ALARMA        AP_ESTADO_NORMAL
      AP_ESTADO_ANORMAL     VALOR_LOGICO          ALARMAR_DIAGRAMA      AP_SISTEMA
      AP_SUBSISTEMA         COLOR_VAR             CANASTA               TARJETA
      BIT
    )CAMPOS
  ANALOGICAS_CALC   Reg_analogicas_calc   analog_c.dat   tab_analogicas_calc    240    LIB_ANALOGICAS_CALC
    CAMPOS(
      PUNTO_BLOQUEADO       $CAM_INH              ALARMA_INH            VAR_ALARMADA
      EVENTO_INH            ESTADO_FUNCIONAL      VALOR_DADO            BANDA_MUERTA
      BANDA_MUERTA_PUESTA   ALARMA_RETORNO        LIM_DINAMICO          LIM_DINAMICO_PUESTO
      ALARMA_NO_RECONOCIDA  IDENTIFICADOR         DESCRIPCION           PRIORIDAD
```

```
    PORC_BANDA_MUERTA     PORC_LIN_DINAMICO     LIN_CRITICO_ALTO      LIN_CRITICO_BAJO
    LIN_PRECRITICO_ALTO   LIN_PRECRITICO_BAJO   AP_LINEARIZACION       AP_INTEGRACION
    VALOR_ANALOGICO       AP_UNIDAD             ZUMBA_ALARMA          IMPRIME_ALARMA
    LIN_OP_ALTO           LIN_OP_BAJO           RANGO_GRAF_INFERIOR   RANGO_GRAF_SUPERIOR
    ALARMAR_DIAGRAMA      AP_SISTEMA            AP_SUBSISTEMA         COLOR_VAR
    CANASTA               TARJETA               BIT
  )CAMPOS
CONSTANTES      Reg_constantes      const.dat     tab_constantes        700    LIB_CONSTANTES
  CAMPOS(
    PUNTO_BLOQUEADO       SCAN_INH              VALOR_DADO            IDENTIFICADOR
    DESCRIPCION           PRIORIDAD             VALOR_ANALOGICO       AP_UNIDAD
    AP_SISTEMA            AP_SUBSISTEMA         CANASTA               TARJETA
    BIT
  )CAMPOS
DIAGNOSTICOS    Reg_diagnosticos    diagnost.dat  tab_diagnosticos      600    LIB_DIAGNOSTICOS
  CAMPOS(
    PUNTO_BLOQUEADO       SCAN_INH              VALOR_DADO            ESTADO_NORMAL
    ALARMA_0A1            ALARMA_1A0            IDENTIFICADOR         DESCRIPCION
    PRIORIDAD             AP_ESTADO_NORMAL      AP_ESTADO_ANORMAL     VALOR_LOGICO
    AP_SISTEMA            AP_SUBSISTEMA         CANASTA               TARJETA
    BIT
  )CAMPOS
CONFIGURACIONES Reg_configuraciones config.dat    tab_configuraciones   215    LIB_CONFIGURACIONES
  CAMPOS(
    PUNTO_BLOQUEADO       SCAN_INH              VALOR_DADO            IDENTIFICADOR
    DESCRIPCION           PRIORIDAD             VALOR_ANALOGICO       AP_SISTEMA
    AP_SUBSISTEMA         CANASTA               TARJETA               BIT
  )CAMPOS
;  Registros auxiliares
UNIDADES        Reg_unidades        unidades.dat  tab_unidades          50     LIB_UNIDADES
  CAMPOS( NOMBRE_UNIDAD )CAMPOS
ESTADOS         Reg_estados         estados.dat   tab_estados           50     LIB_ESTADOS
  CAMPOS( NOMBRE_ESTADO )CAMPOS
ACCIONES        Reg_acciones        acciones.dat  tab_acciones          15     LIB_ACCIONES
  CAMPOS( NOMBRE_ACCION )CAMPOS
SISTEMAS        Reg_sistemas        sist.dat      tab_sistemas          20     LIB_SISTEMAS
  CAMPOS( NOMBRE_SISTEMA )CAMPOS
SUBSISTEMAS     Reg_subsistemas     subsist.dat   tab_subsistemas       90     LIB_SUBSISTEMAS
  CAMPOS( NOMBRE_SUBSISTEMA  SISTEMAS_VALIDOS )CAMPOS
LINEARIZACIONES Reg_linearizaciones liner.dat     tab_linearizaciones   15     LIB_LINEARIZACIONES
  CAMPOS( NOMBRE_LINEARIZACION  FUNCION_LINEARIZACION )CAMPOS
PROGRAMAS       Reg_programas       prog.dat      tab_programas         20     LIB_PROGRAMAS
  CAMPOS( NOMBRE_PROGRAMA  NUMERO_PROGRAMA )CAMPOS
INTEGRACIONES   Reg_integraciones   integ.dat     tab_integraciones     240    LIB_INTEGRACIONES
  CAMPOS(
    FACTOR_INT            TIEMPO_MUESTREO_INT   TIEMPO_ACTUAL_INT     TIEMPO_MAXIMO_INT
    CUENTA_CORRIENTE_INT  CUENTA_MAXIMA_INT     CUENTA_ULTIMA_INT     AP_VAR_ANALOGICA
  )CAMPOS
CANASTAS        Reg_canastas        canastas.dat  tab_canastas          30     LIB_CANASTAS
  CAMPOS(
    CANASTA_PRIMARIA  CANASTA_EN_SERVICIO  CANASTA_CANAL  CANASTA_REAL
  )CAMPOS
)REGISTROS

INDICES(
; Identificador     tipo      no_niv  Asc/Desc  llav_unica  func()      campo_perte
  IND_IDENTIF       string      6       A           1       identif_kw   IDENTIF_PERTE
    CAMPOS( IDENTIFICADOR )CAMPOS
    REGISTROS(
      LOGICAS  ANALOGICAS  LOGICAS_CALC  ANALOGICAS_CALC  CONSTANTES  DIAGNOSTICOS  CONFIGURACIONES
    )REGISTROS
  IND_CRONOAL       enum        -       -           0       -            CRONOAL_PERTE
    CAMPOS( )CAMPOS
    REGISTROS(
      LOGICAS  ANALOGICAS  LOGICAS_CALC  ANALOGICAS_CALC  CONSTANTES
    )REGISTROS
  IND_PRIORAL       int         -       A           0       priora1_kw   PRIORAL_PERTE
    CAMPOS( PRIORIDAD )CAMPOS
    REGISTROS(
      LOGICAS  ANALOGICAS  LOGICAS_CALC  ANALOGICAS_CALC  CONSTANTES
```

```
    )REGISTROS
  !MD_SISYSUB      string      B      A            0     misysub_kw   SISYSUB_PERTE
    CAMPOS( AP_SISTEMA  AP_SUBSISTEMA  !DENTIFICADOR )CAMPOS
    REGISTROS(
      LOGICAS  ANALOGICAS  LOGICAS_CALC  ANALOGICAS_CALC  CONSTANTES  DIAGNOSTICOS  CONFIGURACIONES
    )REGISTROS
  !MD_DIRHW          user        -      -            -      -           -
    CAMPOS(  )CAMPOS
    REGISTROS(  )REGISTROS
)!NDICES

ENFOQUES(
; Se dan los nombres de los identificadores de enfoque
    SISTEMA             PI_ADENDEME         PI_ALARMAS          PI_MSE
    PI_HISTORIAS        PI_CALCULOS         PI_SPOOLER          PI_FALLAS
    PI_ADMONFP          FP_LISVAR           FP_DIALAR_C         FP_DIALAR_P
    FP_GUIAS            FP_REPORTES         FP_ARRYPAR          FP_GRACOOR
    FP_DIAFLU           FP_BARRAS           FP_DIATEN           FP_GRAPAP
    FP_DIACON           FP_DIASER
)ENFOQUES

INCLUIR(
; Nombre de los archivos que se incluyen y que deben contener las funciones
; para el cálculo de las llaves de los índices definidos y de aquellas para
; el control de los índices de usuario
   fun_bdgp.c
)INCLUIR
```

APENDICE B

ARCHIVO DE ENCABEZADO BD_DESC.H

```
/* ARCHIVO bd_desc.h */

#ifndef BD_DESC_H
#define BD_DESC_H  1

#include <stdio.h>
#include <string.h>

#include $mutex

/* Códigos de error */
#include "BDERR.H"

#define BD_OPEN_RDWR  514
#define BD_INT  unsigned short
#define BD_BIT  unsigned
#define NULO    0

/* Tipos de Campos e índicee */
#define BD_SHORT          1
#define BD_INTEGER        2
#define BD_FLOAT          4
#define BD_DOUBLE         8
#define BD_CHAR          16
#define BD_STRING       256
#define BD_POINTER      512
#define BD_REFERENCE   1024
#define BD_NBITS       2048
#define BD_UNSIGNED    4096

/* Tipos de índice especiales */
#define BD_USER           10
#define BD_ENUM           11

/* Identificadores de los campos */
#define PUNTO_BLOQUEADO            1
#define SCAN_INH                   2
#define ALARMA_INH                 3
#define VAR_ALARMADA               4
#define EVENTO_INH                 5
#define ESTADO_FUNCIONAL           6
#define VALOR_DADO                 7
#define CALIDAD_DATO               8
#define BANDA_MUERTA               9
#define BANDA_MUERTA_PUESTA       10
#define ALARMA_RETORNO            11
#define LIM_DINAMICO              12
#define LIM_DINAMICO_PUESTO       13
#define ALARMA_NO_RECONOCIDA      14
#define ACCION_INH                15
#define ACCION_EN_PROGRESO        16
#define NO_RETAL                  17
#define ESTADO_NORMAL             18
```

```
#define ALARMA_0A1                19
#define ALARMA_1A0                20
#define IDENTIFICADOR             21
#define DESCRIPCION               22
#define PRIORIDAD                 23
#define PORC_BANDA_MUERTA         24
#define PORC_LIM_DINAMICO         25
#define LIM_CRITICO_ALTO          26
#define LIM_CRITICO_BAJO          27
#define LIM_PRECRITICO_ALTO       28
#define LIM_PRECRITICO_BAJO       29
#define AP_LINEARIZACION          30
#define VALOR_ANALOGICO           31
#define AP_UNIDAD                 32
#define ZUMBA_ALARMA              33
#define IMPRIME_ALARMA            34
#define AP_ESTADO_NORMAL          35
#define AP_ESTADO_ANORMAL         36
#define AP_ACCION                 37
#define DURACION_ACCION           38
#define RETRASO_RETAL             39
#define AP_RETAL                  40
#define VALOR_RETAL_ESPERADO      41
#define VALOR_LOGICO              42
#define AP_INTEGRACION            43
#define LIM_OP_ALTO               44
#define LIM_OP_BAJO               45
#define RANGO_GRAF_INFERIOR       46
#define RANGO_GRAF_SUPERIOR       47
#define ALARMAR_DIAGRAMA          48
#define GANANCIA_SENAL            49
#define AP_SISTEMA                50
#define AP_SUBSISTEMA             51
#define COLOR_VAR                 52
#define TENDENCIA_VAR             53
#define CANASTA                   54
#define TARJETA                   55
#define BIT                       56
#define NOMBRE_UNIDAD             57
#define NOMBRE_ESTADO             58
#define NOMBRE_ACCION             59
#define NOMBRE_SISTEMA            60
#define NOMBRE_SUBSISTEMA         61
#define SISTEMAS_VALIDOS          62
#define NOMBRE_LINEARIZACION      63
#define FUNCION_LINEARIZACION     64
#define NOMBRE_PROGRAMA           65
#define NUMERO_PROGRAMA           66
#define FACTOR_INT                67
#define TIEMPO_MUESTREO_INT       68
#define TIEMPO_ACTUAL_INT         69
#define TIEMPO_MAXIMO_INT         70
#define CUENTA_CORRIENTE_INT      71
```

```
#define CUENTA_MAXIMA_INT          72
#define CUENTA_ULTIMA_INT          73
#define AP_VAR_ANALOGICA           74
#define CANASTA_PRIMARIA           75
#define CANASTA_EN_SERVICIO        76
#define CANASTA_CANAL              77
#define CANASTA_REAL               78
#define IDENTIF_PERTE              79
#define CRONOAL_PERTE              80
#define PRIORAL_PERTE              81
#define SISYSUB_PERTE              82
#define LIBRES_PERTE               83

/* Identificadores de los registros */
#define LOGICAS                    1
#define ANALOGICAS                 2
#define LOGICAS_CALC               3
#define ANALOGICAS_CALC            4
#define CONSTANTES                 5
#define DIAGNOSTICOS               6
#define CONFIGURACIONES            7
#define UNIDADES                   8
#define ESTADOS                    9
#define ACCIONES                   10
#define SISTEMAS                   11
#define SUBSISTEMAS                12
#define LINEARIZACIONES            13
#define PROGRAMAS                  14
#define INTEGRACIONES              15
#define CANASTAS                   16

/* Identificadores de los indices */
#define IND_IDENTIF                1
#define IND_CRONOAL                2
#define IND_PRIORAL                3
#define IND_SISYSUB                4
#define IND_DIRHW                  5

/* Identificadores de los indices de registros libres */
#define LIB_LOGICAS                6
#define LIB_ANALOGICAS             7
#define LIB_LOGICAS_CALC           8
#define LIB_ANALOGICAS_CALC        9
#define LIB_CONSTANTES             10
#define LIB_DIAGNOSTICOS           11
#define LIB_CONFIGURACIONES        12
#define LIB_UNIDADES               13
#define LIB_ESTADOS                14
#define LIB_ACCIONES               15
#define LIB_SISTEMAS               16
#define LIB_SUBSISTEMAS            17
#define LIB_LINEARIZACIONES        18
#define LIB_PROGRAMAS              19
```

```
#define LIB_INTEGRACIONES          20
#define LIB_CANASTAS               21

/* Identificadores de los enfoques */
#define SISTEMA                    1
#define PI_ADENDEME                2
#define PI_ALARMAS                 3
#define PI_MSE                     4
#define PI_HISTORIAS               5
#define PI_CALCULOS                6
#define PI_SPOOLER                 7
#define PI_FALLAS                  8
#define PI_ADMONFP                 9
#define FP_LISVAR                  10
#define FP_DIALAR_C                11
#define FP_DIALAR_P                12
#define FP_GUIAS                   13
#define FP_REPORTES                14
#define FP_ARRYPAR                 15
#define FP_GRACOOR                 16
#define FP_DIAFLU                  17
#define FP_BARRAS                  18
#define FP_DIATEN                  19
#define FP_GRAPAP                  20
#define FP_DIACON                  21
#define FP_DIASER                  22

/* Número de campos, registros, indices y enfoques */
#define BD_NO_CAMPOS     83
#define BD_NO_REGISTROS  16
#define BD_NO_INDICES    21
#define BD_NO_ENFOQUES   22

/* Estructura para la protección por MUTEX */
extern struct bd_struct_prot {
  MUTEX  mutex;
}bd_obj_mutex[BD_NO_REGISTROS+BD_NO_INDICES+1];

/* Descriptor de campos */
extern struct bd_desc_campo {
  BD_INT  tipo;
  BD_INT  longitud;
  BD_INT  reg_ref;
  BD_INT  campo_ref;
  BD_BIT  indexado:1;
}bdd_campo[BD_NO_CAMPOS+1];

/* Descriptor de registros */
extern struct bd_desc_registro {
  struct bd_struct_prot  *prot;
  char     archivo[11];
  int      ap_arch;
  BD_INT   longitud;
```

```
  BD_INT  cantidad;
  char    *arreglo;
  BD_INT  libres;
  BD_INT  despl[BD_NO_CAMPOS+1];
  char    mascara[BD_NO_CAMPOS+1];
}bdd_registro[BD_NO_REGISTROS+1];

/* Descriptor de indices */
extern struct bd_desc_indice {
  struct bd_struct_prot  *prot;
  BD_INT   tipo;
  BD_INT   no_niveles;
  BD_BIT   orden:1;
  BD_BIT   llave_unica:1;
  struct bd_nodo   *raiz;
  int      (*func)();
  BD_INT   campo_perte;
  char     campo_llave[BD_NO_CAMPOS+1];
}bdd_indice[BD_NO_INDICES+1];

/* Descriptor de enfoques */
extern struct bd_desc_enfoque {
  BD_INT  no_indice;
  char    *registro;
  char    filtro_activo;
  unsigned char          filtro[9];
  unsigned char          llave[9];
  struct bd_desc_indice  *indice;
  struct bd_nodo         *posicion;
}bdd_enfoque[BD_NO_ENFOQUES+1];

/* Estructura bd_nodo */
extern struct bd_nodo {
  unsigned char    valor;
  struct bd_nodo   *sig_nodo;
  struct bd_nodo   *sig_nivel;
};

/* Area para campos invalidos              */
/*   (previene access-violation en VAX)    */
extern char bd_NULL[16];

/* Estructuras de los registros declarados en el
   esquema y sus arreglos */
extern struct Reg_logicas {
  char      tipo_reg;
  unsigned  libres_perte:1;
  unsigned  Punto_bloqueado:1;
  unsigned  Scan_Inhibido:1;
  unsigned  Alarma_inhibida:1;
  unsigned  Bandera_alarma:1;
  unsigned  Evento_Inhibido:1;
  unsigned  Valor_operador:1;
```

```
  unsigned  Alarma_nreconocida:1;
  unsigned  Control_inhibido:1;
  unsigned  Control_progreso:1;
  unsigned  No_retro:1;
  unsigned  Estado_normal:1;
  unsigned  Alarma_01:1;
  unsigned  Alarma_10:1;
  unsigned  Estado_control:1;
  unsigned  Valor_logico:1;
  unsigned  Alarmar_diagrama:1;
  unsigned  identif_perte:1;
  unsigned  cronoal_perte:1;
  unsigned  prioral_perte:1;
  unsigned  sisysub_perte:1;
  char      Identificador[7];
  char      Descripcion[31];
  char      Prioridad;
  unsigned short  Zumba_alarma;
  unsigned short  Imprime_alarma;
  BD_INT    Ap_estado_normal;
  BD_INT    Ap_estado_anormal;
  BD_INT    Accion_control;
  int       Timer_control;
  int       Timer_retro_control;
  BD_INT    Ap_retro;
  BD_INT    Numero_sistema;
  BD_INT    Numero_subsistema;
  char      Color;
  char      Canasta;
  char      Tarjeta;
  char      Bit;
}tab_logicas[1450];

extern struct Reg_analogicas {
  char      tipo_reg;
  unsigned  libres_perte:1;
  unsigned  Punto_bloqueado:1;
  unsigned  Scan_inhibido:1;
  unsigned  Alarma_inhibida:1;
  unsigned  Bandera_alarma:1;
  unsigned  Evento_Inhibido:1;
  unsigned  Estado_funcional:1;
  unsigned  Valor_operador:1;
  unsigned  Calidad_dato:1;
  unsigned  Banda_muerta:1;
  unsigned  Banda_puesta:1;
  unsigned  Alarma_retorno:1;
  unsigned  Limite_dinamico:1;
  unsigned  Limite_puesto:1;
  unsigned  Alarma_nreconocida:1;
  unsigned  Control_inhibido:1;
  unsigned  Control_progreso:1;
  unsigned  Alarmar_diagrama:1;
```

```
  unsigned  identif_perte:1;
  unsigned  cronoal_perte:1;
  unsigned  prioral_perte:1;
  unsigned  sisysub_perte:1;
  char      Identificador[7];
  char      Descripcion[31];
  char      Prioridad;
  char      Porc_banda_muerta;
  char      Porc_limite_dinamico;
  unsigned short  Limite_crit_alto;
  unsigned short  Limite_crit_bajo;
  unsigned short  Limite_prec_alto;
  unsigned short  Limite_prec_bajo;
  BD_INT    Ap_linearizacion;
  unsigned short  Valor_variable;
  BD_INT    Unidades;
  unsigned short  Zumba_alarma;
  unsigned short  Imprime_alarma;
  BD_INT    Ap_est_integracion;
  float     Limite_oper_alto;
  float     Limite_oper_bajo;
  float     Rango_gra_papel_inf;
  float     Rango_gra_papel_sup;
  char      Ganancia;
  BD_INT    Numero_sistema;
  BD_INT    Numero_subsistema;
  char      Color;
  char      Tendencia;
  char      Canasta;
  char      Tarjeta;
  char      Bit;
}tab_analogicas[1200];

extern struct Reg_logicas_calc {
  char      tipo_reg;
  unsigned  libres_perte:1;
  unsigned  Punto_bloqueado:1;
  unsigned  Scan_Inhibido:1;
  unsigned  Alarma_inhibida:1;
  unsigned  Bandera_alarma:1;
  unsigned  Evento_Inhibido:1;
  unsigned  Valor_operador:1;
  unsigned  Alarma_nreconocida:1;
  unsigned  Estado_normal:1;
  unsigned  Alarma_01:1;
  unsigned  Alarma_10:1;
  unsigned  Valor_logico:1;
  unsigned  Alarmar_diagrama:1;
  unsigned  identif_perte:1;
  unsigned  cronoal_perte:1;
  unsigned  prioral_perte:1;
  unsigned  sisysub_perte:1;
  char      Identificador[7];
```

```
  char      Descripcion[31];
  char      Prioridad;
  unsigned short  Zumba_alarma;
  unsigned short  Imprime_alarma;
  BD_INT    Ap_estado_normal;
  BD_INT    Ap_estado_anormal;
  BD_INT    Numero_sistema;
  BD_INT    Numero_subsistema;
  char      Color;
  char      Canasta;
  char      Tarjeta;
  char      Bit;
}tab_logicas_calc[65];

extern struct Reg_analogicas_calc {
  char      tipo_reg;
  unsigned  libres_perte:1;
  unsigned  Punto_bloqueado:1;
  unsigned  Scan_Inhibido:1;
  unsigned  Alarma_inhibida:1;
  unsigned  Bandera_alarma:1;
  unsigned  Evento_Inhibido:1;
  unsigned  Estado_funcional:1;
  unsigned  Valor_operador:1;
  unsigned  Banda_muerta:1;
  unsigned  Banda_puesta:1;
  unsigned  Alarma_retorno:1;
  unsigned  Limite_dinamico:1;
  unsigned  Limite_puesto:1;
  unsigned  Alarma_nreconocida:1;
  unsigned  Alarmar_diagrama:1;
  unsigned  identif_perte:1;
  unsigned  cronoal_perte:1;
  unsigned  prioral_perte:1;
  unsigned  sisysub_perte:1;
  char      Identificador[7];
  char      Descripcion[31];
  char      Prioridad;
  char      Porc_banda_muerta;
  char      Porc_limite_dinamico;
  unsigned short  Limite_crit_alto;
  unsigned short  Limite_crit_bajo;
  unsigned short  Limite_prec_alto;
  unsigned short  Limite_prec_bajo;
  BD_INT    Ap_linearizacion;
  BD_INT    Ap_est_integracion;
  unsigned short  Valor_variable;
  BD_INT    Unidades;
  unsigned short  Zumba_alarma;
  unsigned short  Imprime_alarma;
  float     Limite_oper_alto;
  float     Limite_oper_bajo;
  float     Rango_gra_papel_inf;
```

```
  float     Rango_gra_papel_sup;
  BD_INT    Numero_sistema;
  BD_INT    Numero_subsistema;
  char      Color;
  char      Canasta;
  char      Tarjeta;
  char      Bit;
}tab_analogicas_calc[240];

extern struct Reg_constantes {
  char      tipo_reg;
  unsigned  libres_perte:1;
  unsigned  Punto_bloqueado:1;
  unsigned  Scan_Inhibido:1;
  unsigned  Valor_operador:1;
  unsigned  identIf_perte:1;
  unsigned  cronoal_perte:1;
  unsigned  prioral_perte:1;
  unsigned  sisysub_perte:1;
  char      Identificador[7];
  char      Descripcion[31];
  char      Prioridad;
  unsigned short  Valor_variable;
  BD_INT    Unidades;
  BD_INT    Numero_sistema;
  BD_INT    Numero_subsistema;
  char      Canasta;
  char      Tarjeta;
  char      Bit;
}tab_constantes[700];

extern struct Reg_diagnosticos {
  char      tipo_reg;
  unsigned  libres_perte:1;
  unsigned  Punto_bloqueado:1;
  unsigned  Scan_Inhibido:1;
  unsigned  Valor_operador:1;
  unsigned  Estado_normal:1;
  unsigned  Alarma_01:1;
  unsigned  Alarma_10:1;
  unsigned  Valor_logico:1;
  unsigned  identIf_perte:1;
  unsigned  sisysub_perte:1;
  char      Identificador[7];
  char      Descripcion[31];
  char      Prioridad;
  BD_INT    Ap_estado_normal;
  BD_INT    Ap_estado_anormal;
  BD_INT    Numero_sistema;
  BD_INT    Numero_subsistema;
  char      Canasta;
  char      Tarjeta;
  char      Bit;
```

```
}tab_diagnosticos[600];

extern struct Reg_configuraciones {
  char      tipo_reg;
  unsigned  libres_perte:1;
  unsigned  Punto_bloqueado:1;
  unsigned  Scan_Inhibido:1;
  unsigned  Valor_operador:1;
  unsigned  identif_perte:1;
  unsigned  sisysub_perte:1;
  char      Identificador[7];
  char      Descripcion[31];
  char      Prioridad;
  unsigned short  Valor_variable;
  BD_INT    Numero_sistema;
  BD_INT    Numero_subsistema;
  char      Canasta;
  char      Tarjeta;
  char      Bit;
}tab_configuraciones[215];

extern struct Reg_unidades {
  char      tipo_reg;
  unsigned  libres_perte:1;
  char      Nombre_unidades[7];
}tab_unidades[50];

extern struct Reg_estados {
  char      tipo_reg;
  unsigned  libres_perte:1;
  char      Nombre_estado[7];
}tab_estados[50];

extern struct Reg_acciones {
  char      tipo_reg;
  unsigned  libres_perte:1;
  char      Nombre_accion[8];
}tab_acciones[15];

extern struct Reg_sistemas {
  char      tipo_reg;
  unsigned  libres_perte:1;
  char      Nombre_sistema[32];
}tab_sistemas[20];

extern struct Reg_subsistemas {
  char      tipo_reg;
  unsigned  libres_perte:1;
  char      Nombre_subsistema[32];
  unsigned int  Sistemas_validos;
}tab_subsistemas[90];

extern struct Reg_linearizaciones {
```

```
  char      tipo_reg;
  unsigned  libres_perte:1;
  char      Nombre_linearizacion[8];
  char  *   Funcion_linearizacion;
}tab_linearizaciones[15];

extern struct Reg_programas {
  char      tipo_reg;
  unsigned  libres_perte:1;
  char      Nombre_programa[32];
  unsigned short  Numero_programa;
}tab_programas[20];

extern struct Reg_integraciones {
  char      tipo_reg;
  unsigned  libres_perte:1;
  float     Factor_int;
  int       Tiempo_muestreo_int;
  int       Tiempo_actual_int;
  int       Tiempo_maximo_int;
  float     Cuenta_corriente_int;
  float     Cuenta_maxima_int;
  float     Cuenta_ultima_int;
  BD_INT    Ap_var_analogica;
}tab_integraciones[240];

extern struct Reg_canastas {
  char      tipo_reg;
  unsigned  libres_perte:1;
  unsigned  Canasta_primaria:1;
  unsigned  Canasta_en_servicio:1;
  unsigned  Canasta_canal:1;
  unsigned  Canasta_real:1;
}tab_canastas[30];

/* Declaración de tipos de las funciones */
extern int      identif_kw(),prioral_kw(),sisysub_kw();

extern int      bd_inicia_bd(),bd_fin_bd();
extern BD_INT   bd_despl_bit();
extern char     bd_mascara_bit();
extern BD_INT
bd_asign(),bd_llave(),bd_busca(),bd_perte(),bd_salta();
extern BD_INT   bd_ccamp(),bd_reemp(),bd_apunt(),bd_apreg();
extern BD_INT   bd_ccamp_reg(),bd_reemp_reg(),bd_apunt_reg();
extern BD_INT   bd_filtr(),bd_index(),bd_xinde(),bd_copia();
extern BD_INT   bd_borra(),bd_idreg(),bd_idtip(),bd_idind();
extern BD_INT   bd_carg0(),bd_carg1(),bd_cier0(),bd_cier1();
extern BD_INT   bd_escr0(),bd_escr1(),bd_escr2(),bd_debug();
extern char     *bd_campo(),*bd_campo_reg(),*bd_regap();
extern char     *bd_regis(),*bd_agreg();

#endif
```

NO

EXISTE

PAGINA

APENDICE C

ARCHIVO DE ENCABEZADO BDERR.H

NO

EXISTE

PAGINA

```
#ifndef BDERR_H
#define BDERR_H  1

/* Los comentarios que inician con (!) indican los códigos que
   se toman en cuenta cuando el nivel de detección es BD_BUGSYS */

/* Niveles de detección */

#define BD_BUGON    0xfff1  /* mensajes de error habilitados */
#define BD_BUGOFF   0xfff0  /* mensajes deshabilitados */
#define BD_BUGSYS   0xfff8  /* solo mensajes muy importantes */

/* Códigos de error */

#define BDE_SOLMEM      1   /* ! falla en la solicitud dinámica de memoria */
#define BDE_NOREGLIB    2   /* ! no hay mas registros libres */
#define BDE_REGARR      3   /* ! la dirección del registro es incorrecta */
#define BDE_REGNULL     4   /* ! el registro está en NULL */

#define BDE_NOASIGN    11   /* id_e sin asignacion de id_i */
#define BDE_CAMPINV    12   /* campo no esta en la estructura del registro */
#define BDE_CAMPREF    13   /* el campo debe ser tipo referencial */
#define BDE_REFINV     14   /* ! campo referencial < 0 o > # de registros */

#define BDE_IDENF      21   /* ! id_e desconocido */
#define BDE_IDIND      22   /* ! id_i desconocido */
#define BDE_IDCAM      23   /* ! id_c desconocido */
#define BDE_IDREG      24   /* ! tipo de registro desconocido */

#define BDE_INDASIG    31   /* id_i asignado a un id_e, no se puede borrar */
#define BDE_INDVAC     32   /* índice vacio */
#define BDE_FININD     33   /* fin de índice */
#define BDE_STRINT     34   /* ! id_i debe ser tipo BD_STRING o BD_INTEGER */
#define BDE_USRIND     35   /* el índice no debe ser tipo BD_USER */

#define BDE_DUPLIC     41   /* intento de duplicación de llave en índice */
#define BDE_LONG       42   /* ! llave tipo string de longitud incorrecta */
#define BDE_BUSC       43   /* llave no encontrada */
#define BDE_LLAVE      44   /* llave NULL */

#define BDE_PERTEX     51   /* registro sin campo_parte a id_i */
#define BDE_PERTE1     52   /* ! campo_parte en 1, registro sin indexar */
#define BDE_PERTE0     53   /* campo_parte en 0 */

#define BDE_OPEN       61   /* ! error al abrir archivo */
#define BDE_CLOSE      62   /* ! error al cerrar archivos */
#define BDE_READ       63   /* ! error al adquirir datos de archivo */
#define BDE_WRITE      64   /* ! error al escribir datos al archivo */
#define BDE_LSEEK      65   /* ! error al moverse dentro del archivo */

#endif
```