UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

"ARAGON"

METODOLOGIA DE DISEÑO Y APLICACION DE LAS BASES DE DATOS, UN ENFOQUE MULTIDISCIPLINARIO

T E S I S

QUE PARA OBTENER EL TITULO DE :

INGENIERO EN COMPUTACION PRESENTAN:

MARTHA LUCILA ALANIS ROJANO

HUGO REYES ALONSO

1 9 9 3.

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

I N D I C E

I N D I C E

I N D I C E

CAPITULO 6.- PLANEACION DE LA BASE DE DATOS. EL DISEÑO LOGICO.

CAPITULO 7.- IMPLEMENTACION DE LA BASE DE DATOS. EL DISEÑO FISICO.

CAPITULO 8.- SOLUCION DE PROBLEMAS MEDIANTE USO DE BASES DE DATOS.

INTRODUCCION.

Debido a las recientes reformas que se han dado dentro del contexto de la Universidad Nacional Autónoma de México, como una respuesta a la inquietud de autoridades, profesores y alumnado, se plantea el hecho de que los planes de estudio en lo referente a la carrera de Ingeniería en Computación deben ser modernizados. Es así como surge la necesidad de implementar uno de los cursos más necesarios en las universidades y, en general, en todos los lugares donde se enseña computación; esto es, un curso sobre la realidad de la tecnología de las bases de datos.

El Desarrollo de las bases de datos colectivas es sin duda una de las actividades más importantes en el campo de la informática; actualmente, es impresionante observar cómo crecen en importancia y en volúmenes los archivos de datos de las computadoras.

Las tasas de crecimiento de la capacidad de almacenamiento de las computadoras, han hecho posible que se tenga un desmesurado crecimiento de la disponibilidad de información, con lo que se apoya definitivamente el gran desarrollo industrial, comercial, científico y de servicios de nuestro tiempo; Cuanto mayor es la cantidad de datos a que tiene acceso la computadora, tanto mayor es su potencial. Es por esto que las Bases de Datos, como una herramienta de manejo de datos, cobran una importancia tal vez aún mayor que la de la invención de la imprenta.

Cada vez son más las áreas de aplicación de las bases de datos. Se puede mencionar entre muchas de ellas a las siguientes:

- Manufactura de control de inventarios.
- Procesamiento de facturas de materiales.
- Manejo cronológico de equipo de producción
- En el gobierno a todos los niveles, con registros de causantes, de propiedades, de electores, etc.
- En las instituciones financieras, con listas de cuentas individuales, inversiones, datos para análisis de convertibilidad de fondos, etc.
- Industrias de servicios (Hotelería, gastronomía, transporte, etc.), con listas de capacidad de servicios, datos de miembros, listas de empleados, programas de asignación, etc.
- Servicios médicos con registros de pacientes, historias clínicas, clasificación de problemas, datos de efectividad de tratamiento, etc.
- Modelado económico, utilizando datos de producción y consumo para planeación y asignación de recursos.

- En investigación científica, mediante la utilización de bases de conocimiento y sistemas expertos, estas bases de datos contienen información previamente obtenida, y son empleados para determinar comportamientos actuales del modelo experimental, así como posibles directrices de investigación.
- Oficinas que automatizan el manejo de información.
- Bibliotecas que catalogan resúmenes e índices de su acervo.
- Y en general, actualmente en muchas empresas e instituciones donde el manejo de los datos contenidos en cintas y paquetes es ineficiente, dada la gran cantidad de datos que se tienen y la dispersión con que se encuentran. Es en este tipo de aplicación donde las bases de datos juegan un papel muy importante.

Los programadores ven los datos cada uno a su modo y quieren siempre modificarlos a medida que varían sus necesidades, es por esto que la implementación de Bases de Datos debe contemplar que los datos se almacenen de manera que se les pueda utilizar indiferentemente para una amplia variedad de aplicaciones y que además pueda cambiarse fácil y rápidamente la manera de usarlos.

Es decir, los dos aspectos a cuidar en el diseño de Bases de Datos son :

a) que los datos sean independientes de los programas,
b) que debe ser posible interrogar y explorar la base de datos sin que esto implique escribir programas en lenguajes convencionales. En lugar de esto debe usarse lenguajes especiales de Base de Datos.

El problema de diseño de Bases de Datos es cada vez mayor, puesto que existe infinidad de software que se emplea erróneamente o se comprende mal. Hay muchas maneras de estructurar los datos y cada una de ellas tiene ventajas y desventajas. Cada tipo de datos exhibe características propias que afectan el modo de organizarlos y los requisitos que se imponen al diseñador son tantos que resulta imposible satisfacer las necesidades con una organización única. Es decir, existen muchos problemas interdependientes, pero también son muchas las técnicas para resolverlos. En este estudio nos enfocaremos a la filosofía básica de diseño de Bases de Datos para las microcomputadoras mas usuales.

Es necesario remarcar que pronto la tecnología habrá de reemplazar algunos de los aspectos de Hardware y Software que aquí se describan, pero los conceptos fundamentales del diseño de bases de datos seguirán intactos, por lo que será en estos puntos en los que nos detengamos para explicarlos más ampliamente, tratando de encaminar al lector hacia una comprensión y familiarización de las muchas alternativas que se tienen para organizar los datos, por lo tanto, esta obra debe servir para que, al conocerse los principios de diseño de Bases de Datos, el lector adapte las técnicas descritas a sus propias necesidades.

CAPITULO 1 : BASES DE DATOS, LA HERRAMIENTA INFORMATICA DEL FUTURO

1.1 DEFINICIONES.

1.1.1. QUE ES UNA BASE DE DATOS ?.

Algunas personas conciben a la Base de Datos como un enorme receptáculo en el que un organismo guarda todos los datos procesables que reúne y al cual acuden muy diversos usuarios a accesarlos. Este gran almacén puede estar ubicado en una misma localidad o distribuido en varias, todas ellas interconectadas mediante una red de telecomunicaciones. Tienen acceso a la base de datos diversos programas de distinta índole.

De nuestra experiencia diaria, tanto en la Secretaría de Educación Pública (SEP), como en Ferrocarriles Nacionales de México (FNM), nos hemos dado cuenta que esto no es posible, puesto que, en el caso de la SEP, que es un gran organismo, los datos referentes al pago de un empleado son encontrados en una Base de Datos diferente a la que guarda los datos escolares de un alumno de secundaria, además, cada una de las Coordinaciones Estatales de este organismo cuenta con una versión de estas dos Bases de Datos.

Es por esto que una base de datos semejante a la descrita anteriormente es todavía un sueño que no se ha hecho realidad, aunque debemos mencionar que sí existen servicios de información que cuentan con ésta infraestructura. Por el momento, toda Base de Datos más o menos complicada sirve sólo a un número limitado de aplicaciones, y conserva datos con íntima relación. De aquí se desprende que :

" *La base de datos puede definirse como una colección de datos interrelacionados, almacenados en conjunto sin redundancias perjudiciales o innecesarias; su finalidad es la de servir a una aplicación o más, de la mejor manera posible; los datos se almacenan de manera tal que resulten independientes de los programas que los usan; se emplean métodos bien determinados para incluír datos nuevos y para modificar o extraer datos almacenados. Dícese que un sistema comprende una colección de bases de datos cuando éstas son independientes desde el punto de vista de su estructura lógica*" [1].

1.1.2. REDUNDANCIA CONTROLADA.

La Base de Datos ha sido definida como una colección no redundante de items de datos, pero, en realidad, en muchas bases de datos se admite cierta redundancia con el fin de disminuír los tiempos de acceso o simplificar los métodos de direccionamiento. Existe la necesidad de armonizar el grado de redundancia con otras características deseables de la Base de Datos, de modo que es preferible hablar de *redundancia controlada*.

[1] MARTIN, JAMES. "ORGANIZACION DE LAS BASES DE DATOS", 1975, pag. 19.

Con la *redundancia controlada*, se eliminan eficientemente los inconvenientes de tener tantos datos redundantes que no se pueda tener a todos ellos en un mismo grado de actualización, ademas del eminente costo adicional de almacenamiento y de procesamiento de estos datos. Son estos factores los que desacreditan a un sistema de cómputo y lo podrían hacer parecer injustificado, afortunadamente, las bases de datos contemplan estos factores sin perjudicar onerosamente los tiempos de acceso y proceso de los datos.

1.1.3. INDEPENDENCIA DE LOS DATOS.

Esta idea implica que los datos y los programas de aplicación que de ellos se sirven son mutuamente independientes, de manera que unos puedan ser modificados sin tener en cuenta a los otros. En particular, el programador de aplicaciones no debe ser afectado por los cambios que sufran los datos en su estructura lógica o física.

En realidad, esta idea también es modificada pues pocas veces se cumple al pie de la letra, sin embargo, es uno de los más valiosos argumentos en pro de las Bases de Datos.

1.1.4. LOGICO Y FISICO.

La descripción de los datos y la de las relaciones que entre ellos existen adopta una de dos formas: *lógica y física.* La descripción física de los datos se ocupa de cómo se les registra en el hardware. La descripción lógica en cambio, se refiere a la forma como se le presentan al programador de aplicaciones. Las palabras lógico y físico se usarán para definir otras características de los datos, pero simpre se referirán, la primera a cómo ve los datos el programador de aplicaciones o el usuario, y la segunda, a la forma como se registran en los dispositivos de almacenamiento.

1.1.5. COMPARTIR DATOS

La idea es poseer la capacidad para que varios programas de aplicación compartan en forma independiente una base base de datos integrada. Es decir, que las representaciones de datos sean concurrentes y múltiples, que se tengan mecanismos de acceso eficientes para subconjuntos de datos específicos.

1.1.6. RELACIONABILIDAD

La habilidad para relacionar datos o relacionabilidad se usa para denotar la propiedad de la existencia de relaciones entre diferentes registros lógicos. Un registro representa un concepto del mundo real; por ejemplo nombre, dirección y salario de un empleado, las relaciones se dan entre registros por ejemplo: hijos de un padre, grados de un alumno, habilidades de un empleado.

1.1.7. INTEGRIDAD

El término integridad se refiere a la coordinación del acceso de datos por programas distintos, a la propagación de valores actualizados a otras copias y valores independientes, y asegurar la validez de los datos; incluye también una bitácora en donde se registran todos los accesos y cambios que afecten a cada dato.

1.1.8. FLEXIBILIDAD DE ACCESO

Se refiere a la capacidad de tener acceso a los datos en forma fácil de diferentes maneras y en base a diferentes llaves de acceso, la capacidad de usar un lenguage de consulta, y que la base pueda ser accesada por lenguajes convencionales.

1.1.9. SEGURIDAD

Se refiere a los mecanismos adecuados para asignar deerechos de acceso a los datos. Ciertos artículos y combinaciones o selecciones de ellos pueden ser sensibles y requerir de altos niveles da autorización para que se permita su acceso.. Un dato debe ser protegido contra actos maliciosos y no autorizados en una base de datos.

1.2. OBJETIVOS DE LAS BASES DE DATOS.

En el cuadro 1.1 se resumen los objetivos que debe perseguir toda organización de bases de datos, y que de alcanzarse, reportarían grandes ventajas sobre la organización de archivos convencionales, los objetivos de hecho, son la base de la cual se debe partir para calificar un cierto software de bases de datos.

Es claro que el software existente en el mercado tiene limitaciones que no permiten que cumpla con todos los objetivos que aquí se plantean, sin embargo, esto no quiere decir que sea malo, sino que el diseñador de la base de datos debe contemplar cuáles son sus necesidades primordiales, a fin de saber que tipo de software escoger para realizar su proyecto, ya que no todos tienen los mismos atributos ni contemplan los mismos tipos de computadoras en los que podrían ser implantados. Incluso, la configuración del equipo de cómputo con que se cuenta, influye en la decisión de escoger el software adecuado.

1.3. ALTERNATIVAS, VENTAJAS Y DESVENTAJAS DE UNA BASE DE DATOS

Ya hemos hablado de que los sistemas de bases de datos no cumplen en si una función "mágica" y es obsoleto pensar en ellos como la solución a toda clase de problemas, puesto que el enfoque de cada uno de los diferentes paquetes de software disponibles en el mercado difiere en cuanto a orientación específica. Sin embargo, pueden señalarse ventajas generales de la organización de bases de datos, así como sus desventajas.

- Los datos podrán accesarse de múltiples maneras.
Por diferentes ususarios que los perciben de manera diferente

- Se protegerá la inversión intelectual.

- Rapidez de desempeño

- Claridad
Para que los ususarios comprendan a que datos tienen acceso.

- Facilidad de acceso.

- Flexibilidad

- Rápida atención a problemas no previstos.

- Facilidad de cambio.

- Precisión y coherencia de datos.

- Control de acceso (reserva de datos).

- Protección contra pérdida o daño.

- Disponibilidad inmediata para los ususarios

- Independencia física y lógica de datos.

- Redundancia controlada.

- Fácil recuperación de datos en caso de falla.

- Funcionar como ayuda en el diseño y la supervisión.

Cuadro 1.1 Objetivos de las Bases de datos.

VENTAJAS

De la consecución de los objetivos listados en el cuadro 1.1, se desprende que:

- No será necesario rehacer programas o estructuras lógicas cuando se modifique la base de datos, lo cual sí sucedería si se tuvieran los datos en archivos convencionales.
- Se minimiza el costo de almacenamiento de datos, al eliminar la redundancia perjudicial y al existir facilidad de acceso a los mismos datos por diferentes aplicaciones.
- Se reduce el costo de capacitación del personal, debido a que no es necesario para todos los usuarios de los datos, el conocer las complejidades internas, sino sólo par el grupo de administradores de los datos.
- Se evita el exceso de programación, debido a que existen lenguajes de consulta.
- Se evita el acceso no autorizado a los datos. Los datos están protegidos contra fallas o acciones de personas que deseen falsearlos.
- El hardware de almacenamiento de datos y las técnicas de almacenamiento podrán ser modificados, así como el agregado de nuevos items de datos y la modificación de la estructura lógica de la base de datos podrán hacerse sin modificar los programas existentes, debido a la independencia física y lógica de datos.
- Se consigue una normalización de los datos dentro de un organismo, con la finalidad de no generar datos incompatibles, lo que llevará un considerable ahorro en los tiempos de acceso a informacion entre departamentos con diferentes actividades.

DESVENTAJAS

Quizá la mayor desventaja que se tenga al implementar un sistema de bases de datos, sobre todo en sitemas de microcomputadoras, sea el hecho de que los sistemas de bases de datos no estén hechos a la medida de los clientes, con lo que la conección de redes para facilitar el acceso a los datos se vuelve utópica puesto que las redes no son compatibles, por ejemplo, Televideo y Altos nunca tendrán el mismo software, lo cual hace a los propietarios de un sistema de bases de datos esclavo o prisionero de la compañía que vende el software que el utiliza.

1.4. CONCEPTOS BASICOS

1.4.1. LENGUAJE DE DEFINICION DE DATOS (DDL)

Un esquema de Base de Datos se especifica por medio de una serie de definiciones que se expresan en un lenguaje especial llamado Lenguaje de Definición de Datos (DDL.-Data Definition Language). El resultado de la compilación de éstas definiciones es una serie de instrucciones que especifica los detalles de implantación de los esquemas de Bases de Datos que normalmente no pueden ver los usuarios.

El DDL es el lenguaje de un sistema generalizado para manejo de Bases de Datos, usado para definir la estructura lógica de los datos, se tienen 3 lenguajes DDL:

-El DDL de esquemas para uso del DBA.
-El DDL de subesquemas para uso también del DBA y
-El DDL de subesquemas para uso del usuario.

Los DDL de subesquemas para el administrador y el usuario pueden ser muy similares.

1.4.2.LENGUAJE DE MANEJO DE DATOS (DML)

Un Lenguaje de Manejo de Datos (DML.-Data Manipulation Language) permite a los usuarios manejar o tener acceso a los datos que estén organizados por medio del modelo apropiado. Existen básicamente dos tipos de DML:

1.-De Procedimientos.-Necesitan que el usuario especifique cuáles datos quiere y cómo deben obtenerse.

2.-Sin Procedimientos.-Requieren que el usuario especifique cuáles datos quiere sin especificar cómo obtenerlos.

El DML es un conjunto de comandos de un sistema generalizado para manejo de Bases de Datos que se usa para almacenar, recopilar, actualizar, agregar y eliminar datos en una Base de Datos; el DML incluye a todos los comandos para Entrada/Salida y a otros para navegación en la Base. Los comandos del DML se usan en la división de procedimientos de un programa Cobol. El DML de un sistema generalizado para manejo de archivos es esencialmente un lenguaje de consultas, diseñado tanto para los requerimientos de la producción de reportes como en instalaciones donde existen sólo archivos sencillos y no Bases de Datos.

1.4.3. MANEJADOR DE BASE DE DATOS (DBM)

El manejador de Base de Datos (DBM.-Data Base Manipulation) es un módulo de programa que constituye la interfaz entre los datos de bajo nivel almacenados en la Base de Datos y los programas de aplicaciones y las consultas hechas al sistema.

El DBM es un sistema computarizado de mantenimiento de registros. Sistema cuyo propósito global es mantener información y hacer ésta información disponible cuando y donde se le demande. Es un sistema de software usado para manejar y mantener datos de una o múltiples aplicaciones al mismo tiempo para diferentes propósitos, independientemente de la clase de dispositivos de almacenamiento o métodos de acceso. Parte de las tareas de un DBM serán:

-La consistencia de los datos.

-Resolver los problemas de concurrencia.

-Proveer una interfaz universal a los datos.

-Regular el acceso a los datos.

1.4.4. ADMINISTRADOR DE BASE DE DATOS (DBA)

Una de las razones principales para contar con sistemas de Manejo de Bases de Datos es tener un control centralizado tanto de los datos como de los programas que tienen acceso a ellos. La persona que tiene este control centralizado sobre el sistema es el Administrador de la Base de Datos (DBA.-Data Base Administrator). Es decir, el DBA es uno o más individuos que se responsabilizan de la definición de los esquemas, subesquemas, derechos de acceso, niveles de rendimiento, verificaciones de la integridad, etc.; en general, el DBA tiene a su cargo tanto el control y la administración como el uso de la totalidad del Banco de Datos.

1.4.5. USUARIOS DE LA BASE DE DATOS

Típicamente se pueden diferenciar tres diferentes tipos de usuarios de una base de datos:

El diseñador.

Es la persona encargada de hacer el análisis de la información que va a residir en una base de datos, así como de implementar las estructuras lógicas que se usan para el manejo de dicha información, es decir, es aquel que se encarga de "crear" la base de datos.

El administrador.

Aunque se usa el singular, normalmente se trata de un grupo de personas que tienen conocimientos sobre la estructura física y lógica de la base de datos, lo cual les permite hacer modificaciones, crear derechos de acceso y en general, controlar el acceso a la información contenida en los archivos de base de datos. Normalmente estas personas se encargan de asesorar a los usuarios finales respecto a las posibilidades y restricciones del uso de la base de datos en cuestión.

Los usuarios finales.

Como ususarios finales debe conocerse a los *programadores de aplicaciones*, que se encargan de realizar programas que permitan la utilización de los datos contenidos en la base, y a los *ususarios de los programas*, típicamente capturistas u operadores, quienes sólo utilizan la información junto con los programas sin saber más acerca de ellos.

1.4.6. ARQUITECTURA DE UN SISTEMA DE BASES DE DATOS

Un Sistema de Bases de Datos se divide en módulos que se encargan de cada una de las tareas del sistema general; algunas de las funciones del Sistema de Bases de Datos pueden ser revisadas por el Sistema Operativo, pero en la mayor parte de los casos, el Sistema Operativo proporciona únicamente los servicios más elementales y la Base de Datos debe partir de ese fundamento.

Un sistema de Bases de Datos consiste de varios componentes funcionales entre los que se encuentran:

MANEJADOR DE ARCHIVOS:

Encargado de asignar espacio en el disco y de las estructuras de datos que se van a emplear para representar la información almacenada en el disco.

MANEJADOR DE BASES DE DATOS:

Constituye la interface entre los datos de bajo nivel almacenados en la Base de Datos y los programas de aplicaciones y las consultas que se hacen al sistema.

PROCESADOR DE CONSULTAS:

Traduce las proposiciones en el Lenguaje de Consulta a instrucciones de bajo nivel que puede entender el Manejador de la Base de Datos. Además, el Procesador de Consultas trata de convertir la solicitud del usuario a una forma equivalente pero más eficiente, encontrando una estrategia adecuada para ejecutar la consulta.

PRECOMPILADOR DE DML:

Convierte las Proposiones en DML incrustadas en un programa de aplicaciones en llamadas normales a procedimientos en Lenguaje Huesped. El Precompilador debe interactuar con el Procesador de Consultas para generar el código apropiado.

EL COMPILADOR DE DDL:
Convierte las proposiciones en DDL en un conjunto de tablas que contienen metadatos; tales tablas se almacenan después en el Diccionario de Datos.

Además se requieren varias estructuras de datos como parte de la implantación del sistema físico, incluyendo:

-Archivos de Datos, que guardan la Base de Datos.
-Diccionario de Datos, que almacena la información relativa a la estructura de la Base de Datos, se usa constantemente por lo que debe tenerse mucho cuidado de desarrollar un diseño apropiado y una implantación eficiente.
-Indices, que permiten el acceso rápido a elementos de información que contienen valores determinados.

1.5. MODELADO DE DATOS

El Modelado de Datos es un grupo de herramientas conceptuales para describir los datos, sus relaciones, su semántica y sus limitantes. Se han propuesto varios modelos de datos diferentes, los cuales pueden dividirse en tres grupos: Los Modelos Lógicos basados en objetos, en registros y los Modelos Fisicos de Datos.

1.5.1. MODELOS LOGICOS BASADOS EN OBJETOS

Los Modelos Lógicos basados en objetos se utilizan para describir los datos en los niveles conceptual y de visión, se caracterizan por el hecho de que permiten una estructuración bastante flexible y hacen posible especificar claramente las limitantes de los datos. Algunos de los más conocidos son:

-El Modelo Entidad-Relación.

-El Modelo Binario.

-El Modelo Semántico de Datos.

-El Modelo Infológico.

El Modelo de Datos Entidad-Relación se puede tomar como representativo de la clase de modelos lógicos basados en objetos, porque éste ha tenido bastante aceptación como modelo de datos apropiado para el diseño de Bases de Datos y porque se utiliza ampliamente en la práctica.

El Modelo de Datos Entidad-Relación (E-R), se basa en una percepción de un mundo real que consiste en un conjunto de objetos básicos llamados entidades y de las relaciones entre estos objetos. Una entidad es un objeto que existe y puede distinguirse de otros. La distinción se logra asociando a cada entidad un conjunto de atributos que describen al objeto.

La estructura lógica general de una base de datos puede expresarse gráficamente por medio de un Diagrama E-R que consta de los siguientes componentes:

-Rectángulos, que representan conjuntos de entidades.

-Elipses, que representan atributos.

-Rombos, que representan relaciones entre conjuntos de entidades.

-Líneas, que conectan los atributos a los conjuntos de entidades y los conjuntos de entidades a las relaciones.

1.5.2. *MODELOS LOGICOS BASADOS EN REGISTROS*

Los Modelos Lógicos basados en registros se utilizan para describir los datos en los niveles conceptual y de visión. A diferencia de los Modelos de Datos basados en objetos, éstos modelos sirven para especificar tanto la estructura lógica general de la Base de Datos como una descripción en un nivel más alto de la implantación, sin embargo, no permiten especificar en forma clara las limitantes de los datos.

Los tres Modelos de Datos que han tenido la más amplia aceptación son los siguientes:

MODELO RELACIONAL:

Modelo en el que la Base de Datos está constituída por un conjunto de tablas planas o relaciones, en el cual éstas se expresan por el hecho de que dos relaciones tengan un campo o dominio en común y en el que las relaciones pueden ser 1:N o M:N . Cada una de las tablas tiene varias columnas con nombres únicos.

MODELO DE RED:

Los datos en el Modelo de Red se representan por medio de conjuntos de registros y las relaciones entre los datos se representan con ligas, que pueden considerarse como apuntadores. Los registros de la Base de Datos se organizan en forma de conjuntos de gráficas arbitrarias. En éste modelo cualquier tipo de registro puede estar relacionado ya sea como sucesor o antecesor con cualquier cantidad de otros tipos de registros; para cada ocurrencia del registro antecesor pueden existir una o más ocurrencias relacionadas del registro sucesor.

MODELO JERARQUICO:

El Modelo Jerárquico es similar al Modelo de Red en cuanto a que los datos y las relaciones entre los datos se representan por medio de registros y ligas, respectivamente. El Modelo Jerárquico difiere del de Red en que los registros están organizados como conjuntos de árboles en vez de gráficas arbitrarias. En este Modelo un tipo de registro antecesor puede tener uno o más tipos de registros sucesores, pero no se permite a un tipo de registro sucesor tener más de un tipo de registro antecesor. El término Modelo Arbóreo de Datos es sinónimo de Modelo Jerárquico.

1.5.3. MODELOS FISICOS DE DATOS

Los Modelos Físicos de Datos sirven para describir los datos en el nivel más bajo. A diferencia de los Modelos Lógicos de Datos, son muy pocos los Modelos Físicos utilizados. Algunos de los más conocidos son:

-El Modelo Unificador.
-El Modelo de Cuadros.

CAPITULO 2.- LA INFORMACION: ANALISIS Y NATURALEZA

2.1.- DIFERENTES PUNTOS DE VISTA

Existen diferentes conceptos de utilización de una Base de Datos, por lo mismo, existen diferentes puntos de vista en lo que respecta a las personas que la utilizan. Típicamente, los sistemas de Bases de Datos contemplan estas circunstancias, sin embargo, es bueno especificar cuáles son estas diferentes formas de ver los datos.

2.1.1. DEL USUARIO.

La visión que el usuario tiene de la base de datos, debe adoptar, muy a menudo, la forma que a éste más convenga, es decir, un usuario no debe preocuparse por cómo están almacenados físicamente los datos, o de cómo el manejador de Base de Datos con el que trabaja realiza las operaciones que le permiten visualizar su información, mas aún, por razones de seguridad, el usuario de una Base de Datos no tiene por qué conocer la totalidad de la información que ésta contiene, sino limitarse al conjunto de datos con los que él trabaja.

Aquí se hará una pausa para identificar a dos tipos de los llamados *usuarios* de la base de datos. Existen, por una parte, los *usuarios en sí*, que regularmente son personas que muy poco conocen acerca de procesamiento computarizado de datos, por lo que su visión de la Base de Datos es extremadamente corta, limitándose a los programas de aplicación que él maneja y de los cuales normalmente desconoce su funcionamiento.

Este tipo de usuario es también llamado *usuario terminal*, debido a que es el encargado de aplicar u operar los programas de aplicación.

El otro tipo es el de los usuarios expertos (léase programadores de aplicaciones), para los cuales se extiende un poco la visión de los datos, en razón a sus conocimientos de programación, sin embargo, esta visión se reduce, por razones de control o, como ya dijimos, de seguridad, a una porción de la información contenida en la base, a través de lo que se conoce como *subesquemas*.

Este usuario además conoce la descripción lógica de los datos que están contenidos en su porción de Base de datos, es decir, la relación entre los diferentes campos, su longitud y formato, etc.

2.1.2. DEL ADMINISTRADOR.

Debido a que el administrador de una Base de Datos es el encargado de supervisar y mantener la vista lógica global de los datos, tiene una visión amplia en lo que a la estructura general de la Base de Datos se refiere, para poder modificarla *lógicamente*.

Esto no significa que el administrador sabe qué contienen los registros. Sabe que existe un registro llamado XXXXX y que éste contiene un campo llamado YYYYY, sin embargo, no sabe cuál es el valor de este campo. No obstante, si se requiere, digamos, ampliar un campo de 6 a 7 dígitos, es tarea del administrador hacerlo, y debe conocer los procedimientos para hacerlo.

Además, el administrador está de tal manera vinculado con la estructura lógica de la Base de Datos, que es el que mejor conoce las utilerías de el manejador de Base de Datos que están orientadas a la conservación de la integridad de la misma (tales como procesos de reorganización de datos, análisis de errores, expansión o compresión del espacio físico utilizado, etc.).

Cabe mencionar que tampoco conoce cómo es que están organizados físicamente los datos, sin embargo, utiliza herramientas que están ligadas con ésta visión. A la visión que ésta persona o grupo de personas (normalmente es un grupo, ya que sus funciones son muchas y de diversa índole) maneja se le llama *esquema*.

2.1.3. DEL DISEÑADOR.

La visión que el diseñador de la Base de Datos tiene de ésta, es mucho mas amplia que la de los administradores o usuarios, ya que sus propias funciones implican que no sólo conozca los datos, sino también la forma en que éstos son almacenados, precisa conocer no sólo los elementos de hardware que intervienen en los procesos, sino que debe conocer a fondo cuál es la relación, tanto física como lógica, que guardan con la Base de datos, es pues, quien tiene una visión periférica de todo lo que sucede en torno a la Base de Datos, su organización de archivos, las estructuras de datos manejadas, etc.

Dadas estas características, el diseñador de la Base de Datos está preocupado por la vista *física* de la base de datos. Estructuras de datos usadas, algorítmos de almacenamiento, métodos de acceso a discos, cintas. etc. y es, por tanto, quien debe conocer más a fondo la estructura lógica *interna* de la Base.

Todo lo dicho anteriormente queda reducido a la figura 2.1.

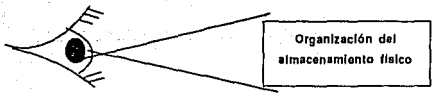
2.2.-POLITICA DE MANEJO DE DATOS.

Dados los diferentes puntos de vista que tienen las personas que hacen uso de la Base de Datos, se desprende la necesidad de normar o formalizar políticas de manejo de datos que aseguren la integridad de los mismos, y que comprometan a los diferentes usuarios a desempeñar su trabajo de manera tal que no se perjudique a ninguno de los demás partícipes, tanto en su información, como en el tiempo de utilización de la Base de Datos que les haya sido otorgado.

Para facilitar ésta labor, se asigna a los usuarios cierto tipo de funciones de acuerdo a sus carcterísticas dentro de la estructura orgánica de las instalaciones donde resida la Base de Datos

2.2.1. FUNCIONES OPERATIVAS.

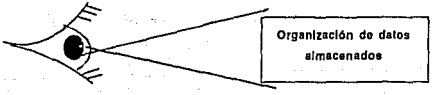
Se distingue como funciones operativas todas aquellas que están relacionadas con el manejo de la información. Estas funciones están a cargo típicamente de usuarios finales (capturistas, operadores, etc.) y es responsabilidad de los mismos el que todas las operaciones de este tipo se lleven a cabo satisfactoriamente.



representada por la distribución física de los datos

1.Vista de los diseñadores de base de datos que se interesan por el rendimiento.

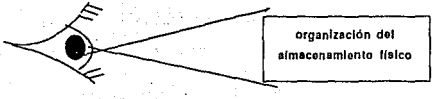
Conversión realizada por el software de administración de datos



representada por la distribución física de los datos

2.Vista del administrador de la base de datos (vista global de los datos)

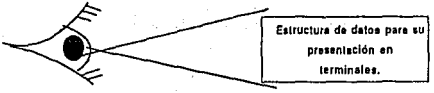
Conversión realizada por el software de administración de datos



representada por la distribución física de los datos

3.Vista de los programadores de aplicaciones y de los programas de aplicación.

Conversión realizada por el software de administración de datos



representada por la distribución física de los datos

4. Vista del usuario final

Conversión realizada por el software de administración de datos

Fig. 2.1 Diferentes puntos de vista.

Son ejemplo de funciones operativas las siguientes.

- Altas de información en la Base de datos.

- Modificaciones, siempre a la información (bajas, cambios, etc.)

- Realización de Programas de aplicación.

- Realización de respaldos de la información propia.

Como normas a seguir por los individuos encargados de la funciones operativas, se distinguen las siguientes.

- Conocer la estructura de comandos del sistema operativo bajo el cual está instalada la Base de Datos.

- Conocer la estructura operativa de los programas de aplicación. (dónde residen, qué archivos necesitan para su ejecución, etc.)

- Ejecutar los pasos indicados en los manuales de cada aplicación.

- Hacer uso sólo de las apicaciones que les hayan sido asignadas.

- Hacer uso sólo del equipo que les haya sido asignado.

- Comunicar al administrador de la Base de Datos, de cualquier anomalía que exista, tanto en información, como en la ejecución de cualquier programa.

Como puede observarse, las políticas para los usuarios que tienen a su cargo las funciones operativas se refieren específicamente al manejo de los datos.

Es necesario aclarar, que las obligaciones y características antes listadas, varían de instalación en instalación y, definitivamente, siempre están en relación con el grado de seguridad que se quiere implementar para cada Base de Datos en particular, puesto que no es lo mismo operar una base de datos de escolares, que la base de datos de la nómina de los funcionarios del gobierno, o la de armamento nuclear de la N.A.S.A., es por ello que se definen éstas políticas específicamente para la aplicación de que se trate, y son tan importantes como la información misma y deben hacerse observar, siempre que se quiera asegurar la integridad de los datos.

2.2.2. FUNCIONES DE CONTROL.

Se distinguen como funciones de control, todas aquellas que están encaminadas a controlar el estado que guarda la base de datos dentro del equipo, así como las encaminadas a asegurar que los recursos de la Base de Datos sean aprovechados óptimamente, esto es, con una alta seguridad en cada una de las transacciones realizadas.

Es claro que tiene mucho que ver con las funciones del Administrador, sin embargo, son también partícipes de las funciones de control los diseñadores, programadores de aplicaciones y operadores del equipo. Quiza esto se comprenda mejor después de ver los ejemplos de funciones de control. Ejemplos de éstas son los siguientes.

- Diseño de Bases de Datos.

- Diseño de algoritmos de almacenamiento y búsqueda.

- Manejo de utilerías de auditoría a la Base de Datos.

- Puesta en línea de las mismas.

- Asignación o restricción de derecho a un usuario sobre los espacios de la Base de Datos.

- Generación de utilerías tendientes a conservar la consitencia de los datos.

- Utilerías de transporte de Bases de Datos.

- Implantación de aplicaciones.

- Soporte técnico.

Son ejemplos de normas a seguir por los individuos encargados de las funciones de control las siguientes :

- Conocer la arquitectura del equipo utilizado.

- Conocer la descripción física y lógica de lo datos, así como las relaciones entre los mismos.

- Conocer el manejo de las utilerías de auditoría.

- Hacer respetar las normas de control.

- etc.

Es notorio que las necesidades, tanto de herramientas de software, como de equipo, son diferentes para estos dos tipos de individuos, puesto que, mientras un usuario solo necesitará de su terminal y tendrá que hacer requerimientos de otros elementos de hardware, el administrador ó el diseñador tendrán acceso a casi todos los elementos de hardware o de software.

Cabe destacar que aunque todas las funciones referentes al manejo de Base de Datos son importantes, muchas veces se tiene que dar prioridad a funciones de control, en aras de evitar inconsistencias en la información. Como ejemplo podemos citar el caso de que se tenga un error en las ligas (relaciones entre registros) . En este caso, es prioritario corregir este error, antes que continuar con la captura de datos, pues de no ser así pueden presentarse problemas mayores.

2.3. DIAGRAMA DE FLUJO DE DATOS.

Los diagramas de flujo de datos, muestran gráficamente la relación entre los procesos y los datos, es decir, todos los componentes esenciales del sistema y cómo se relacionan entre sí.

Por lo tanto, los diagramas de flujo de datos ayudan a ilustrar los componentes esenciales de un proceso y la forma en la que interactuan.

En la figura 2.2 se muestran cuatro sencillas notaciones, con las cuales pueden completarse los diagramas de flujo de datos.

En el diagrama de flujo de datos cada componente se etiqueta con un nombre descriptivo. Los nombres del proceso se identifican con un número que se utilizará para propósitos de identificación. (Este número no representa la secuencia).

La figura 2.3 muestra la notación en un diagrama de flujo de datos sencillo, que consiste de cinco flujos de datos, dos procesos, un almacenamiento de datos, un origen y un destino.

Una ventaja de los diagramas de flujo de datos es la característica de mostrar *actividades paralelas*. Nótese que en la figura 2.3 se están realizando en forma simultánea los flujos de datos uno y dos.

Los diagramas de flujo de datos se concentran en los *datos* que se mueven a través del sistema y no en los dispositivos o el equipo.

Los diagramas de flujo también muestran cuándo entran o se retiran los datos del área de aplicación, por qué se retiran o se introducen y qué proceso se está llevando a cabo.

NOTACION

1. FLUJO DE DATOS.

LOS DATOS CAMBIAN EN UNA DIRECCION ESPECIFICA, DESDE SU ORIGEN HASTA SU DESTINO, EN FORMA DE UN DOCUMENTO, CARTA LLAMADA TELEFONICA U OTRO MEDIO. EL FLUJO ES EN DE "PAQUETES DE DATOS".

2. PROCESOS.

EL PERSONAL, PROCEDIMIENTOS O DISPOSITIVOS UTILIZAN O PRODUCEN (TRANSFORMAN) DATOS, NO SE IDENTIFICA EL COMPONENTE FISICO.

3. ORIGEN Y DESTINO DE LOS DATOS.

EL ORIGEN EXTERNO O DESTINO DE LOS DATOS QUE PUEDEN SER INDIVIDUOS, PROGRAMAS, EMPRESAS U OTRAS ENTIDADES, INTERACTUAN CON EL SISTEMA PERO PERMANECEN O ESTAN FUERA DE SU LIMITE.

4. DATOS ALMACENADOS.

AQUI LOS DATOS SE ALMACENAN O SE HACE REFERENCIA A ELLOS A TRAVEZ DE UN PROCESO DEL SISTEMA. PUEDE O NO REPRESENTAR DISPOSITIVOS DE COMPUTADORA.

FIG. 2.2. NOTACION PARA DIAGRAMAS DE FLUJO DE DATOS

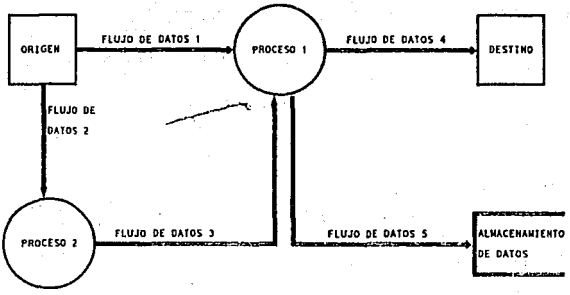


FIG.2.3. DIAGRAMA DE FLUJO DE DATOS

Las notaciones mostradas en la figura 2.2 son muy sencillas y fácil de entender por los usuarios, de tal manera que, los analistas pueden hacer que los usuarios participen en el estudio del flujo de datos. Los usuarios pueden sugerir modificaciones o examinar los diagramas y señalar los problemas rápidamente, de modo que se puedan corregir antes de que se inicie otro trabajo de diseño.

Esto es muy importante, ya que los problemas que no se localizan anticipadamente en el proceso de desarrollo, serán muy difíciles de corregir posteriormente, incluso evitarlos a tiempo puede prevenir una falla del sistema.

Los diagramas de flujo de datos muestran las características *lógicas* de las aplicaciones; es decir, señalan *qué* ocurre y *cuándo*, pero no establecen *cómo*, ya que los detalles *físicos* son métodos de almacenaje, dispositivos de entrada o componentes de la computadora, y no son aspectos principales para los analistas durante la parte inicial del análisis.

2.3.1. DESCOMPOSICION DE LOS DIAGRAMAS DE FLUJO DE DATOS.

Una investigación de sistemas elaborada en forma amplia produce conjuntos de muchos diagramas de flujo de datos. Algunos de estos diagramas proporcionan una visión general de los procesos principales y otros dan un mayor detalle para señalar elementos de datos, almacenamiento de éstos y etapas del proceso.

Si los analistas desean revisar el sistema en su totalidad, utilizan los diagramas generales, sin embargo, si les interesa estudiar un proceso en particular, utilizan los diagramas de flujo de datos de los procesos de menor nivel.

Por lo tanto, los diagramas de flujo de datos se desarrollan y utilizan de manera progresiva, que va de lo general a lo particular.

Para entender mejor la descomposición de los diagramas de flujo de datos se seguirá un ejemplo en donde se tratan los procesos desde un nivel general a un nivel de detalle cada vez mayor. (Este proceso se conoce como análisis *descendente*).

Supóngase que un analista que inicia un trabajo sobre un sistema de cuentas por pagar está interesado en conocer los elementos con los que trabaja el sistema (entradas) y lo que produce (salidas).

El diagrama de flujo de datos de la figura 2.4 describe el proceso de cuentas por pagar a nivel muy general. La información contenida en la figura 2.4 es inadecuada para entender los requerimientos de sistemas. Por lo tanto, el siguiente paso es describir el proceso de cuentas por pagar con mayor detalle. La figura 2.5 muestra el proceso de cuentas por pagar incluyendo tres subprocesos : APROBACION DE FACTURAS, REVISION DEL SALDO VENCIDO Y ESCRITURA DE CHEQUES.

Este paso de descripción de lo general a lo particular (descendente) se repite muchas veces en una investigación de sistemas.

Los diagramas de flujo de datos no serán útiles si son difíciles de entender, ésto quiere decir, que cada nivel inferior debe definirse utilizando de tres a siete procesos para extender un proceso de mayor nivel. Utilizar más de siete provoca que el diagrama sea difícil de manejar.

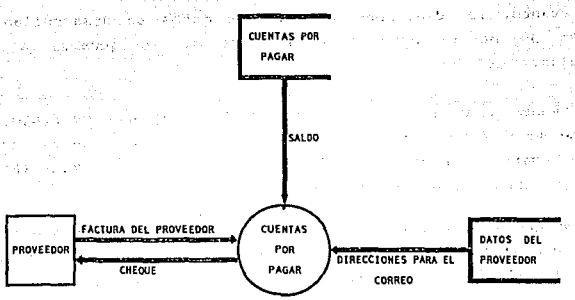


FIG. 2.4. DIAGRAMA GENERAL DEL FLUJO DE DATOS DEL SISTEMA DE CUENTAS POR PAGAR.

Los diagramas de flujo de datos son más fáciles de leer si la descripción de un proceso de un nivel puede dibujarse en una sola hoja de papel. Si se necesitan más datos, el siguiente nivel puede extenderse.

Es muy importante mantener la consistencia entre los procesos al hacer la extensión. Por ejemplo, nótese que en la figura 2.5, el primer proceso tiene la misma entrada (factura del vendedor) y la misma salida (escritura de cheque) que la figura 2.4; por lo tanto, la extensión es consistente.

Hasta este momento, los diagramas mostrados no incluyen información de control. Aunque ésta información es necesaria al final, no debe importar mientras se está identificando el flujo de datos en general.

Los diagramas secundarios (bajo el segundo o tercer nivel) muestran un manejo de errores y de excepciones en el proceso que se extiende.

La figura 2.6 es el diagrama secundario de flujo de datos para el proceso APROBACION DE FACTURA de la figura 2.5.

De igual forma, las figura 2.7 y 2.8 muestran el diagrama secundario de datos para los procesos REVISION DEL SALDO VENCIDO Y ESCRITURA DE CHEQUES de la figura 2.5 respectivamente.

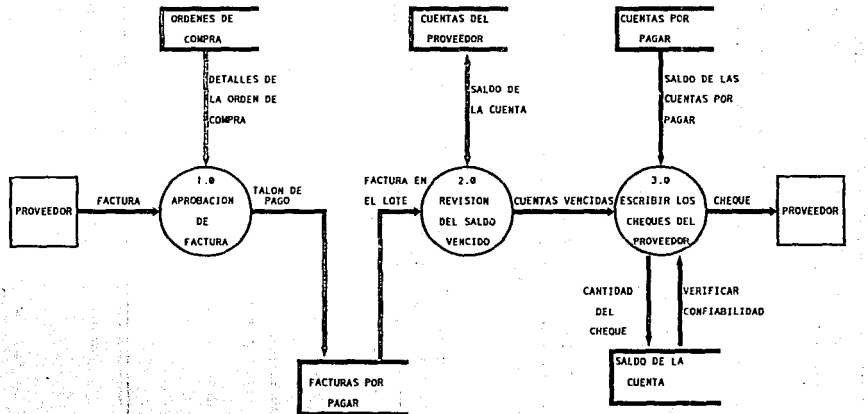


FIG. 2.5. DIAGRAMA DE FLUJO DEL PRIMER NIVEL PARA EL PROCESO DE CUENTAS POR PAGAR.

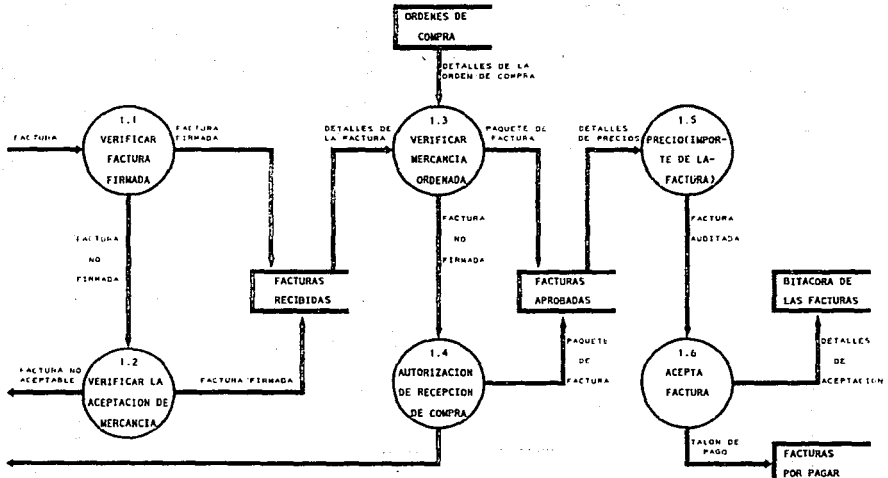


FIG. 2.6. DIAGRAMA DE FLUJO DE DATOS DE SEGUNDO NIVEL PARA EL PROCESO DE APROBACION DE FACTURAS.

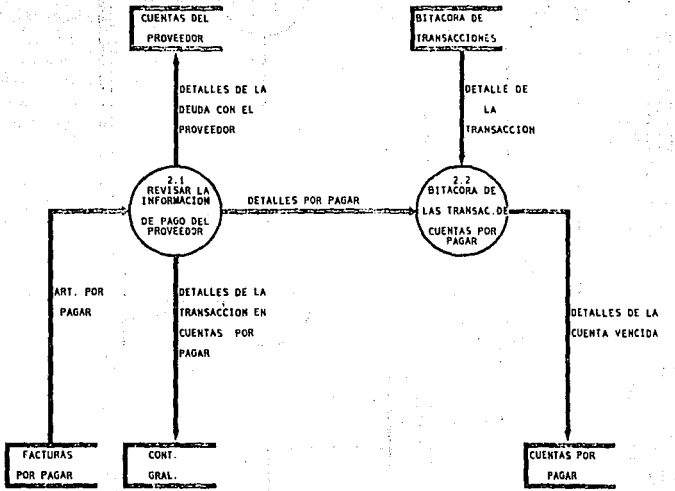


FIG. 2.7. DIAGRAMA DE FLUJO DE DATOS DE SEGUNDO NIVEL PARA EL MANTENIMIENTO DE SALDOS DE PROVEEDORES.

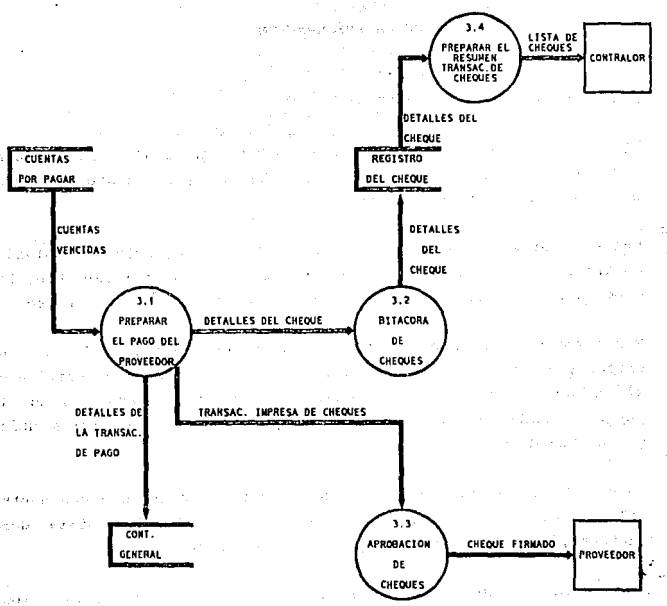


FIG.2.8. DIAGRAMA DE FLUJO DE DATOS DE SEGUNDO NIVEL PARA EL PROCESO DE PAGO A VENDEDORES.

2.4.- CARACTERISTICAS DE LA INFORMACION.

2.4.1. *ENTIDADES Y SUS ATRIBUTOS*

Llamaremos entidades a las cosas sobre las cuales se almacena información. Una entidad es un objeto que existe y puede distinguirse de otros objetos.

Una entidad puede ser un objeto tangible, por ejemplo un empleado, un artículo o un lugar, pero también puede ser algo intangible, tal como un suceso, la cuenta de un cliente o un concepto abstracto.

Por ejemplo, Luis Méndez, con número de empleado 2740478, es una entidad, ya que identifica en forma única una persona específica en el universo. De la misma manera, la cuenta número 115 en la sucursal Centro es una entidad, ya que identifica en forma única una cuenta determinada.

Es decir, una entidad es una persona, cosa o lugar, que cae dentro del alcance del sistema, acerca de la cual el sistema debe mantener, correlacionar y desplegar información.

Un *conjunto de entidades* es un grupo de entidades del mismo tipo. Por ejemplo, todas las personas que tienen una cuenta en el banco, pueden definirse como el conjunto de entidades *cuentahabiente*. De igual manera, todas las cuentas en un banco determinado podrían ser el conjunto de entidades *cuenta*.

No es necesario que las entidades pertenezcan exclusivamente a un solo conjunto. Por ejemplo se pueden definir las siguientes entidades:

-El conjunto de entidades de todos los empleados de un banco (*empleado*).

-El conjunto de entidades de todos los cuentahabientes del banco (*cuentahabiente*).

Entonces, una entidad *persona* puede ser una entidad *empleado*, una entidad *cuentahabiente*, ambas, o ninguna de las dos.

Toda entidad tiene propiedades que eventualmente conviene registrar como color, valor monetario o nombre, a éstas propiedades se les llama *atributos*.

Así pues, un atributo es una característica o cualidad de una entidad, que cae dentro del alcance del sistema, acerca del cual el sistema debe mantener, correlacionar y desplegar información.

Los atributos se modelan como columnas de la entidad. Por ejemplo, los posibles atributos del conjunto de entidades EMPLEADOS se muestran en la figura 2.9.

ENTIDAD: EMPLEADOS

A T R I B U T O S

R E G I S T R O S

Num_emp	Apellido	Puesto	Salario	Num_dep	Proyecto
7329	ALCANTARA M.	ANALISTA	1.000,000	10	NOMINA
7499	GIRON T.	DBA	1,500,000	20	INVENTARIO
7521	CONTRERAS P.	LIDER	2.000,000	20	INVENTARIO
7566	REYES A.	GERENTE	3.000,000	10	----------
7654	ALANIS R.	ANALISTA	1.000,000	20	INVENTARIO
7782	PALOMARES E.	LIDER	2.000,000	10	NOMINA
7934	ESCALANTE R.	DBA	1.500.000	10	NOMINA
7982	MARENTES A.	ANALISTA	1.000,000	20	INVENTARIO

FIG. 2.9.

2.4.2. *TRES CAMPOS.*

Cuando hablamos de información podemos referirla a tres diferentes campos:

1.-El de el mundo real, en el que existen entidades y las entidades exhiben ciertas propiedades.

2.-El dominio de las ideas y la información existente en las mentes de las personas y los programadores. Aquí hablamos de los atributos de las entidades y nos referimos a estos simbólicamente. Es decir, asignamos valores a los atributos.

3.-El tercer campo es el de los datos, en el que usamos caracteres o bytes para codificar items de información.

2.4.3. *ARCHIVOS PLANOS.*

La manera más común de asociar un valor con un ítem de datos y de asociar ítems de datos con atributos de entidad, consiste en almacenar juntos los ítems de datos en una secuencia fija, como lo ejemplifica la figura 2.9.

Dentro del recuadro de la fig. 2.9 aparece un conjunto de ítems de datos. Cada fila de ítems de datos se refiere a una entidad en particular y cada columna contiene un tipo particular de ítem de datos.
A esta forma de representar atributos y entidades también se les conoce como TABLA.

Una tabla es un arreglo de dos dimensiones, compuesta de renglones y columnas; cada columna debe tener un nombre de columna y este nombre debe ser único dentro de la tabla.

Los atributos se modelan como columnas de la entidad. En la parte superior de la fig. 2.9, fuera del recuadro, se anotan los nombres de los atributos.

La primera columna de la izquierda contiene los ítems de datos que identifican las entidades, a ésta columna se le llama también **llave primaria** (PK) e identifica en forma única a cada renglón de la tabla.

El atributo que se considera en este caso como identificador o llave primaria es Num_emp (número de empleado).

Debe considerarse que:

-No se permiten valores nulos (faltantes) en una llave primaria.

-Una llave primaria no puede tener valores duplicados.

-No se permiten cambios sobre los valores de la llave primaria.

-Una llave primaria puede constar de mas de una columna y en ese caso se le llama **llave primaria compuesta.**

Los nombres de los atributos y las representaciones de valor no se registran en el archivo, aunque deben hallarse registrados en alguna parte, por ejemplo en un Diccionario de Datos en que se alistan los nombres y los tipos de todos los ítems de datos de la base.

Obsérvese que algunos de los valores de los atributos pueden ser nombres o identificadores de entidades en otros archivos o tablas, es decir, es una columna (o grupo de columnas) que es llave primaria en alguna otra parte y entonces se le llama llave foránea o clave secundaria.

Por ejemplo, Num_dep (número de departamento) en la figura 2.9, es un atributo de la entidad EMPLEADO; en algún otro lugar puede existir un archivo DEPARTAMENTO que da los valores de atributo de cada departamento de la empresa.
Cada grupo de datos que forma una fila horizontal en la figura 2.9 puede constituír un registro de entidad.

La simple distribución bidimensional de elementos de datos en la figura 2.9 se llama a menudo también disposición plana. Se habla así de archivos planos y algunas clases de organización de archivos se preveén para este tipo de archivo.

2.5.- DICCIONARIO DE DATOS Y SUS CARACTERISTICAS.

Un diccionario de datos es una ayuda para identificar y clasificar los datos almacenados en la base de datos. Consiste de archivos, registros y campos que contienen información descriptiva de los archivos, registros y campos de la base de datos.

El Diccionario de Datos es una librería central para definir el significado, uso, características y otros datos relevantes de todos los ítems de datos, campos, entidades, sinónimos, referencias cruzadas y las relaciones que existen entre ellos.

Así pues, el Diccionario de Datos no especifica los valores actuales de los datos, sino que define el tipo de valor que debe ir en cada campo.

Los Sistemas Manejadores de Bases de Datos (DBMS) tienen muchos de los elementos de un Diccionario de Datos (DD), pero no están diseñados para manejar las especificaciones de los datos.

Es importante que al introducir nuevas aplicaciones al sistema y nuevos datos, el diseño de la base de datos se modifique para reflejar estas nuevas entradas; El DBMS no identificará la magnitud del cambio, mientras que el Diccionario de Datos podrá ayudarnos a determinar el impacto de los cambios.

Además podrá ser una herramienta de documentación automática excelente, auxiliando al Administrador de la Base de Datos (DBA) en la realización de sus funciones.

Resumiendo: un Diccionario de Datos es una Base de Datos que contiene datos acerca de los datos de la Base de Datos.

Los Diccionarios de datos pueden ser de dos tipos: Integrado y Standalone. A continuación se mencionan las características de cada uno.

INTEGRADO.

*Desarrollado y comercializado por el vendedor del DBMS.

*Puede trabajar con otros DBMS pero ésto provocaría que perdiera algunas de sus ventajas.

*Es recomendable trabajar con el DBMS del vendedor pues de ésta manera se logra un mayor control sobre los datos y reforzamiento de los estándares.

*El hecho de que éstos diccionarios estén integrados, provoca que los cambios a la base de datos se reflejen en forma automática en el Dicionario de Datos.

STANDALONE.

*No obliga al uso de un DBMS en particular.

*Trabaja con varios DBMS.

*Permite una documentación gradual.

*Requiere un proceso separado para ser actualizado.

*Si el proceso de actualización no se realiza, entonces habrá inconsistencia.

Como principales características del el Diccionario de Datos se tiene que :

*Provee una interfaz conveniente para una o mas Bases de Datos.

*Permite acceder la información de los datos a personal no informático.

*Soporta diversas variedades de estructuras físicas.

*Libera al usuario de tareas repetitivas y sencillas, por ejemplo: generación de reportes.

*Permite almacenar toda la información relevante sobre los datos.

Otra definición del diccionario de datos (relacionándolo con los diagramas de flujo de datos) es la siguiente:

"Un diccionario de datos es una lista de todos los elementos incluídos en el conjunto de los diagramas de flujo de datos que describen un sistema".

A menudo, el diccionario de datos es la única fuente común de definiciones para los usuarios. Se utiliza como la única fuente de respuestas a todas las preguntas que se relacionan con el formato y el contenido de los conjuntos de datos utilizados en el sistema.

El diccionario de datos puede revelar información como la que sigue:

- **Listado de atributos y entidades.** Todos los datos utilizados en el sistema y que incluyen nombre, descripción, longitud de campo y nombres alternos (alias).

- **Listado del proceso.** Todos los procesos que se llevan a cabo en el sistema, además de una descripción de las actividades asociadas con cada uno de ellos.

- **Verificación de referencia cruzada.** Determinación de dónde se utilizan los datos en el sistema, por ejemplo: ¿qué procesos utilizan los mismos datos?, ¿qué datos se utilizan?.

- **Detección de errores.** Encontrar inconsistencias, como datos necesarios de un proceso que nunca se introducen al sistemna, procesos que no se alimentan con flujos de datos internos o que no producen flujos de datos como salida.

El diccionario de datos lo utilizan los analistas para entender el sistema y consultar los detalles y descripciones que almacenan durante el diseño de sistemas, cuando la información sobre los mismos, como longitud de datos, nombres alternos y uso de los datos en procesos específicos debe estar disponible.

El diccionario de datos también almacena información de validación para guiar a los analistas en la especificación de controles para la aceptación de datos por parte del sistema.

Los diccionarios de datos se pueden desarrollar manualmente o con procedimientos automatizados.

Aunque el propósito y función de los diccionarios de datos manuales y de los automatizados, es el mismo, los segundos son muy convenientes en sistemas grandes, ya que producirán un glosario de datos, listados cruzados e informes de errores conforme se requieran y ahorrarán una gran cantidad de tiempo comparado con los sistemas manuales.

Sin importar la forma, los diccionarios de datos son un aspecto esencial del análisis de flujo de datos y determinación de requerimientos.

CAPITULO 3 .- ENFOQUE RELACIONAL, EL MODELO ACTUAL.

3.1.- FUNDAMENTACION TEORICA DEL MODELO RELACIONAL.

Un sistema manejador de Bases de Datos debe ser capaz de representar y manipular entidades (registros o segmentos) y sus relaciones de una manera fácil y conveniente. En el enfoque jerárquico se representa la relación entre dos segmentos por la posición relativa de abajo hacia arriba y de izquierda a derecha que guardan entre sí los elementos. En el enfoque de red, las relaciones se representan mediante apuntadores, los cuales enlazan un tipo de registro propietario con uno miembro. Para el caso de Bases de Datos grandes el manejo de los datos y su estructura lógica puede ser demasiado complejo para los programadores y usuarios casuales, pues es necesario conocer jerarquías y rutas de acceso. De esta manera, puede ocurrir que muchos cambios en la Base de Datos violen la independencia de los datos y afecten a los programas de aplicación. El grado de complejidad de muchas Bases de Datos suele crecer a medida que crecen sus aplicaciones, con lo que estos sistemas se transforman en una maraña de datos e interrelaciones.

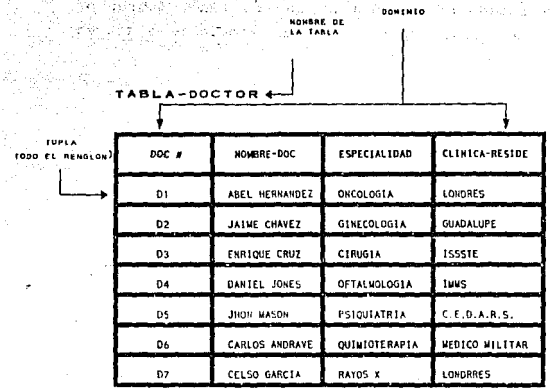
El enfoque relacional concebido por E.F. Codd y que continua evolucionando dede 1970, tiende a reducir los inconvenientes antes planteados, y está basado en la idea de que se puede lograr de cualquier conjunto de datos, *representaciones tabulares* a través de la *normalización*.

3.2 ESTRUCTURA DE BASES DE DATOS RELACIONALES

3.2.1. REPRESENTACION TABULAR.

Una de las maneras en que más fácilmente se pueden representar datos para el usuario, consiste en usar tablas bidimensionales, tales como las de la Fig. 3.1. El usuario está familiarizado con ésta forma de representacion gráfica y la comprende, visualiza y recuerda sin dificultad. Para que las tablas en cuestión cumplan con el objetivo de facilitar las operaciones de usuario, deben ser matrices rectangulares que puedan ser descritas matemáticamente, y que posean las siguientes propiedades generales.

- Cada entrada de las tablas debe representar un item de datos; no hay grupos repetitivos.
- Todos los items de una columna son de la misma clase.
- Cada columna tiene un nombre propio
- Todas las filas son diferentes, no se admiten filas duplicadas.
- Tanto las filas como las columnas pueden ser consideradas en cualquier secuencia y en cualquier momento sin afectar con ello el contenido de la información ni la semántica de cualquier función que utilize la tabla.



DOC #	NOMBRE-DOC	ESPECIALIDAD	CLINICA-RESIDE
D1	ABEL HERNANDEZ	ONCOLOGIA	LONDRES
D2	JAIME CHAVEZ	GINECOLOGIA	GUADALUPE
D3	ENRIQUE CRUZ	CIRUGIA	ISSSTE
D4	DANIEL JONES	OFTALMOLOGIA	IMMS
D5	JHON MASON	PSIQUIATRIA	C.E.D.A.R.S.
D6	CARLOS ANDRAYE	QUIMIOTERAPIA	MEDICO MILITAR
D7	CELSO GARCIA	RAYOS X	LONDRRES

FIG. 3.1 TABLA PLANA O RELACION.

Así, una tabla como la de la Fig.3.1 se llama *relación*. Las Bases de Datos construídas por estas tablas son Bases de Datos *relacionales*, en consecuencia, " *una base de datos se dice ser relacional, cuando está compuesta por una serie de matrices (tablas) planas (sin grupos repetitivos) de ítems de datos*".

La relación o tabla es un conjunto de *tuplas*. Si se trata de n-tuplas, es decir , si la tabla tiene n-columnas, se dice que la relación es de grado n. El conjunto de valores de cada columna (la columna en sí) constituye un *dominio*.

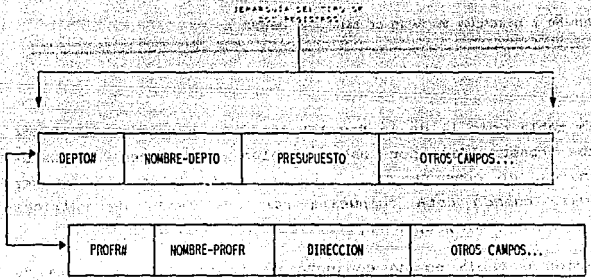
En lenguaje matemático diríamos que, dados n dominios D_1, D_2,...D_n (no necesariamente diferentes), R es una relación sobre estos dominios, si R es un conjunto de tuplas las cuales tiene su primer elemento del dominio D_1 su segundo elemento del dominio D_2 y así para todos los n dominios.

NORMALIZACION.

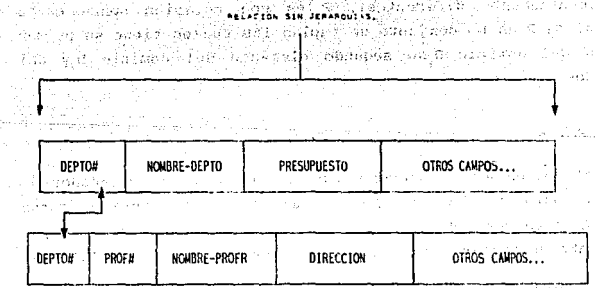
La normalización es el proceso mediante el cual un diseñador de bancos de datos pueden transformar cualquier estructura de datos no plana, a un conjunto de *relaciones normalizadas*, es decir, a un conjunto de tablas planas que no contengan grupos repetitivos.

Así como pueden transformarse las Bases de Datos de tipo red en jerárquicas mediante la introducción de cierta redundancia, se puede transformar cualquier tipo de base de datos a una de tipo relacional vía la introducción de redundancia adicional.

Se dice que una relación no plana o no normalizada contendrá al menos un dominio que será en realidad otra relación. Una relación normalizada tiene únicamente dominios simples, es decir, dominios que no son a su vez otra relación.



a) Base de datos sin normalizar.



b) Base de datos normalizada.

FIG. 3.2 CONVERSION MEDIANTE NORMALIZACION.

Un archivo que sea plano excepto por un grupo repetitivo, se normalizará al quitar el grupo repetitivo y formar con el una relación separada. Esta deberá tener un nombre propio así como un campo *llave*.

La Fig.3.2 muestra esquemáticamente cómo se realiza la *normalización* para una base de datos con dos tipos diferentes de registros. Note que el campo *llave* (del cual discutiremos en la sección posterior) DEPTO#, debe repetirse en la nueva relación PROFESOR y se combina con PROF# para conservar las características unívocas de cada una de las tuplas. Cabe mencionar aquí que el hecho de tener redundancia *lógica* de datos no necesariamente significa que haya redundancia física, pero significa necesariamente al menos cierto costo extra para evitar la redundancia física.

LLAVES.

Toda tupla debe estar asociada con una *llave* que permita su identificación . Esta llave o *campo llave*, no es más que uno de los atributos de las tuplas, que, para ser considerado como llave debe tener las siguientes características :

- *Identificación unívoca.* El valor de la llave en cada tupla debe ser diferente.

- *No redundancia.* Ningun valor de la llave debe ser descartado, pues esto acabaría con las características unívocas.

Cabe mencionar que ésta *llave* también es llamada *campo clave*, y que puede estar formado por un atributo solo o un conjunto de atributos, en el caso en que un sólo atributo de la base de datos no satisfaga los requisitos anteriormente citados.

Sin embargo, existen Bases de Datos en las que hay más de un conjunto de atributos que cumplan con las dos características citadas, a estos conjuntos de atributos (dominios) se les llama *llaves candidatas O claves candidatas* y una de ellas debe ser designada como llave o clave *primaria* usada para identificar efectivamente al registro o tupla, no está por demas decir que ésta clave o llave *primaria* debe ser escogida de manera tal que el conjunto de dominios que la forman sea mínimo (incluso solo un campo).

Hasta aquí hemos visto que existen equivalencias entre términos que son comunmente usados (tupla = registro, etc.), por lo que decidimos incluír una relación de los términos relacionales más usados y sus equivalencias (Fig. 3.3).

NOTACION.

Para obviar el tener que usar tablas como la de la figura 3.1, a partir de este momento se usarán notaciones como las siguientes:

CLIENTE (CLIENTE#, NOMBRE, DOMICILIO, CLI-DESDE, ULT-COMPRA)

VENTA (NO-ORDEN, NO-ARTICULO, PRECIO, CANTIDAD)

ORDEN-VENTA (NO-ORDEN, CLIENTE#, FECH-ORDEN, FECHA-ENTREGA, TOTAL-$)

El concepto que antecede al paréntesis es el *nombre de la relación*. Los nombres que aparecen dentro de los paréntesis son los *campos* (o dominios o columnas) de la relación. Los campos subrayados indican que se están usando como *llave primaria* para identificar las tuplas.

EQUIVALENCIAS	
CAMPO	ATRIBUTO, DATO ELEMENTAL
VALOR DE CAMPO	VALOR DE DATO ELEMENTAL VALOR DE ATRIBUTO
DOMINIO	COLUMNA,CONJUNTO VERTICAL DE CAMPOS
ESTRUCTURA DE RELACION	TIPO DE REGISTRO, ESTRUCTURA DE TABLA
TUPLA	REGISTRO, ENEADA, RENGLON, ENTIDAD
LLAVE	PARTE RECTORA
RELACION	ARCHIVO, TABLA, BASE-DATOS
GRADO DE UNA RELACION	NUMERO DE CAMPOS, NUMERO DE ELEMENTOS

FIG. 3.3 TERMINOLOGIA RELACIONAL EQUIVALENTE.

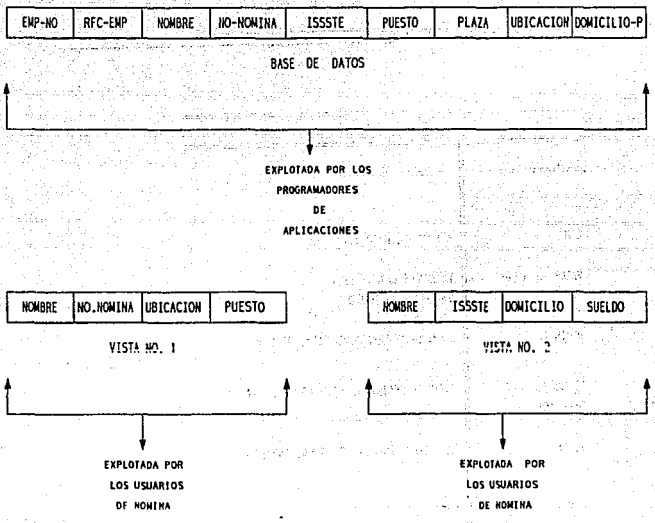


FIG.3.4. ARQUITECTURA DE BASES DE DATOS RELACIONES.

3.3 ARQUITECTURA DEL MODELO RELACIONAL

La Fig. 3.4 muestra la arquitectura relacional que se propone. La base de datos global es un conjunto de relaciones al que se hace referencia como : *modelo relacional de datos, relaciones almacenadas, relaciones base o base de datos relacional.* El término *esquema* no se utiliza muy frecuentemente. El modelo de datos lo define el administrador del sistema (DBA) mediante un *lenguaje de descripción del modelo relacional de datos.*

Al modelo particular de datos de un usuario, que se extrae del modelo de datos global, se le llama *submodelo relacional de datos, vista, o subesquema.* El término *subesquema* no se usa muy generalmente (se usa cada vez mas el término *vista).*

Al igual que el modelo relacional de datos, este submodelo es definido por el DBA, y es invocado por el usuario a través de un *lenguaje de descripción del submodelo relacional de datos.*

El submodelo relacional de datos es una colección de relaciones que pueden derivarse de las del modelo global mediante ciertas *operaciones relacionales.*

El lenguaje de descripción del submodelo de datos incluirá las operaciones y mecanismos relacionales necesarios para formar submodelos permisibles a partir del modelo. De hecho, el lenguaje de descripción de submodelos es básicamente el mismo sublenguaje de datos, con ciertas adiciones para definición de datos.

Puede definirse un número arbitrario de submodelos sobre un modelo de datos dado. Un submodelo podrá ser el modelo de datos completo. Un número arbitrario de usuarios puede compartir un modelo de datos dado vía los submodelos. Los submodelos pueden traslaparse.

Cada usuario posee uno o más espacios o *áreas de trabajo* para contener temporalmente una parte razonable de los datos del modelo a los que se tiene acceso o que se almacenan ahí.

El manejo simultáneo de diferentes *vistas o subesquemas*, puede ocasionar problemas de definición de un modelo válido de Base de Datos. El caso de dos o más vistas puede ser ejemplificado por una relación en un archivo CLIENTE-MEDICINA, donde por un lado, se necesite controlar las medicinas recetadas para cada problema específico (diabetes, cáncer, etc.), o las necesarias para atender a cada uno de los pacientes (independientemente de que éstos padezcan o hayan padecido más de una enfermedad).

Es claro aquí que la organización de la información deberá ser diferente para cada uno de los problemas, y en consecuencia, la definición del modelo de Base de Datos se complica. Para eliminar este problema, en las Bases de datos relacionales se definirá la Base de Datos desde un solo punto de vista, Con posterioridad, se integrarán las *vistas* necesarias.

3.3.1. TIPOS DE RELACIONES.

Los tipos de relaciones a que se hará referencia, y que son los que se utilizan para construír vistas y en general Bases de Datos, están determinados por la función que la llave y los campos asociados de una tupla tienen dentro de la definición de la tabla o relación, además de su interacción con otras tablas o relaciones y por último, por los tipos de dominio a que hacen referencia los atributos de una relación. A continuación se proporcionan ejemplos de estos dos tipos de atributos, los *atributos de valor*, y los *atributos de referencia*.

Los primeros describen en forma directa propiedades características de una entidad al asignar un valor al campo dado.

Supongamos las relaciones siguientes:

CLIENTE (CLIENTE#, NOMBRE, DOMICILIO, CLI-DESDE, ULT-COMPRA)
VENTA (NO-ORDEN, NO-ARTICULO, PRECIO, CANTIDAD)
ORDEN-VENTA (NO-ORDEN, CLIENTE#, FECH-ORDEN, FECH-ENTREGA, TOTAL-$)

Son ejemplos de atributos de valor el NOMBRE, DOMICILIO de un CLIENTE, pues si éste cliente deja de serlo, y se elimina su tupla de la relación, automáticamente se eliminan también estos valores.

Los segundos, describen características de una entidad en forma indirecta al hacer referencia a otra entidad, incluso en otra Base de Datos, con lo que no desaparecen al desaparecer la tupla que los contiene, por estar contenidos en otra relación.

Un ejemplo de atributo de valor es el CLIENTE# de una ORDEN-VENTA que también está contenido en la tabla CLIENTE, y que por lo mismo, si la orden de venta ha sido atendida, y su tupla borrada de la tabla, el CLIENTE# se conserva en la tabla de clientes. Se dice entonces que su existencia está determinada de manera independiente a la asignación que de éste atributo se haga.

3.3.2. RELACION DE ENTIDAD.

A una relación que define a un conjunto de elementos o entidades independientes se le denomina *relación de entidad*. La elección de tipos de entidad es un aspecto fundamental del diseño de una Base de Datos o cualquier vista.

Las entidades son comunmente elementos que pueden tocarse, contarse, moverse, adquirirse o venderse, y que no pierden su identidad durante cualquiera de estas manipulaciones. Un cambio en cualquier tupla de otra relación debido a la actualización, no requerirá ningun cambio en una relación de entidad.

Los atributos pueden ser valores básicos, o referencias a tuplas de otras relaciones, en cuyo caso la relación de entidad se transforma en una *relación de entidad referida*. En la Fig. 3.5 se visualiza este tipo de relación.

3.3.3. RELACIONES ANIDADAS.

La primera forma normal de relaciones excluye la repetición de grupos de atributos. Durante el proceso de normalización se trasladan estos *nidos* de atributos a otras tablas separadas a las que se denomina *relaciones anidadas*.

Las tuplas de las relaciones anidadas tienen una llave que es una composición de la llave de la relación propietaria y un valor atributo que es único dentro del nido. La Fig. 3.6 ilustra ésta situación.

Para continuar existiendo, las tuplas de las relaciones anidadas dependen de la tupla propietaria. Esto significa que es posible insertar tuplas en una relación anidada sólo si existe una tupla propietaria, y que al desaparecer ésta, también sus tuplas en la relación anidada tienen que desaparecer.

TABLA-EMPLEADO.

NOMBRE	FECHA-NAC	FECHA-ING	DEPTO#	S.S
MARTINEZ	12/[illegible]/54	2[illegible]/10/59	45	[illegible]
FLORES	17/[illegible]/54	29/10/59	42	[illegible]
PEÑA	30/04/[illegible]	[illegible]/10/59	42	[illegible]
SANCHEZ	01/03/56	29/10/59	45	[illegible]
ORNELAS	[illegible]	29/10/59	34	[illegible]
[illegible]	12/05/55	29/10/59	40	[illegible]
[illegible]	13/07/45	29/10/59	34	[illegible]

ESTE CAMPO ES UNA REFERENCIA A OTRA TABLA.

FIG. 3.5 RELACION DE ENTIDAD.

A)

TABLA-SUPER.

NOM-P	CVE-P	DPTO	CONYUGE	HIJOS	EDAD
PAUL	34	12	MARY	JOSEPH	11
PAUL	34	12	MARY	WILLIAM	10
JOSE	33	23	ANA	ANA	07
MARIO	32	12	ALICE	LUIS	13
MARIO	32	12	ALICE	ENRIQUE	05
ROBERTO	30	21	CECILIA	ROBERTO	03

B)

TABLA-SUPER.

NOM-P	CVE-P.	DPTO	CONYUGE
PAUL	34	[illegible]	MARY
JOSE	[illegible]	23	ANA
MARIO	32	12	ALICE
ROBERTO	30	[illegible]	CECILIA

TABLA-HIJOS.

CVE-P	HIJOS	EDAD
34	JOSEH	11
34	WILLIAM	10
33	ANA	07
32	LUIS	13
32	ENRIQUE	05
30	ROBERTO	03

FIG. 3.6 RELACION ANIDADA A) SIN NORMALIZAR B) NORMALIZADA

3.3.4. RELACIONES ANIDADAS EN ORDEN "n".

Las relaciones asociadas o *asociativas*, tienen una llave o *parte rectora* compuesta de dos o más atributos. Los atributos relacionan a cada tupla de la relación asociada, con eneadas de dos o más relaciones propietarias. El ejemplo en la Fig.3.7 reafirmará este concepto.

Cabe mencionar que, como se puede apreciar en la Fig. 3.7 las relaciones asociativas son aquellas en que se hace referencia a tablas mediante una clave, y la tupla correspondiente a ésta clave, siempre nos proporciona una información más detallada de las características del objeto en cuestión.

Resumiendo podemos listar las características de cada relación:

a) Relación primaria de entidad.

- No referida dentro del modelo.
- La parte rectora define a la entidad
- La existencia de Tuplas es determinada externamente.

b) Relación de entidad referida.

- Referida al interior de una vista o tabla.
- La parte rectora define a la entidad y establece los dominios para los atributos de referencia.
- La existencia de tuplas es definida externamente, pero la eliminación depende de la existencia de la referencia.

c) Relación anidada.

- Cada tupla de ésta debe tener una tupla asociada a la relación de entidad correspondiente.

RELACION-PROVEEDORES (RELACION PROPIETARIA)

ID_PROV	NOMBRE	DOMICILIO	C.P.
PROV1	ACEMEX	MORAL # 45 MZ.147	05645
PROV2	CAMSA	CEIBA # 1234	09876
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...
PROV90	NACOSA	GUELATAO # 24	08769

RELACION-PARTES (RELACION PROPIETARIA)

ID_PART	DESCRIPCION	TAMAÑO	PESO
PART1	RONDANAS	11 X 15	0.04
PART2	TORNILLO A.	4	0.25
...	...	...	...
...	...	...	...
...	...	...	...
PART65	CALENTADOR	165	10.0

RELACION-PEDIDO (RELACION ASOCIADA)

ID_PRO	ID_ART	NOMBRE-C	IMPORTE
PROV90	PART35	GAMES HMOS.	324,678.00
PROV23	PART2	GREYER INC.	34,567.00
...	...	...	...
PROV23	PART5	DISOS S.A.	256,700.00

FIG. 3.7 RELACION ASOCIADA DE ORDEN "n".

- La parte rectora define un propietario y sus tuplas anida correspondientes.
- Un nido poseído puede tener cero o más tuplas.

d) Relación asociada de orden "n"

- Cada tupla puede tener "n" propietarios.
- La parte rectora define solo a uno de éstos propietarios.
- Una combinación de propietarios puede tener una o más tuplas en la asociación.

3.4.- LENGUAJES DE CONSULTA FORMALES Y COMERCIALES.

3.4.1. LENGUAJES FORMALES.

Un lenguaje de consulta sirve para que el usuario solicite información de la Base de Datos. Estos lenguajes son normalmente de alto nivel, mayor que el de los lenguajes estándar de programación. Los lenguajes de consulta pueden clasificarse como *lenguajes de procedimientos* y *lenguajes sin procedimientos*. En un lenguaje de procedimientos el usuario le ordena al sistema que ejecute una serie de operaciones con la Base de Datos para obtener el resultado deseado. En un lenguaje sin procedimientos el usuario describe la información que desea sin indicar un procedimiento específico para obtenerla.

Todos los sistemas manejadores o sistemas comerciales de Bases de Datos de la actualidad cuentan con un lenguaje de consulta que incluye elementos de los dos tipos, tanto con procedimientos como sin ellos.

En primera instancia, existen dos lenguajes puros que se apegan a esta descripción: el *Algebra Relacional* y el *Cálculo relacional*. Aunque estos lenguajes puros no tienen la riqueza sintáctica de un lenguaje más comercial, sí ilustran las técnicas fundamentales para extraer la información de una Base de Datos.

3.4.2. ALGEBRA RELACIONAL.

El álgebra relacional es un lenguaje de consulta de procedimientos. Existen cierto tipo de operaciones que se pueden hacer para extraer la información. Dedicaremos mucha atención a este punto, dado que la estructura de lenguajes de consulta de los manejadores comerciales de Bases de Datos para microcomputadoras están basados en esta concepción.

Se puede decir que existen cinco operaciones básicas en el álgebra relacional, que son : **selección**, **proyección**, **unión** **diferencia** y **producto cartesiano**, y el resultado de todas ellas consiste en una nueva tabla o relación.

Al final de esta sección se hablará de otras operaciones en el álgebra relacional, que en realidad son combinaciones de las cinco operaciones básicas mencionadas, pero que vale la pena estudiar.

Las operaciones de **selección** y **proyección**, se denominan operaciones unarias, ya que actúan sobre una sola relación. las otras tres operaciones operan sobre parejas de relaciones, por l que se denominan binarias.

SELECCION.

La operación de selección opta por tuplas o renglones que satisfagan ciertas condiciones (predicado). Se utiliza la letra griega σ (sigma minúscula) para señalar la selección. El predicado aparece como subíndice de la σ. La relación que constituye el argumento o dominio de la operación, se da entre paréntesis despues de la σ. Así, para elegir las tuplas de la Tabla-Hijos de la Fig. 3.6 en las que el nombre es JOSE se escribirá:

$$\sigma_{\text{PADRE = "JOSE"}}\ (\text{TABLA-HIJOS})$$

con lo que se obtendría :

PADRE	NOMBRE	EDAD
JOSE	ALICE	07
JOSE	JOSE	10

en general se permite todo tipo de comparaciones utilizando los operadores $=$, $\neq$, $<$, $>$, $\leq$, $\geq$, además, se pueden combinar estos operadores utilizando los conectivos *y* ($\wedge$) y *o* ($\vee$).Así,para hallar a los hijos llamados JOSE JR. y cuyo padre se llame JOSE en la Tabla-Hijos de la Fig.3.6 se tendría:

$$\sigma_{\text{PADRE= "JOSE"}\ \wedge\ \text{NOMBRE ="JOSE"}}\ (\text{TABLA-HIJOS})$$

con lo que tendríamos la tabla :

PADRE	NOMBRE	EDAD
JOSE	JOSE	10

CLIENTE.

NOMBRE	CALLE	CIUDAD
PEPEZ	ALAMO	D.F.
GOMEZ	SAUCE	LEON
ALBA	LUCES	MEZA.

PRESTAMO.

SUCURSAL	NO-PREST	NOMBRE-C	IMPORTE
CENTRO	16	PEPEZ	100 000.00
NORTE	11	GOMEZ	22 000.00
SUR	09	ALBA	122 000.00

FIG 3.8 TABLAS CLIENTE Y PRESTAMO

PRESTAMO

SUCURSAL	NOMBRE	IMPORTE
CENTRO	PEREZ	400000.00
NORTE	GOMEZ	922000.00

DEPOSITO.

SUCURSAL	NOMBRE	IMPORTE
CENTRO	ALBA	400000.00
NORTE	PEREZ	922000.00

FIG. 3.9 TABLAS DEPOSITO Y PRESTAMO

Dado que la operación selección también permite comparaciones entre dos atributos o campos de la misma relación, una forma más fácil de realizar la selección anterior sería :

$$\sigma \text{ PADRE} = \text{NOMBRE (TABLA-HIJOS)}$$

y se obtendría el mismo resultado.

PROYECCION.

Cuando se trata de simplificar resultados, tomando en cuenta o presuponiendo ciertas consideraciones, se puede usar la operación proyección, la cual en realidad copia de su relación argumento algunas columnas. La operación proyección se denota por una Π (pi mayúscula), como subíndice se ponen los nombres de las columnas que se quiere *proyectar*.

Se podría expresar con la operación proyeccion el ejemplo de los hijos con el mismo nombre de su padre de la siguiente manera:

$$\Pi \text{ NOMBRE,EDAD } (\sigma \text{ NOMBRE=PADRE (TABLA-HIJOS)}$$

con lo que el resultado sería:

NOMBRE	EDAD
JOSE	10

Analizando el resultado, observamos que no aparece la columna nombre, como en el ejemplo anterior, esto a raiz de que nosotros sabemos el objetivo de nuestra consulta, por lo que sabemos que el padre de JOSE se llama también JOSE.

PRODUCTO CARTESIANO.

Esta es una operación que permite combinar información de varias relaciones, en la cual se utiliza notación infija, por lo que el producto cartesiano de dos relaciones se escribirá como r_1 X r_2.

Supongamos las relaciones de la fig 3.8.

La operación R = CLIENTE X PRESTAMO formaría la sig. tabla :

R(CLIENTE X PRESTAMO)

NOM-BRE	CALLE	CIUDAD	SUCURSAL	NO. PRES	NOMBRE-C	IMPORTE
PEREZ	ALAMO	D.F.	CENTRO	16	PEREZ	100,000.00
PEREZ	ALAMO	D.F.	NORTE	11	GOMEZ	22,000.00
PEREZ	ALAMO	D.F.	SUR	08	ALBA	122,500.00
GOMEZ	SAUCE	LEON	CENTRO	16	PEREZ	100,000.00
GOMEZ	SAUCE	LEON	NORTE	11	GOMEZ	22,000.00
GOMEZ	SAUCE	LEON	SUR	08	ALBA	122,500.00
ALBA	LUCES	NEZA	CENTRO	16	PEREZ	100,000.00
ALBA	LUCES	NEZA	NORTE	11	GOMEZ	22,000.00
ALBA	LUCES	NEZA	SUR	08	ALBA	122,500.00

Puede entonces decirse que simplemente se hace una lista de todos los atributos de ambas relaciones, agregando a cada uno el nombre de la relación de la cual provienen. Ahora, ¿de cuántas tuplas está compuesta la relación resultante? y, ¿cuáles son sus tuplas?.

Para contestar éstas preguntas, al mirar la tabla se deduce que si en la relación CLIENTES hay 3 tuplas y en la de PRESTAMO hay 3 tuplas el número total de tuplas en la relación CLIENTES X PRESTAMO sería 3 X 3 = 9 tuplas, así , en general, se tendría que el número total de tuplas en la tabla resultado de un producto cartesiano, es de n1 x n2 tuplas, donde n1 es el número de tuplas en la relación 1 y n2 es el número de tuplas en la relación 2.

Otra de las características del producto cartesiano es que existen tuplas, como en el caso del ejemplo, en que el campo CLIENTE.nombre ≠ PRESTAMO.nombre-c, cuando todos esperaríamos que fueran iguales, ésto quizá sea una deficiencia de la operación, sin embargo, puede subsanarse con una adecuada composición de operaciones de álgebra relacional. Así, si quisieramos obtener todos los clientes asociados a sus préstamos correspondientes, bastaría con una expresión como :

(σ CLIENTE.nombre = PRESTAMO.nombre-c (CLIENTE X PRESTAMO))

Con lo que se obtendría:

R(σ CLIENTE.nombre = PRESTAMO.nombre-c (CLIENTE X PRESTAMO))

CLIENTE NOMBRE	CLIENTE CALLE	CLIENTE CIUDAD	PRESTAMO SUCURSAL	PRESTAMO NO-PREST	PRESTAMO NOMBRE-C	PRESTAMO IMPORTE
PEREZ	ALAMO	D.F.	CENTRO	16	PEREZ	100,000.00
GOMEZ	SAUCE	LEON	NORTE	11	GOMEZ	22,000.00
ALBA	LUCES	NEZA	SUR	08	ALBA	122,500.00

UNION.

Como en el caso de la teoría de conjuntos, la *unión* en el álgebra relacional, corresponde en hallar los elementos que se encuentren en una relación o en otra o en ambas, y se simboliza por r1 U r2, y debe cumplir con dos condiciones para poderse realizar :

- Las relaciones r1 y r2 deben tener el mismo número de atributos
- Los dominios de los atributos deben ser los mismos.

En general, debe cuidarse que la unión se realice entre relaciones compatibles, pues no tendría sentido unir las relaciones CLIENTE y PRESTAMO puesto que una es de tres columnas y la otra de cuatro, además sus campos o atributos no corresponden. Considérense las relaciones préstamo y depósito :

La unión de estos dos conjuntos, expresada como una proyección de los nombres de los clientes, dado que es así como normalmente se utiliza, expresada por:

(Π PRESTAMO.NOMBRE ∪ Π DEPOSITO.NOMBRE)

Sería la relación de la figura 3.10:

DIFERENCIA.

El operador diferencia, señalado por (-) permite encontrar las tuplas que están en una relación, pero no en otra. La notación es r1 - r2, y el resultado será las tuplas que estén en r1 *pero no* en r2. Nótese que el orden de las relaciones afectaría el resultado.

Como ejemplo, encontraremos a todos los clientes que estén en la relación de depósito pero no en la de préstamo (Fig. 3.9). La operación se escribirá DEPOSITO - PRESTAMO y el resultado sería :

DEPOSITO - PRESTAMO

SUCURSAL	NOMBRE	IMPORTE
SUR	ALBA	34 000.00

Cabe destacar que ésta operación debe cumplir con las mismas reglas de la unión.

Operadores Adicionales.

Las cinco operaciones fundamentales del álgebra relacional, son suficientes para expresar cualquier consulta a la Base de Datos, sin embargo, algunas consultas resultarán ser expresiones muy largas si solamente se usan estos operadores. Es por esto que se definen operadores adicionales que, sin hacer más potente al álgebra relacional, simplifican muchísimo el formato de algunas expresiones de consultas comunes.

INTERSECCION.

Como en la teoría de conjuntos, ésta operación sirve para encontrar elementos que están en uno u otro conjunto, con la diferencia de que aquí se manejan relaciones. La forma de escribir la operación intersección es r1 $\cap$ r2, y la razón de que no sea nombrada como una operación fundamental, es la de que se puede expresar como una pareja de diferencias de conjunto, así:

$$r1 \cap r2 = r1 - (r1-r2)$$

Como ejemplo, encontrar los clientes que tienen un depósito y también un préstamo en las tablas de la Fig. 3.9, se escribiría DEPOSITO $\cap$ PRESTAMO, y daría como resultado :

DEPOSITO $\cap$ PRESTAMO

SUCURSAL	NOMBRE	IMPORTE
CENTRO	PEREZ	100 000.00
NORTE	GOMEZ	22000.00

DIVISION.

Una de las consultas más comúnes a una Base de datos es la ejemplificada por "encuentrese el atributo A1 en la relación r1 y todos sus correspondientes atributos A2 en la tabla r2". Para realizar ésta operación, en términos de los operadores fundamentales, se necesitan 5 operaciones fundamentales, sin embargo existe la operación división que simplifica la notación, esto es :

$$r1 \div r2 = \Pi_{r1-r2}(r1) - \Pi_{r1-r2}((\Pi_{r1-r2}(r1) \times r2) - r1)$$

Todo esto, que parece muy complicado, se ejemplificará a continuación:

Sean las relaciones:

LENGUAJES

EMPLEADO	LENGUAJE
JONES	COBOL
PEREZ	CLIPPER
AGUILAR	FORTRAN

LENG_MINI

CVE.	DESCRIP
01	COBOL
02	FORTRAN

La expresión LENGUAJES.lenguaje ÷ LENG_MINI.descrip arroja como resultado:

EMPLEADO
JONES
AGUILAR

que es muy similar a seleccionar rengoles, restar, unir y proyectarlos, pero es más sencillo de escribir.

Existen otras operaciones del álgebra relacional tales como el *producto natural* y el *producto* ⊖, que son básicamente, combinaciones de producto cartesiano, selección y proyección.

3.4.3. *CALCULO RELACIONAL.*

El cálculo relacional es un lenguaje de consulta formal *sin procedimientos*, ya que en él se da una descripción formal de la información deseada sin explicar cómo obtenerla. Existen dos variantes del cálculo relacional, una en la que se consideran como variables a las tuplas y otra en la que se consideran como variables a valores de dominios, pero ambas son muy similares.

Una consulta en el cálculo relacional de tuplas se expresa como:

$\{ t \mid P(t) \}$

es decir, el conjunto de todas las tuplas t tales que el predicado P se cumpla para t. También se emplea t[A] para denotar el valor de la tupla t con respecto del atributo A, y $t \in r$ para expresar que la tupla t está en la relación r.

Regresando al ejemplo de la Fig.3.9 la lista de las tuplas en las que el nombre en la relación PRESTAMO es igual que el de la relación DEPOSITO se expresaría como:

$\{ t \mid t \in \text{PRESTAMO} \wedge s \in \text{DEPOSITO} \wedge t[\text{NOMBRE}] = s[\text{NOMBRE}] \}$

Otros operadores usados en el cálculo relacional de tupla son :

- $\exists$ para expresar la frase "existe...."
- $\forall$ para expresar la frase "... para todos(as)"

Definiendo formalmente al cálculo relacional de tuplas, se tiene que, P es una fórmula, y una variable de tupla es una variable *libre* cuando no está cuantificada por un "$\exists$" o un "$\forall$". Cuando sí sucede ésto, se dice que es una variable *atada*.

Una fórmula en el cálculo relacional de tuplas está formada por átomos. Un átomo puede tomar una de las siguientes formas :

- $s \in r$; donde s es una variable de tupla y r es una relación.
- $s[x] \ominus u[y]$; donde s y u son variables de tupla , x es un atributo en el que s está definida y $\ominus$ es un operador de comparación. Se requiere que los atributos x y y tengan dominios cuyos miembros puedan compararse por medio de $\ominus$.
- $s[x] \ominus c$; donde c es una constante en el dominio del atributo x.

Las fórmulas se construyen con átomos empleando las reglas siguientes :

- Un átomo es una fórmula.
- Si P_1 es una fórmula, entonces $\neg P_1$ y (P_1) también lo son.
- Si P_1 y P_2 son fórmulas, entonces también lo son $P_1 \wedge P_2$ y $P_1 \vee P_2$.
- Si $P_1(s)$ es una fórmula que contiene a una variable de tupla libre, entonces $\exists s(P_1(s))$ y $\forall s(P_1(s))$ también son fórmulas.

Cabe destacar que aquí también se pueden escribir muchas expresiones equivalentes, para lo cual se incluyen las dos reglas siguientes:

- $P_1 \wedge P_2$ es equivalente a $\neg(\neg P_1 \vee \neg P_2)$.
- $\forall t(P_1(t))$ es equivalente a $\neg\exists t(\neg P_1(t))$.

El último aspecto a mencionar en el cálculo relacional de tuplas es que ; para evitar evaluar condiciones infinitas, se incluye el concepto de *dominio de una f rmula.*

El dominio, expresado por dom(P), es el conjunto de los valores mencionados explícitamente en P, como los de las relaciones mencionadas en la misma fórmula.

Para aclarar esto, supongamos la consulta {t |t ∉ préstamo }. Si no existiera el concepto de dominio, hablaríamos de un resultado infinito, puesto que existen infinidad de tuplas que no pertenecen a la relación PRESTAMO, incluso fuera del sistema de Base de Datos, y al existir el concepto, la búsqueda se restringe a las tuplas en la relación PRESTAMO y el resultado es una relación vacía, pues no se menciona otra condición.

Esta consideración hace posible la concepción de *expresión segura* dentro del cálculo relacional de tuplas, y la cúal cumple sólo para los valores mencionados en P, ya sea explícita o implícitamente en forma de relaciones.

CALCULO DE DOMINIOS.
Una consulta en el cálculo relacional de dominios se expresa como:

$\{ < X_1, X_2, \ldots, X_n > \mid P(X_1, X_2, \ldots, X_n) \}$

Donde las X's representan variables de dominio y la P es una fórmula. Las variables de dominio no son más que nombres de columnas en la tabla. Las fórmulas están formadas por átomos y éstos son de las siguientes formas :

- Un átomo es una fórmula.
- Si P_1 es una fórmula, entonces $\neg P_1$ y (P_1) también lo son.

- Si P_1 y P_2 son fórmulas, entonces también lo son $P_1 \wedge P_2$ y $P_1 \vee P_2$.
- Si $P_1(s)$ es una fórmula que contiene a una variable de tupla libre, entonces $\exists\, s(P_1(s))$ y $\forall\, s(P_1(s))$ también son fórmulas.[1]

La idea de *expresión segura* también se aplica en el cálculo de dominios.

3.4.4. LENGUAJES COMERCIALES.

Los lenguajes formales permiten representar las consultas a una Base de Datos en forma concisa, sin embargo, es necesario que los manejadores de Bases de Datos comerciales, en virtud de que están orientados "al usuario", contengan un lenguaje de consulta que sea más amable con éste.

Aunque existen infinidad de lenguajes comerciales de consulta, dado que con cada nuevo manejador se intenta implementar un nuevo lenguaje de consulta comercial, sería difícil analizarlos todos, por lo que enfocaremos nuestro estudio hacia tres lenguajes comerciales: SQL, QBE, y QUEL.

La razón de que sean éstos y no otros los lenguajes escogidos es la de que cada uno de estos tres lenguajes representa un estilo diferente: QBE esta basado en estructuras de cálculo relacional de dominios, QUEL se basa en el cálculo relacional de tuplas y SQL utiliza una combinación de álgebra relacional y construcciones de cálculo relacional. Además, los tres son importantes no sólo en las Bases de Datos para investigación, sino también en los sistemas distribuídos comercialmente para minis y microcomputadoras.

[1] KORTH AND SILBERSCHARTZ/ FUNDAMENTOS DE BASES DE DATOS/ED MC. GRAW-HILL/ P.65

3.4.5. SQL.

La implementación inicial de SQL se pensó como lemguaje de consulta par el SYSTEM R. El nombre SQL está formado por las iniciales en inglés de "Lenguaje de consulta estructurado" (Structured Query Language), e inicialmente su nombre fue SEQUEL.

La estructura básica de una expresión en SQL se compone de tres cláusulas principales:

- SELECT (Elegir)
- FROM (De)
- WHERE (Donde)

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

La cláusula FROM es una lista de relaciones o tablas que se examinarán durante la ejecución de la expresión, es decir, es el dominio de la expresión, su campo de acción.

La cláusula WHERE corresponde al predicado de selección del álgebra relacional. Se compone de un predicado que es una serie de condiciones que incluyen atributos de las relaciones involucradas en la cláusula FROM.

Una consulta en SQL esta construída de la siguiente manera :

SELECT A_1, A_2, ... , A_n
FROM r_1, r_2, ..., r_n
WHERE P

Las A's representan atributos, las r's representan relaciones, y la P es un predicado.

Si se omite la cláusula **where**, el predicado P es verdadero. La relación de atributos $A_1...A_n$ puede sustituírse por *, lo cual causa que se elijan todas las columnas de la relación o relaciones que aprecen en la cláusula **from**.

El proceso que sigue SQL, para realizar una consulta, es el siguiente:

- SQL forma el producto cartesiano de las relaciones que se nombran en la cláusula **from**.

- Realiza una selección utilizando el predicado de la cláusula **where**.

- Proyecta el resultado a los atributos de la cláusula **select**.

El resultado de una consulta en SQL es, por definición, una relación. En los lenguajes formales, se utilizó el concepto de que una relación es un conjunto, por lo que en los resultados no aparecen tuplas repetidas, sin embargo, en SQL sí se permite duplicidad, y el eliminar las tuplas duplicadas debe indicarse explícitamente con la cláusula **DISTINCT**.

SQL incluye las operaciones **UNION**, **INTERSECTION** y **MINUS** que corresponden directamente a los operadores $\cup$, $\cap$ y - del álgebra relacional.

Utilizando las tablas de la Fig.3.9, podemos escribir la consulta "Encuentre los usuarios que tienen un depósito o un crédito o ambos en la sucursal centro" de la sig. manera :

```
select nombre
from   depósito
where  sucursal="centro"
unión
select nombre  from   préstamo
where  sucursal="centro"
```

Con lo que obtendríamos el mismo resultado de la Fig. 3.10.

Ya dijimos que en la cláusula from se designan las relaciones a ser utilizadas, por lo que, operaciones como el producto natural son fácilmente escritas. Cosidérese las tablas de la Fig. 3.8.

La consulta " encuéntrense los usuarios que tienen un préstamo en la sucursal norte y su correspondiente ciudad" se escribiría:

```
select cliente.nombre, ciudad
from   préstamo, cliente
where  cliente.nombre = préstamo.nombre and
       sucursal="NORTE".
```

Nótese que se emplea la notación *relación.atributo* para evitar ambiguedad, ya que un mismo atributo puede aparecer en distintas relaciones. Cuando se está seguro que un nombre de atributo es único para las relaciones mencionadas en la cláusula from, no resulta ambiguo escribir sólo el nombre del atributo.

En este último ejemplo se utilizó la cláusula and, la cual, al igual que or y not, se relacionan directamente con los símbolos matemáticos $\wedge$, $\vee$ y $\neg$, respectivamente, y su uso es el mismo que en el álgebra relacional.

Anteriormente se mencionó que SQL utilizaba construcciones del cálculo relacional. Estas construcciones están representadas por las cláusulas in, not in, any, contains, not contains, order by y group by.

La función de la cláusula in es la de checar si cierto atributo pertenece a un cierto conjunto, por ejemplo, la consulta :

```
select cliente.nombre
from   cliente
where  nombre in ( select nombre
                   from   prestamo)
```

Al contrario, la cláusula not in verifica si no se pertenece a un cierto conjunto, como en la consulta:

```
Select cliente.nombre
from   cliente
wherw  nombre not in  (select nombre
                       from   préstamo)
```

la cual da como resultado una lista de los nombres de clientes que no tengan un préstamo.

Aquí, nótese que, como en el caso del álgebra relacional, se pueden anidar subconsultas para lograr expresiones complicadas.

El SQL emplea el concepto de variables de tupla del cálculo relacional de tuplas. Una variable de tupla en SQL debe estar asociada a una relación determinada.

Las variables de tupla a usar se definen dentro de la cláusula **from**, colocándolas después del nombre de la relación con la que están vinculadas, y son muy útiles cuando se quiere comparar dos tuplas de la misma relación.

Supóngase la consulta "encontrar a todos los clientes que tengan un depósito en el mismo lugar que el cliente GOMEZ"

Utilizando la base de datos DEPOSITO, se escribiría:

```
select T.nombre
from   depósito S, depósito T
where  S.nombre = " GOMEZ " and
       S.sucursal = T.sucursal
```

cuyo resultado sería :

```
R
-----------
  GOMEZ
  HERNAN
  GARCES
-----------
```

Para una tabla de depósito como la de la Fig 3.11. Observe que no puede utilizarse la notación DEPOSITO.nombre, por que no se sabría a qué referencia de la relación depósito se refiere la consulta.

Pero el utilizar este tipo de construcciones tiende a hacer mas enredado el proceso de compresión de una consulta, por ello se utilizan operadores alternos.

NOMBRE
PEREZ
GOMEZ
ALBA

FIG. 3.10 Π DEPOSITO.NOMBRE ∪ Π PRESTAMO.NOMBRE.

SUCURSAL	NO.PREST.	NOMBRE	IMPORTE
CENTRO	10	PEREZ	100000.00
NORTE	11	GOMEZ	22000.00
SUR	08	ALBA	122000.00
NORTE	12	GARCES	945879.00
CENRO	13	RUIZ	54900.00
NORTE	04	HERNAN	12000.00

FIG. 3.11 TABLA DEPOSITO.

TABLA-CLIENTE

CLIENTE	NOMBRE	CALLE	CIUDAD

TABLA-DEPOSITO

DEPOSITO	SUCURSAL	DEP-NO.	NOMBRE	IMPORTE

TABLA-PRESTAMO

PRESTAMO	SUCURSAL	DEP-NO.	NOMBRE	IMPORTE

FIG. 3.12 ESQUELETOS DE TABLAS EN QBE.

Considere la consulta "encontrar todos los depósitos que sean mayores a alguno de los hechos en la sucursal norte". Utilizando la Fig. 3.1, se escribiría:

```
select T.sucursal
from   depósito T, depósito S
where  T.importe > S.importe and
       S.sucursal = "norte"
```

Dado que estamos haciendo una comparación "mayor que", es claro que no puede ser construída ésta consulta a través de un operador in, por lo que se utilizará el operador > any para destacar ésta consulta, además, esto permite que la construcción de la consulta se parezca mucho a su expresión en español. Así la construcción alterna sería :

```
select sucursal
from   depósito
where  importe > any ( select importe
                       from depósito
                       where sucursal =
                      " norte " )
```

La subconsulta despues de " > any " genera el conjunto que se va a evaluar, y la comparación > any es verdad si por lo menos un miembro del conjunto seleccionado es menor que la tupla a evaluar.

El SQL permite evaluar condiciones que incluyan todos los operadores de relación (>, =, ≠, etc) y la cláusula any no es la excepción. Como caso especial, = any es idéntica a in.

En ocasiones se pretenderá evaluar conjuntos para saber si uno contiene a otro o no, para ello se utilizan las construcciones **contains** y **not contains**.

Si consideramos las tablas de la fig. 3.8, y la consulta "verificar que todos los clientes tienen un depósito". podríamos escribir:

```
select nombre
from   clientes
where   (select nombre
         from clientes)
        contains
        (select nombre
         from   depósito).
```

La cláusula **order by** hace que las tuplas seleccionadas en una consulta, aparezcan en un orden predeterminado. Para listar en orden alfabético todos los clientes que tengan un depósito en la sucursal norte escribiremos :

```
select   nombre
from     depósito
where    sucursal = "NORTE"
order by nombre
```

Es claro que para satisfacer una solicitud de **order by** , el SQL debe realizar una operación de clasificación, operación por demás costosa en cualquier sistema, trátese del que se trate, por eso es conveniente ordenar los archivos de otra manera, típicamente, con el uso de archivos de índices de los manejadores de Base de Datos comerciales.

Dentro de las funciones que ofrece el SQL, existe una que permite el cálculo por grupos, ésta es la cláusula group by. El atributo que se da en la cláusula group by sirve para escoger las tuplas que tengan el mismo valor en este atributo, las cuales son agrupadas. Además, SQL incluye funciones de evaluación como avg, que calcula el promedio, min, que calcula el mínimo, max, máximo, sum, calcula la suma, count, calcula el número de tuplas en la selección. Estas operaciones se llaman *operaciones de agregados*, por que operan sobre grupos de tuplas.

Así, averiguar cuál es el importe promedio de todas las sucursales en la tabla de la Fig. 3.11 se escribirá :

```
select   sucursal, avg (importe)
from     depósito
group by sucursal.
```

Algunas veces, es conveniente utilizar condiciones que se aplican a grupos, puesto que minimiza el dominio de búsqueda en alguna operación, para ello, se utiliza la cláusula having (teniendo). Los predicados de ésta cláusula se aplican después de la formación de grupos, por lo que ahorran trabajo.

Como ejemplo, supóngase que se quiere una lista con las sucursales en las que el importe promedio sea mayor de 100,000.00, lo cual obtendríamos con la siguiente consulta :

```
select      sucursal, avg(importe)
from        depósito
group by sucursal
having avg(importe)> 100000.00
```

Frecuentemente, el operador de agregados **count** se utiliza para contar todas las tuplas de una relación, utilizando la notación :

```
select count(*)
from   nombre-relación.
```

Para obtener información negativa, por ejemplo las tuplas de la relación depósito cuyo nombre no exista en la relación cliente, utilizar count es un tanto difícil, para ello, existe una construcción especial, llamada exist, que resuelve este problema. La cláusula exist es verdadera, a menos que la proposición select que tiene en su argumento resulte en una relación vacía.

Para realizar la consulta anterior, utlizando **exist**, escribiríamos:

```
select nombre
from   depósito
where  not exist ( select  nombre
                   from    cliente
                   where cliente.nombre =
                         depósito.nombre)
```

MODIFICACIONES A LA BASE DE DATOS

Hasta ahora se ha definido al SQL como un lenguaje de consulta, sin embargo, a través de él se pueden realizar otro tipo de operaciones, tales como eliminar, cambiar o agregar información nueva a la Base de Datos. En general, los lenguajes formales como el álgebra y cálculo relacionales, no preveen la modificación de la Base de Datos, pero los lenguajes comerciales sí incluyen estas funciones, pues es incluso, a través de ellos, como se realiza la definición lógica de la Base de Datos.

Elinimación.

Eliminar tuplas en una relación es sencillo, dado que una solicitud de eliminación se expresa en SQL en forma muy parecida a una consulta, sólo que en vez de mostrar las tuplas seleccionadas, éstas se borran de la Base de Datos.La reestricción es que solo se pueden eliminar tuplas completas; no pueden eliminarse únicamente los valores de determinado atributo. Su construcción es como sigue:

```
delete r
where  P
```

El comando **delete** opera sobre una sola relación, de modo que si se requiere borrar tuplas de diferentes relaciones, es preciso utilizar un comando **delete** para cada relación. El predicado de la cláusula **where** puede ser tan sencillo, incluso no existir, con lo que se borrarán todas las tuplas de la relación referenciada, o tan complicado como cualquier clase de consulta en SQL. Aquí se darán los ejemplos para cada uno de estos casos.

- Borrar todas las tuplas de la relación préstamo

```
delete   préstamo
```

- Borrar los clientes que no tienen préstamo de la relación depósito.

```
delete depósito
where  nombre not in(select  nombre
                     from    préstamo)
```

Es necesario recalcar que aunque sólo se pueden borrar tuplas de una relación a la vez, esto no afecta el que se puedan mencionar cualquier numero de relaciones en una subconsulta.

Una característica especial de la eliminación de tuplas, es el hecho de que primero se marcan las tuplas seleccionadas, y posteriormente, al final de la selección, son borradas, esto permite que las selecciones que son afectadas por el número de tuplas existentes en la Base de Datos, no se alteren y produzcan resultados erróneos.

En la mayoría de los manejadores de Bases de Datos de la actualidad, el borrado definitivo de las tuplas marcadas se realiza fuera de la petición de eliminación, mediante un comando especifico (v.g. *pack* en DBASE III+ o DBMS PRIME).

Inserción.
La inserción de datos en una relación se realiza de dos diferentes maneras, ya sea especificándose la tupla que se va a insertar, o escribiendo una consulta que dé como resultado el conjunto de tuplas a insertar. Los valores de las tuplas para cada uno de los atributos, deben pertenecer al dominio de cada uno de estos y deben ser del mismo orden que las existentes.

El ejemplo de inserción de una sola tupla sería el siguiente:

```
insert into  depósito
       values ( "NORTE", 88, "SALLEZ", 1000)
```

En una forma más general, el ejemplo de inserción de tuplas basándose en una consulta sería el siguiente:

```
insert into depósito
     select sucursal, prest#, nombre, importe
     from   préstamo
     where importe > 120000
```

Actualización.

Existen ocasiones en que sólo se desea cambiar el valor de algún atributo de una tupla, sin afectar a los demás. Para ello, se utiliza la proposición **update** (actualizar). Supóngase que los clientes de la tabla depósito, están recibiendo intereses por el 5%, con lo cual se debe actualizar el atributo IMPORTE de la tabla DEPOSITO, esto se escribiría :

```
update depósito
   setimporte = importe * 1.05
```

Dado que no se utilizó una subconsulta para generar el conjunto de tuplas afectadas, ésta operación se realiza una vez para cada tupla en la relación depósito. Para ejemplificar el uso de consultas en la actualización, supóngase que ahora se otorgará un interes de 10%, pero sólo a los clientes que tengan un depósito mayor a 100,000.00 :

```
update depósito
  set importe = importe * 1.1
  where importe > 100000
```

Vistas.

Por razones de seguridad de información, muchas veces no es conveniente que todos los usuarios tengan acceso al modelo conceptual total de la base de datos, por lo que se crean relaciones " virtuales " llamadas *vistas* para, a través de ellas, controlar el acceso de el usuario a la información. Las vistas no son más que conjuntos de atributos de una o más relaciones, que son presentados al usuario como si ésta fuera "su propia" Base de Datos, ya que para él sólo existirán los atributos que él puede ver.

Una vista se define empleando el comando de SQL **create view** (crear vista). Para definir una vista se le debe asignar un nombre y emplear una consulta que calcula la vista. Su sintaxis es:

create view *vista* **as** *<expresión de consulta>*

donde *vista* es el nombre de la vista y *expresión de consulta* es cualquier consulta válida en SQL.

Un ejemplo de creación de vista se da a continuación.

```
create view  cliente-parcial as
       (select nombre, sucursal
        from   cliente)
       union
       (select pres#, importe
        from   préstamo)
```

Esta consulta crearía una vista conteniendo datos de dos relaciones, con lo que se pone de manifiesto otra de las utilidades de las vistas, que es la de combinar información para el usuario, sin que este tenga que realizar ninguna operación adicional.

De ésta consideración se desprende el concepto de *vista universal*. En este modelo se proporciona al usuario una vista que es el producto natural de todas las relaciones de la Base de Datos relacional, y cuya principal ventaja es que los usuarios no necesitan preocuparse de referenciar a la relación correcta, además de que el SQL se simplificaría puesto que algunas cláusulas, (como la cláusula from) no serían necesarias.

Desde este punto de vista sería fácil preguntar el porqué SQL no está orientado a vistas, siendo que reducen el trabajo y proveen seguridad, sin embargo, dado que una vista es una parte de la tabla o relación, las actualizaciones a la vista serían parciales en la relación, lo cual generaría problemas significativos de atributos vacíos.

Existen muchas otras cláusulas y opciones de SQL, entre las cuales se cuenta con proposiciones para crear la Base de Datos, modificar su estructura de atributos, generar sinónimos (*alias) etc*, y varían de acuerdo a la implementación de la que se esté hablando, sin embargo, en la mayoría de la implementaciones de SQL se cuenta con las construcciones descritas en este capítulo, queda a consideración del lector el consultarlas en los manuales de referencia de su implementación de SQL en particular.

Recapitulando, el SQL incluye funciones que no aparecen en el álgebra relacional, lo cual lo hace, en un sentido estricto, más poderoso que el álgebra, sin embargo, este poder tiene un precio, y es el de que el SQL incluye una cantidad considerable de redundancia de operaciones y predicados, sin embargo, se justifica.

Muchas de las implementaciones de SQL permiten hacer consultas desde un programa escrito en lenguajes de aplicación general, tales como COBOL, PL/1, FORTRAN, C, etc. Cuando esto sucede, se dice que se trata de una implementacion de SQL *inmersa* en el lenguaje de programación del cual se trate.

Esto amplía muchísimo la capacidad del programador para manejar la Base de Datos, por que permite que se haga interactuar, tanto a construcciones del propio lenguaje de programación, como a comandos de sistema operativo y construcciones de SQL.

3.4.6. QUEL.

El QUEL se introdujo como lenguaje de consulta para el sistema de Bases de Datos INGRES. La estructura básica del lenguaje es tomada del cálculo relacional de tuplas. Las consultas se realizan a través de las cláusulas **range of** (rango de),**retrieve** (obtener) y **where** (donde). Las reglas generales para construír una consulta en QUEL son las siguientes :

- Cada una de las variables de tupla utilizadas en una consulta es declarada en una cláusula **range of**.

- La cláusula retrieve tiene una función similar a la de **select** del SQL.

- La cláusula where contiene el predicado de selección.

Una consulta en QUEL tiene la forma :

```
range of t1 is r1
range of t2 is r2
range of tn is rn
retrieve (t1.A, t1.B,t2.D,...tn.Z)
where P
```

las t_i son variables de tupla, las r_i son relaciones y las A,B,D...,Z, son atributos. Se utiliza la notación **t.A** para hacer referencia al valor de la tupla t en el atributo a. Esto significa lo mismo que t[A] en el cálculo relacional de tuplas.

El QUEL no incluye operaciones del álgebra relacional, como la intersección, la unión o la diferencia. Además no se pueden anidar subconsultas , es decir, no se puede colocar una cláusula **retrieve** anidada dentro de una cláusula **where**. El QUEL, como el SQL,utiliza los operadores lógicos **and**, **or** y **not** en vez de "$\wedge$", "$\vee$" y "$\neg$" que son utilizados en el cálculo relacional de tuplas.

Para ejemplificar la estructura de las consultas y constatar algunas diferencias con SQL, a continuación se hará una consulta para la cual en SQL se utilizaría la operación unión: la lista de los cuentahabientes que tienen un préstamo o un depósito o ambos en la sucursal norte. Puesto que en QUEL no existe el operador unión, y se sabe que está basado en el cálculo relacional de tuplas, se usará como guía la expresión de ésta consulta en éste cálculo :

$$\{ t/ \exists s(s \in \text{PRESTAMO} \wedge t[\text{NOMBRE}] = s[\text{NOMBRE}] \wedge s[\text{SUCURSAL}] = \text{"NORTE"}) \vee \exists u (u \in \text{DEPOSITO} \wedge t[\text{NOMBRE}] = s[\text{NOMBRE}] \wedge u[\text{SUCURSAL}] = \text{"NORTE"}) \}$$

No podemos construír directamente de ésta expresión una consulta en QUEL, puesto que utiliza dos tablas diferentes mezcladas. Para lograr esto, debe crearse una relación nueva (que se llamará temp) e insertar en ella tuplas a través del comando retrieve into. Los nombres de los depositantes se obtendrían así :

```
range of  u is depósito
retrieve into temp (nombre)
where  u.sucursal ="NORTE"
```

Ahora, para incluír las personas que tienen préstamo se utiliza el comando append, el cual funciona de manera similar al comando retrieve, sólo que las tuplas se agregan a la relación que aparece después de la palabra clave to.

```
range of s is préstamo
append to temp (nombre)
where s.sucursal= "NORTE"
```

Ahora tenemos una relación que tiene nombres repetidos, ya que incluye nombres de clientes con préstamo o depósito o ambos, para eliminar los duplicados, se escribe :

```
range of t is temp
retrive unique (t.nombre)
```

QUEL clasifica temp y elimina las tuplas duplicadas.

El objetivo en este punto es mostrar que algunas consultas son muy complicadas en QUEL si se intenta utilizar sólo sus construcciones básicas, a diferencia de SQL en el que las construcciones básicas casi resuelven todos los problemas de consulta eficientemente con sus construcciones básicas.

Para remediar esto, QUEL considera ciertas *operaciones de agregados* que tienen la forma :

< operación de agregados > (r.A **where** P).

Y las operaciones de agregados son **count**, **sum**, **avg**, **max**, **min**, o **any**. Una operación de agregados puede aparecer en cualquier parte donde pueda aparecer una constante y en la cláusula **where**.

En relación al porqué se utiliza la inserción y la eliminación, diremos que aunque QUEL está íntimamente ligado con el cálculo relacional de tuplas, diremos que no existe un equivalente en QUEL para el "para todos" (V) del cálculo.

3.4.7. QBE.

Query by Example (QBE, consulta por ejemplo) es el nombre de un lenguaje de consulta y también del sistema de Base de Datos que lo contiene. Tiene dos características que lo distinguen: A diferencia de la mayoría de los lenguajes de consulta y de programación, el QBE tiene una sintáxis *bidimensional*. Una consulta en un lenguaje unidimensional como SQL o QUEL, puede escribirse en una sola línea (tan larga como se requiera). Un lenguaje bidimensional requiere dos dimensiones para expresarse. La segunda característica es que las consultas se realizan "por ejemplo", esto es, en vez de especificar un procedimiento para obtener la respuesta requerida, el usuario da un ejemplo de lo que desea y el sistema generaliza este ejemplo para calcular la respuesta a la consulta.

Las consultas en QBE se expresan empleando *esqueletos de tablas*. Estas tablas muestran el esquema de la relación, y se ven como las de la fig. 3.12, que representan a las relaciones depósito, préstamo y cliente.

Para determinada consulta, el usuario llena los esqueletos con *columnas muestra*, que se forman de constantes y de *elementos muestra*. Un elemento muestra es en realidad una variable de dominio; para distinguir éstas de constantes, las primeras van precedidas por un caracter de subrayado (_), v.g. _x, y las segundas aparecen solas.

Para encontrar a todos los clientes que tienen un depósito en la sucursal norte, se pide el esqueleto, de la relación depósito y se llena así :

DEPOSITO	SUCURSAL	PRES-NO	NOMBRE	IMPORTE
	NORTE			P._x

Esta consulta hace que el sistema busque a todas las tuplas en la relación depósito que tengan el valor "NORTE" en el atributo sucursal. Para cada una de esas tuplas, el valor del atributo nombre, se asigna a la variable x. El valor de la variable x se "imprime" (se despliega) porque el comando P aparece junto a la variable x.

A diferencia de QUEL o SQL, QBE sí realiza la eliminación de tuplas duplicadas automáticamente. Cuando se requieren las duplicadas, se inserta el comando ALL. después del comando P.

La función principal de las variables en el QBE es obligar a que los valores de ciertas tuplas tengan el mismo valor en ciertos atributos. Supóngase que se quiere encontrar a todos los clientes que tienen un préstamo en la sucursal norte y las ciudades donde residen. La forma de la consulta sería :

PRESTAMO	SUCURSAL	PRES-NO	NOMBRE	IMPORTE
	NORTE		_X	

CLIENTE	NOMBRE	CALLE	CIUDAD
	P._X		P._Y

Para efectuar ésta consulta, el sistema encuentra las tuplas de préstamo que tengan "NORTE" como valor del atributo sucursal. Para cada una de estas tuplas, el sistema encuentra las tuplas en cliente que tienen el mismo valor en el atributo nombre que la tupla de préstamo. Se despliegan los valores de los atributos nombre y ciudad.

Ahora supóngase que la consulta requiere una comparación de menor o mayor que, como por ejemplo, "encontrar todos los clientes para los cuales el préstamo es mayor a 100,000.00". La consulta sería:

PRESTAMO	SUCURSAL	PRES-NO	NOMBRE	IMPORTE
			P._X	

Para expresar un operador "or" en el QBE, se da una línea de ejemplo aparte para las condiciones que se van a unir con el "or", utilizando variables de dominio diferentes. El ejemplo sería: "encontrar los clientes que tienen un préstamo en la sucursal norte o en la sur o en ambas", y se escribiría :

PRESTAMO	SUCURSAL	PRES-NO	NOMBRE	IMPORTE
	NORTE SUR			P._x P._y

Compárese ésta consulta con la siguiente: "Encontrar todos los clientes que tienen un préstamo tanto en la sucursal norte como en la sur", la cual se muestra a continuación :

PRESTAMO	SUCURSAL	PRES-NO	NOMBRE	IMPORTE
	NORTE SUR		P._x _x	

La distinción fundamental en éstas dos consultas es el empleo de la misma variable de dominio en ambas líneas de la segunda consulta, mientras que en la primera se utilizan dos variables de dominio diferentes.

Las consultas que implican una negación se expresan en QBE colocando un signo "¬" en un esqueleto de tabla debajo del nombre de la relación y junto a una línea de muestra, como en la consulta " encuéntrese todos los clientes que tienen un préstamo en la sucursal norte pero no tienen un depósito ahí", como sigue:

PRESTAMO	SUCURSAL	PRES-NO	NOMBRE	IMPORTE
	NORTE		P._x	

DEPOSITO	SUCURSAL	PRES-NO	NOMBRE	IMPORTE
¬	NORTE		P._x	

El QBE encuentra todos lo valores de x para los cuales :

- Existe una tupla en la relación préstamo cuya sucursal es "NORTE" y cuyo nombre es la variable de dominio x.

- No existe una tupla en la relación depósito con ese mismo nombre que tenga la misma sucursal.

Así, "¬" puede leerse "no existe". Si se hubiera colocado este símbolo bajo del nombre de algún atributo, equivaldría a decir "≠". En ocasiones no es conveniente expresar todas las condiciones dentro del esqueleto de la relación. Para ello, QBE incluye un *cuadro de condiciones*, que permite expresar éstas, y tienen la forma :

condiciones
x ≠ "cruz"

El QBE incluye operaciones de agregados como en SQL y QBE. tales como el promedio (avg), count, sum, min, max. considere la consulta:

PRESTAMO	SUCURSAL	PRES-NO	NOMBRE	IMPORTE
	P.G			P._AVG.ALL

La cual obtiene el importe promedio de los importes por sucursal. La G. es muy similar a "group by sucursal" de SQL. El ALL asegura que se tomarán en cuenta todas las tuplas (dado que QBE elimina automáticamente los duplicados).

Al igual que en SQL y QUEL, el QBE incluye operaciones de actualización de la Base de Datos, refierase a los manuales correspondientes para un análisis de éstas construcciones.

CAPITULO 4.- GENERALIDADES DEL MODELO JERARQUICO

A manera de introducción se verá un base de datos de muestra que concierne a proveedores, partes y remesas; el propósito de utilizar éste ejemplo es preparar el terreno para los temas siguientes del presente capítulo y también del capítulo siguiente.

Los datos de ésta base de datos son: S (proveedores), P (partes), y SP (remesas). Para cada proveedor: un número de proveedor, nombre, el código de estado, lo localización del proveedor; Para cada parte: un número de parte, nombre, color, peso y la localidad donde la parte se almacena; y para cada remesa: un número de proveedor, un número de parte y la cantidad enviada.

Se supone que cada proveedor tiene un número único de proveedor y exactamente un nombre, valor del estatus y localidad. Asimismo, se supone que cada parte tiene un número de parte único y exactamente un nombre, color, peso y localidad; y que, en cualquier momento dado, no existe más de una remesa para una combinación dada de proveedor y parte.

En la figura 4.1 se muestra una posible vista jerárquica para la base de datos de proveedores y partes.

En esta vista los datos se representan por una sencilla estructura de árbol, en donde las partes están a un nivel superior que los proveedores. El usuario ve cuatro árboles individuales, u ocurrencias jerárquicas, una para cada parte. Cada árbol se compone de una ocurrencia de registro de parte, junto con un conjunto de ocurrencias de registro de proveedor subordinadas, una para cada proveedor de la parte. Cada ocurrencia de proveedor incluye la cantidad correspondiente enviada.

Nótese que el conjunto de ocurrencias de proveedor para una ocurrencia específica de parte puede contener cualquier número de miembros, incluso cero (como en el caso de P4).

El tipo de registro en el tope del árbol, se conoce en general como raíz. La figura 4.1 es un ejemplo de la estructura jerárquica más sencilla posible (exceptuando el caso de una jerarquía que se componga de una sola raíz), con una raíz y un solo tipo de registro dependiente. En general, la raíz puede tener cualquier número de dependientes; cada uno de éstos puede tener cualquier número de dependientes de nivel inferior, y así sucesivamente, hasta cualquier número de niveles.

Se puede equiparar la vista jerárquica de la figura 4.1 a un solo archivo, que contiene registros arreglados en cuatro árboles individuales. Sin embargo, éste archivo es muy complejo. En primer lugar, contiene varios tipos de registros, en el ejemplo hay dos, uno para las partes y el otro para los proveedores. En segundo lugar, también contiene *ligas* que conectan ocurrencias de estos registros; en el ejemplo hay ligas entre ocurrencias de parte y

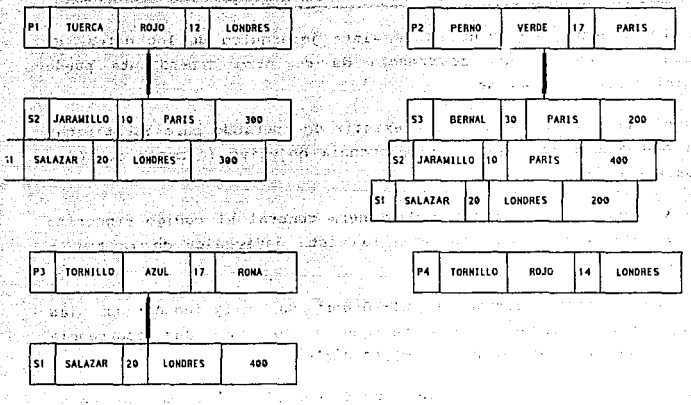


FIG.4.1.- DATOS DE MUESTRA EN FORMA JERARQUICA (LAS PARTES SUPERIORES A LOS PROVEEDORES).

Q1: HALLE NUMEROS DE PROVEEDORES DE LOS PROVEEDORES QUE SUMINISTRAN LA PARTE P2.	Q2: HALLE NUMEROS DE PARTE DE LAS PARTES SUMINISTRADAS POR EL PROVEEDOR S2.
OBTENGA [SIGUIENTE] PARTE DONDE P=P2; HAGA HASTA QUE NO HAYA MAS PROVEEDORES DEBAJO ESTA PARTE; OBTENGA SIGUIENTE PROVEEDOR DEBAJO ESTA PARTE; IMPRIMIR S#; FIN;	HAGA HASTA QUE NO HAYA MAS PARTES; OBTENGA SIGUIENTE PARTE; OBTENGA [SIGUIENTE] PROVEEDOR DEBAJO ESTA PARTE DONDE S# = S2; SI ENCONTRADO ENTONCES IMPRIMA P#; FIN;

FIG.4.2.- DOS CONSULTAS DE MUESTRA CONTRA LA VISTA JERARQUICA.

Es fundamental, en cuanto a la vista jerárquica de los datos, el hecho de que ninguna ocurrencia de registro dependiente puede existir sin su superior.

En el DML, por lo tanto, debe existir un operando para identificar la ocurrencia superior a la ocurrencia objetivo (a menos que ésta sea una raíz).

En la figura 4.2 se muestra de manera general el código requerido para manejar dos consultas con la vista jerárquica de la figura 4.1.

Se han puesto corchetes alrededor de "siguiente" en las preposiciones donde se espera que a lo sumo una ocurrencia satisfaga las condiciones especificadas.

Se supuso también que **donde** puede omitirse si no se desea especificar ninguna condición particular que deba ser satisfecha.
Los dos procedimientos que se muestran en la figura 4.2 no son simétricos. La pérdida de simetría es una consecuencia directa de la vista (figura 4.1), que en sí misma es asimétrica, donde las partes se tratan como superiores y los proveedores como dependientes.

Esta asimetría es un enorme inconveniente del enfoque jerárquico, porque lleva a complicaciones innecesarias para el usuario. El usuario está obligado a dedicar tiempo y esfuerzo a resolver problemas introducidos por la estructura de datos jerárquica. Esta situación empeorará con rápidez a medida que se introduzcan más tipos de registros en la estructura y la jerarquía se vuelva más compleja.
Esto significa que los programas son más complicados de lo necesario, con la consecuencia de que su escritura, depuración y mantenimiento necesitarán más tiempo de programador del debido.

Por otra parte, las jerarquías son una manera natural de modelar estructuras jerárquicas reales del mundo externo.

El ejemplo de los proveedores y las partes no sería el caso, pues hay una correspondencia de muchos a muchos entre los proveedores y las partes. En cambio, departamentos y empleados ofrecen un ejemplo de una estructura jerárquica genuina (siempre que un empleado pertenezca exactamente a un departamento).

Pero incluso en una estructura jerárquica genuina sigue presentándose el problema de la asimetría en la recuperación. Además, incluso las estructuras jerárquicas genuinas tienden a convertirse con el tiempo en estructuras de muchos a muchos más complejas.

Otra desventaja de éste tipo de estructura es que la estructura jerárquica de la figura 4.1 posee ciertas propiedades adicionales indeseables. Se presentan anomalías en relación con cada una de las tres operaciones básicas (insertar, suprimir y actualizar).

Estas anomalías, sin embargo, se deben directamente al hecho de que se trata de una situación de muchos a muchos; no se presentarían en una situación de uno a muchos.

Utilizando tres problemas sencillos que se describen a continuación, podemos explicar las dificultades del enfoque jerárquico:

1.- Dada la información acerca de un nuevo proveedor S4 en el área W, inserte ésta información en la base de datos.

2.- Suprima la remesa que conecta la parte P2 con el proveedor S3.

3.- El proveedor S1 se trasladó de Londres a Amsterdam.

INSERTAR.- Sería necesario introducir una parte ficticia especial, para insertar datos relativos a un proveedor nuevo, por ejemplo S4, hasta que ese proveedor suministre alguna parte.

SUPRIMIR.- La única manera de suprimir una remesa es suprimir la correspondiente ocurrencia de proveedor. Si se suprime la única remesa de un proveedor específico, se pierde toda la información de ese proveedor. Por ejemplo, la supresión de la remesa que conecta a P2 y S3 se hace suprimiendo la ocurrencia para S3 debajo de la parte P2, y como esa ocurrencia es la única para S3, esto causa que se pierda toda la información sobre S3.

Por otro lado, si se desea suprimir una parte que por casualidad es la única suministrada por algún proveedor, surge un problema similar, porque la supresión de cualquier ocurrencia de registro, automáticamente suprime todas las ocurrencias dependientes, de acuerdo con la filosofía jerárquica.

ACTUALIZAR.- Si se necesita cambiar la descripción de un proveedor, por ejemplo, cambiar la ciudad del proveedor S1 a Amsterdam, se debe buscar en toda la vista para hallar cada ocurrencia del proveedor S1.

Es necesario aclarar que para desarrollar éste capítulo, nos referimos al IMS como el ejemplo principal del enfoque jerárquico porque es uno de los más usados de todos los sistemas de bases de datos, jerárquicos o de otro tipo.

El nombre IMS es abreviatura de *Information Management System*. IMS es un programa producto de IBM diseñado para soportar programas de aplicación tanto de procesamiento por lote como en línea.

4.1 ARQUITECTURA DEL MODELO JERARQUICO

De la figura 4.3. podemos notar que los datos almacenados se componen de varias bases de datos, no sólo de una.
Una base de datos IMS es una representación almacenada de una *base de datos física*, y una base de datos física, es sencillamente una relación (más o menos) no normalizada, es decir, se compone de un conjunto de registros jerárquicos.

La *vista conceptual* se compone de un conjunto de bases de datos físicas, ésto no quiere decir que el usuario ve la base de datos exactamente como está almacenada; en realidad, IMS proporciona un grado bastante elevado de aislamiento del usuario con respecto a la estructura de almacenamiento y por lo tanto, un grado muy alto de independencia de los datos.

Cada base de datos física, se define por una descripción de base de datos (DBD). En la DBD también se especifica la correspondencia entre la base de datos física y el almacenamiento. Por lo tanto, el conjunto de todas las DBD's corresponde al esquema conceptual más parte de la definición asociada de la correspondencia conceptual interna.

Al igual que en la arquitectura general del Capítulo 1, el usuario no opera directamente al nivel de la base de datos física, sino sobre una *vista externa* (es la vista que se le presenta al usuario, en la que deliberadamente se omiten los detalles sobre la forma en que están representados los datos) de los datos.

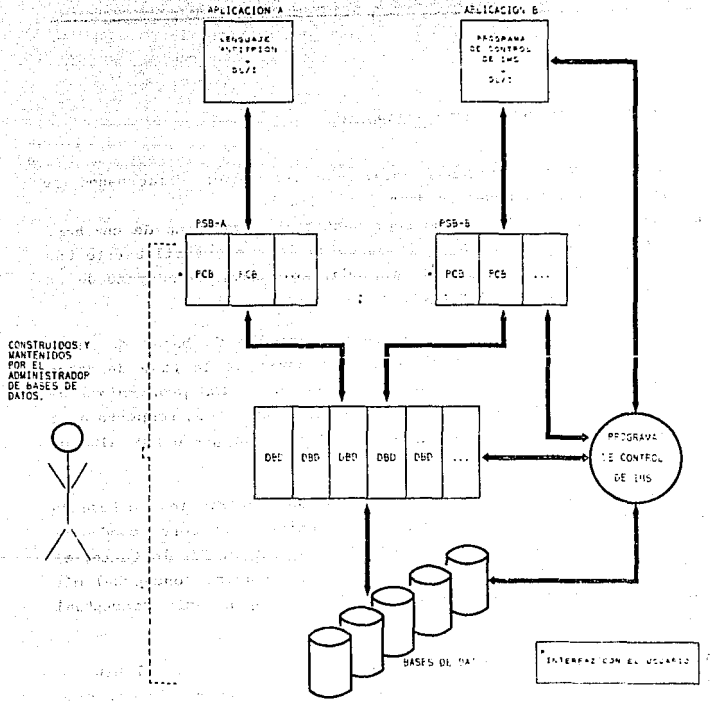


FIG. 4.3. ARQUITECTURA DE UN SISTEMA IMS.

Una vista externa de un usuario particular se compone de un conjunto de *bases de datos lógicas*, en donde cada base de datos lógica es un subconjunto de la base de datos física correspondiente [1].[1]

Cada base de datos lógica se define, junto con su correspondencia, con la respectiva base de datos física, por medio de un *bloque de comunicación del programa (PCB)*. El conjunto de todos los PCB's para un usuario se llama *bloque de especificación del programa (PSB)*, y corresponde al esquema externo más la definición de la correspondencia asociada.

Por último, los usuarios son programadores de aplicaciones ordinarios, que utilizan lenguaje anfitrión desde el cual se puede invocar al lenguaje de manipulación de datos de IMS, DL/I (Data Lenguage/I) por medio de llamadas a subrutinas.

Los *usuarios finales* son soportados (en un sistema con la característica de comunicaciones de datos) por medio de programas de aplicación en línea escritos por el usuario. IMS no proporciona un lenguaje de consulta integrado.

1.-EL TERMINO "BASE DE DATOS LOGICA" EN REALIDAD TIENE DOS SIGNIFICADOS DISTINTOS EN IMS, EL SEGUNDO SIGNIFICADO TAL VEZ SEA EL MAS IMPORTANTE PERO SE VERA POSTERIORMENTE.

4.2.- ESTRUCTURA DE DATOS DEL MODELO JERARQUICO.

4.2.1. BASES DE DATOS FISICAS.

Una base de datos física (PDB) es un conjunto ordenado, en donde los elementos se componen de todas las ocurrencias de un tipo de *registro de base de datos física (PDBR)*. A su vez, una ocurrencia de PDBR se compone de un arreglo jerárquico de ocurrencias de *segmentos* de longitud fija y una ocurrencia de segmento se compone de un conjunto de ocurrencias de *campos* asociados de longitud fija. La unidad de datos más pequeña que se puede accesar a una sola operación de DL/I es la ocurrencia de campo.

Para ejemplificar utilizaremos una PDB que contiene información acerca del sistema de educación interno de una compañía industrial grande. La estructura jerárquica de esta PDB (es decir, el tipo de PDBR) se ilustra en la figura 4.4.

En este ejemplo, suponemos un departamento de educación cuya función es dirigir cursos de capacitación. Cada curso se ofrece en varios sitios diferentes dentro de la compañía. La PDB contiene detalles tanto de los ofrecimientos ya efectuados como de los ofrecimientos programados para el futuro. Los detalles son como sigue:

-Cada curso debe tener un número único de curso, título, descripción, detalles de cursos requisito (si hay alguno), y detalles de todos los ofrecimientos (pasados y planeados).

-Para cada curso requisito de un curso dado, se tendrá el número del curso y el título.

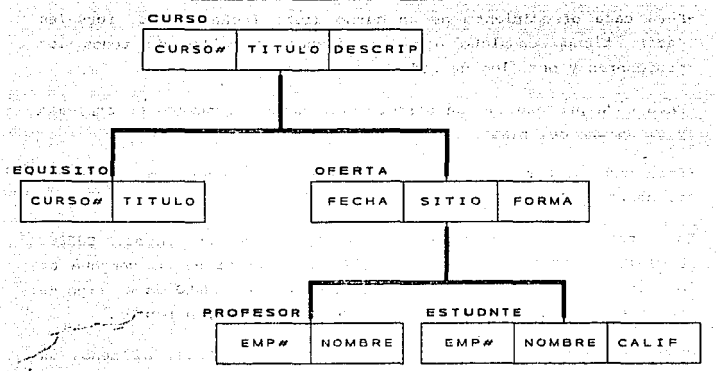


FIG.4.4 TIPO DE PDBR PARA LA BASE DE DATOS EDUCACION.

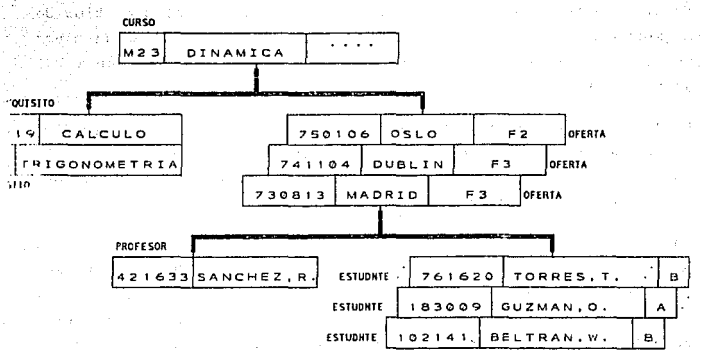


FIG.4.5 OCURRENCIA DE MUESTRA DE PDBR PARA LA BASE DE DATOS DE EDUCACION.

-Para cada ofrecimiento de un curso dado: fecha, sitio, forma(es decir, tiempo completo o medio tiempo), detalles de todos los profesores y detalles de todos los alumnos.

-Para cada profesor de un ofrecimiento dado: el número de empleado y el nombre del mismo.

-Para cada alumno de un ofrecimiento dado: el número de empleado, el nombre y la calificación.

En este ejemplo se tienen cinco tipos de segmentos: CURSO, REQUISITO, OFERTA, PROFESOR y ESTUDNTE; cada grupo se compone de los tipos de campos indicados en la figura 4.4, CURSO es el tipo de segmento *raíz*, los otros son tipos de segmento *dependientes*.

Cada dependiente tiene un *padre*, y cada padre tiene al menos un *hijo*; por ejemplo, el padre de PROFESOR y ESTUDNTE es OFERTA y CURSO tiene dos hijos.

Es importante aclarar que para una ocurrencia de cualquier tipo de segmento dado puede haber cualquier número de ocurrencias (incluso cero) de cada uno de sus tipos de segmento hijos. La figura 4.5 ilustra esto.

En la figura 4.4 se muestra solo una ocurrencia de la raíz (CURSO) y, por definición, una ocurrencia del tipo de PDBR de educación, la PDB completa la constituyen muchas ocurrencias de PDBR que representan información sobre muchos cursos.

En la figura 4.5 se tienen dos ocurrencias de REQUISITO y tres de OFERTA, dependientes de la ocurrencia de CURSO. La primer ocurrencia de OFERTA, a su vez, tiene una ocurrencia de PROFESOR y varias ocurencias de ESTUDNTE (sólo se muestran tres) dependientes de ella. Las otras ofertas no tienen profesores ni alumnos todavía.

La nomenclatura de padres e hijos que se utilizó anteriormente para los tipos de segmento, se aplica también a las ocurrencias de segmento. Por ejemplo, el padre de la ocurrencia de PROFESOR y las ocurrencias de ESTUDNTE es la primera ocurrencia de OFERTA, recíprocamente, las ocurrencias de PROFESOR y ESTUDNTE se consideran como hijos (ocurrencias de segmento hijo) de esta ocurrencia OFERTA. Además, todas las ocurrencias de un tipo particular de segmento hijo que compartan una ocurrencia de padre común se dice que son gemelas. Así, las ocurrencias de OFERTA de la figura 4.5 son gemelas, aunque haya tres de ellas.

Como conclusión a todo lo anterior se tiene que:

-Un tipo de PDBR contiene un sólo tipo de segmento raíz.

-La raíz puede tener cualquier número de tipos de segmento hijos.

-Cada hijo de la raíz puede tener también cualquier número de tipos de segmento hijos, y así sucesivamente (hasta un máximo de 255 tipos de segmento en cualquier ruta jerárquica y un máximo de 255 tipos de segmento en un tipo de PDBR completo [2]).

-Para cada ocurrencia de cualquier tipo de segmento dado puede haber cualquier número de ocurrencias (incluso cero) de cada uno de sus hijos.

-Ninguna ocurrencia de segmento hijo puede existir sin su padre.

-Como una reafirmación de la filosofía jerárquica, debe decirse que si una ocurrencia de un segmento dado se suprime, también se suprimen todos sus hijos.

2.-ESTAS CIFRAS, DESDE LUEGO, SON CARACTERISTICAS DE IMS Y NO PARTE FUNDAMENTAL DEL ENFOQUE JERARQUICO.

4.2.2. DESCRIPCION DE LA BASE DE DATOS.

Cada base de datos física se define por medio de una descripción de base de datos (DBD). La forma fuente de la DBD se escribe usando macroproposiciones especiales del Lenguaje Ensamblador del Sistema /370. (Estas macroproposiciones constituyen, pues, el "DDL conceptual" para IMS). Una vez escrita, la DBD es ensamblada y la forma objeto se almacena en una biblioteca del sistema, de la cual se puede extraer cuando la necesite el programa de control de IMS. En la figura 4.6 se muestra la "parte del esquema conceptual" para el ejemplo de la base de datos de educación. A las preposiciones se les ha asignado números como punteros de referencia para las explicaciones que siguen:

1: Esta preposición sólo asigna el nombre PDBDEDUC ("descripción de la base de datos física de educación") a la DBD. Todos los nombre de IMS están limitados a ocho caracteres.

2 : Define el tipo de segmento raíz con el nombre de CURSO y con una longitud de 256 bytes.

3 a 5 : Definen los tipos de campos de CURSO. A cada uno se le da un nombre, una longitud en bytes y una posición de inicio dentro del segmento. El primer campo CURSO#, se define (mediante la especificación SEQ) como el campo de secuencia para el segmento, lo que significa que dentro de la PDB de educación, las ocurrencias de PDBR estarán secuenciadas en orden ascendente de número de curso.

6 : Define a REQUISITO como un segmento de 36 bytes dependiente de CURSO.

7 a 8 : Definen los campos de REQUISITO. Otra vez se define un campo CURSO#, el cual se define como el campo de secuencia para REQUISITO. Esto significa que para cada ocurrencia del padre (CURSO), las ocurrencias de este hijo (REQUISITO) estarán secuenciadas en orden ascendente de número de curso.

```
 1 DBD     NAME=PDBDEDUC
 2 SEGM    NAME=CURSO,BYTES=256
 3 FIELD   NAME=(CURSO#,SEQ),BYTES
 4 FIELD   NAME=TITULO,BYTES=33,START=1
 5 FIELD   NAME=DESCRIPN,BYTES=220,START=4
 6 SEGM    NAME=REQUISITO,PARENT=CURSO,BYTES=36
 7 FIELD   NAME=(CURSO#,SEQ),BYTES=3,START=1
 8 FIELD   NAME=TITULO,BYTES=33,START=4
 9 SEGM    NAME=OFERTA,PARENT=CURSO,BYTES=20
10 FIELD   NAME=(FECHA,SEQ,M),BYTES=6,START=1
11 FIELD   NAME=SITIO,BYTES=12,START=7
12 FIELD   NAME=FORMA,BYTES=2,START=19
13 SEGM    NAME=PROFESOR,PARENT=OFERTA,BYTES=24
14 FIELD   NAME=(EMP#,SEQ),BYTES=6,START=1
15 FIELD   NAME=NOMBRE,BYTES=18,START=7
16 SEGM    NAME=ESTUDNTE,PARENT=OFERTA,BYTES=25
17 FIELD   NAME=(EMP#,SEQ),BYTES=6,START=1
18 FIELD   NAME=NOMBRE,BYTES=18,START=7
19 FIELD   NAME=CALIF,BYTES=1,START=25
```

FIG.4.6 DBD (parte del esquema conceptual) PARA LA PDB DE EDUCACION.

9 : Se define a OFERTA como un hijo de CURSO y 20 bytes de longitud .

10 a 12 : Se definen los campos de OFERTA. Aquí también, FECHA se define como el campo de secuencia para OFERTA. La especificación M(Múltiple) significa que ocurrencias gemelas de OFERTA pueden contener el mismo valor de este dato.

13 a 15 : Definen el segmento PROFESOR, como un hijo de OFERTA y sus campos.

16 a 19 : Definen el segmento ESTUDNTE, como un hijo de OFERTA y sus campos.

La secuencia de las preposiciones en la DBD es muy importante, las proposiciones SEGM deben aparecer en la secuencia que refleja la estructura jerárquica (de arriba a abajo y de izquierda a derecha); además cada proposición SEGM debe estar seguida inmediatamente por las proposiciones FIELD necesarias.

Algunos puntos adicionales :

-La especificación de secuencia es optativa, excepto por lo que se anota a continuación.

-Cuando se especifica un campo de secuencia, se toma como único, a menos que se especifique M (Múltiple). Esto significa que al ser único, no pueden tener el mismo valor para el campo de secuencia, dos ocurrencias del tipo de segmento a que se refiera.

-Se requiere un campo de secuencia único para el segmento raíz en *HISAM* e *HIDAM*. (Estos conceptos se verán mas adelante, por ahora solo se mencionará que son dos de las 4 estructuras diferentes de almacenamiento proporcionadas por IMS).

-En la regla anterior no se especifica qué ocurre si el campo de secuencia se omite o no es único. En tal caso se necesitan especificaciones adicionales en la DBD, y se puede requerir programación adicional por parte del usuario cuando deba insertarse un nuevo segmento. En algunas situaciones, la falta de un campo de secuencia único puede generar dificultades lógicas serias.

-La proposición FIELD para el campo de secuencia, en caso de haberlo, debe ser la primera preposición de este tipo para el segmento.

-Se pueden definir campos que se traslapen, esto permite que la combinación de varios campos (contiguos) se defina como el campo de secuencia. Por ejemplo, se puede definir un campo del segmento CURSO que se llame CURSO#N (Bytes=2,Start=2) que represente los caracteres (numéricos) segundo y tercero del campo CURSO#.

-La proposición FIELD puede incluír como opción la especificación *TYPE = tipo de datos*, donde *tipo de datos* puede ser : C-caracter, X-Hexadecimal o P-decimal empaquetado. C se supone por omisión.

4.2.3. SECUENCIA JERARQUICA

El concepto de secuencia jerárquica dentro de una base de datos, es muy importante. En términos formales se puede definir como sigue:

"PARA CADA OCURRENCIA DE SEGMENTO SE DEFINE QUE EL 'VALOR DE LA LLAVE DE SECUENCIA JERARQUICA' SE COMPONE DEL VALOR DEL CAMPO DE SECUENCIA PARA ESE SEGMENTO, PREFIJADO CON EL CODIGO DE TIPO PARA ESE SEGMENTO[3], PREFIJADO CON EL VALOR DE LA LLAVE DE SECUENCIA JERARQUICA DE SU PADRE, SI LO TIENE.

Por ejemplo, el valor de la llave de secuencia jerárquica de la ocurrencia de ESTUDNTE para "Beltrán, W." (Veáse figura 4.5) es:

1M2337308135102141

Para poder entender el ejemplo se explicará a continuación lo que significa la cadena anterior:

[3].-IMS IDENTIFICA INTERNAMENTE A CADA TIPO DE SEGMENTO POR SU POSICION EN LA ESTRUCTURA JERARQUICA, ASI, EN LA PDB DE EDUCACION: CURSO TIENE CODIGO DE TIPO 1, REQUISITO TIENE CODIGO DE TIPO 2, OFERTA DE 3, PROFESOR DE 4 Y ESTUDNTE DE 5

102141 : Valor del campo de secuencia de la ocurrencia de ESTUDNTE.

5 : Código de tipo de ESTUDNTE.

730813 : Valor del campo de secuencia de la ocurrencia de OFERTA (padre de ESTUDNTE).

3 : Código de tipo de OFERTA.

M23 : Valor del campo de secuencia de la ocurrencia de CURSO (padre de OFERTA).

1 : Código de tipo de CURSO.

Por lo tanto, la secuencia jerárquica para una base de datos IMS es la secuencia de ocurrencias de segmento, definida por los valores ascendentes de la llave de secuencia jerárquica.

4.3.- ESTRUCTURA LOGICA DE LA BASE DE DATOS.

4.3.1. PRIMERA DEFINICION DE BASE DE DATOS LOGICA.

Anteriormente se mencionó que el término *base de datos lógica* tiene dos significados distintos en IMS. El primer significado se tratará a continuación.

Una base de datos lógica (LDB) es un conjunto ordenado cuyos elementos se componen de todas las ocurrencias de un tipo de registro de base de datos lógica (LDBR).

Un tipo de registro de base de datos lógica es un arreglo jerárquico de tipos de segmento; esta jerarquía se deriva de la jerarquía del PDBR correspondiente de acuerdo con las siguientes reglas:

1.- Cualquier tipo de la jerarquía del PDBR, junto con todos sus dependientes, se puede omitir de la jerarquía del LDBR. Esta regla implica que la raíz de la LDB debe ser igual a la raíz de la PDB.

2.- Los campos de un tipo de segmento del LDBR pueden ser un subconjunto de los del tipo de segmento correspondiente del PDBR, y pueden reordenarse dentro de ese tipo de segmento del LDBR.

SEGMENTOS SENSIBLES

El concepto de segmento sensible protege al usuario contra ciertos tipos de crecimiento de la PDB.

Los tipos de segmentos de la PDB incluídos en la LDB son *sensibles*. Para el usuario de la LDB no existirá ningún otro segmento. Pero si el usuario suprime un segmento sensible o una ocurrencia de un segmento sensible, todos los hijos de ese segmento se suprimirán también, sin importar si son sensibles o no. En la práctica, a un usuario supuestamente no se le debe dar autoridad para suprimir un segmento.

Se puede añadir un nuevo tipo de segmento (como el hijo de un segmento existente) en cualquier punto, siempre que no afecte ninguna asociación padre-hijo. Este segmento nuevo no será sensible para ningún usuario.

El concepto de segmento sensible también ofrece cierto grado de control sobre la seguridad de los datos, pues a los usuarios se les puede impedir accesar tipos de segmentos particulares mediante la omisión de esos segmentos de la LDB.

CAMPOS SENSIBLES

Los campos sensibles son los campos de la PDB que se incluyen en la LDB. Por definición, todo campo sensible debe estar contenido dentro de un segmento sensible.
La sensibilidad de campos, al igual que la sensibilidad de segmentos, protege al usuario contra ciertos tipos de crecimiento de la base de datos y proporciona un nivel sencillo de seguridad de los datos.

EL BLOQUE DE COMUNICACION DEL PROGRAMA

Cada base de datos lógica se define por medio de un bloque de comunicación de programa (PCB). El PCB incluye una capacitación de la correspondencia entre la LDB y la respectiva PDB. Una PCB se escribe usando macroproposiciones especiales del Lenguaje Ensamblador del Sistema /370. Estas proposiciones constituyen en *DDL-externo* para IMS.

El conjunto de todos los PCB's para un usuario específico forma el bloque de especificación de programa (PSB) de ese usuario; la forma objeto del PSB se almacena en una biblioteca del sistema, de la cual se puede extraer cuando lo requiera el programa de control de IMS.

La figura 4.8 muestra el PCB para LDB de la figura 4.7.; de nuevo a las proposiciones se les ha asignado números como puntos de referencia para las explicaciones que siguen.

1 : Especifica que este es un PCB de base de datos, que la DBD para la base de datos física subyacente se llama PDBDEDUC y que longitud del área de realimentación de llaves es de 15 bytes. Este punto requiere explicación: uno de los campos en el PCB es el área de

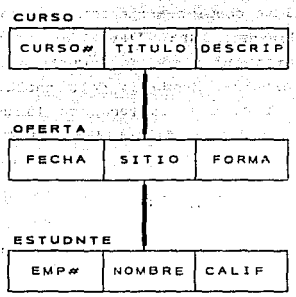


FIG.4.7 TIPO DE LDBR DE MUESTRA PARA LA BASE DE DATOS DE EDUCACION.

```
1 PCB     TYPE=DB,DBDNAME=PDBDEDUC.KEYLEN=15
2 SENSEG  NAME=CURSO,PROCOPT=G
3 SENSEG  NAME=OFERTA,PARENT=CURSO,PROCOPT=G
4 SENSEG  NAME=ESTUDNTE,PARENT=OFERTA,PROCOPT=G
```

FIG.4.8 PCB PARA LA LDB DE LA FIGURA 4.7

realimentación de llaves. Cuando el usuario recupera un segmento de la LDB, IMS coloca una "llave completamente concatenada" en el área de realimentación de llaves. Esta llave se compone de la concatenación de los valores del campo de secuencia de todos los segmentos en la ruta jerárquica, desde la raíz hasta el segmento recuperado; por ejemplo, si el usuario recupera la ocurrencia de ESTUDNTE para "BELTRAN, W." (véase figura 4.5), IMS colocará el valor:

M23730813102141

en el área de realimentación de llaves. (Nótese que la llave completamente concatenada de un segmento no es idéntico a la *llave de secuencia jerarquica*, pues no incluye la información del código de tipo del segmento. Se debe calcular la longitud máxima de una llave completamente concatenada (considerando todos las rutas jerárquicas de la LDB) y citarla en la entrada KEYLEN de la proposición PCB. En el ejemplo ese valor es 15 (3 para CURSO#, más 6 para FECHA, más 6 para EMP#).

2 : Especifíca el primer segmento sensible (la raíz) de la LDB. El nombre de un segmento sensible debe ser igual al nombre del segmento en la DBD. La entrada PROCOPT (opciones de procesamiento) especifíca los tipos de operación que al usuario se le permitirá hacer sobre ese segmento (G=obtener, I=insertar, R=reemplazar y D=suprima). Se puede especificar G,I,R,D o todos en cualquier orden.

3 : Define el segmento sensible de la LDB. Al igual que en la DBD, los segmentos se especifican de arriba a abajo y de izquierda a derecha. PARENT especifica el segmento padre (el cual debe aparecer como se definió en la DBD). El tipo de operación es sólo de *obtener*.

4 : Define el último segmento sensible.

En la figura 4.7, PROCOPT es igual para cada uno de los tres segmentos sensibles; por lo tanto se puede especificar PROCOPT en la proposición PCB en lugar de hacerlo en cada proposición SENSEG; si se especifica en las dos proposiciones, SENSEG anula la entrada PCB.

Otras opciones de PROCOPT son :

- L (*cargar*), que se puede especificar solamente en PCB y no se puede anular.
- K (*sensibilidad a la llave*), sólo se puede especificar en la proposición SENSEG. Se usa cuando el diseñador tiene que agregar un segmento que el usuario del PCB en realidad no requiere pero que sin embargo es necesario.

4.3.2. SEGUNDA DEFINICION DE LDB.

Como se había mencionado anteriormente, la segunda definición de *base de datos lógica* es quizá más importante que la primera.

Una base de datos lógica (LDB) es un conjunto ordenado de ocurrencias del registro de la base de datos lógica (LDBR).

Un LDBR es un arreglo jerárquico de segmentos de longitud fija. Al igual que una PDB, una LDB se define por medio de una DBD.

Una LDB difiere de una PDB en que no tiene existencia por derecho propio, sino que se define en términos de una o más PDB's existentes, de esta manera, la LDB ofrece al usuario una vista alterna de los datos.

Decir que la LDB no existe como tal no es tan verdadero, pues la estructura de almacenamiento de hecho representa la LDB en forma directa por cuanto contiene apuntadores adicionales que ligan a los segmentos almacenados en la estructura deseada; pero lo importante es que los datos en realidad pertenecen a una o más PDB's, y no a la LDB. Dado que es parte del nivel conceptual, lo ideal es que el usuario vea a la LDB como una PDB.

Como ejemplo, considérese que la sección de ornitología de un instituto de conservación de la naturaleza está realizando un estudio sobre la vida de las aves de una región particular. Esta se divide en áreas y para cada área se van a registrar observaciones en una base de datos del estudio.

La información registrada para un área se compone del número del área (único), el nombre del área, la descripción del área y para cada tipo de ave observada, el nombre común, la fecha de observación, las notas del observador, el nombre científico del ave e información descriptiva. De esta manera, el registro de la base de datos, como lo ve el usuario lo muestra la figura 4.9.

Sin embargo, el instituto ya mantiene una PDB con información sobre las aves; esta PDB se muestra en la figura 4.10.

Por lo tanto, si se armara una PDB nueva con la estructura de la figura 4.9, muchos de los datos serían repetitivos. Esto se puede evitar por medio de una base de datos lógica. Primero se define una base de datos *física* como la de la figura 4.9; pero el segmento OBSERVACION solo contiene los campos FECHA y NOTAS, junto con un *apuntador* al segmento AVE adecuado en la PDB de AVES. De esta manera, se tiene la situación mostrada en la figura 4.11.

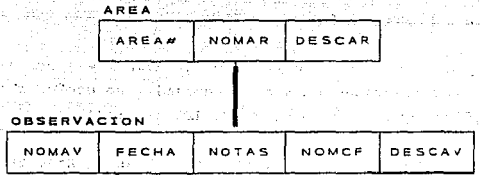


FIG. 4.9 ESTRUCTURA DE REGISTRO REQUERIDA PARA LA BASE DE DATOS DEL ESTUDIO.

FIG. 4.10 ESTRUCTURA DE REGISTRO DE LA BASE DE DATOS DE AVES.

Ahora, se puede definir una base de datos *lógica* con la estructura de la figura 4.12, la cual indica lo que el usuario ve. El usuario puede procesar esta LDB como si fuera una PDB.

Para que una LDB pueda definirse en términos de una o más PDB's, esas PDB's deben estar definidas de manera adecuada, es decir, sus DBD's deben especificar que intervienen en la LDB en cuestión.

El *segmento apuntador* OBSERVACION de la figura 4.11 es un hijo del segmento AREA, también se considera hijo del segmento AVE. Entonces, OBSERVACION tiene dos segmentos padre. Para distinguirlos se hace referencia a AREA como el padre *físico* y a AVE como el padre *lógico;* de igual manera, se hace referencia a OBSERVACION como hijo físico de AREA y como hijo lógico de AVE. Un segmento dado puede tener a lo sumo un padre físico y un padre lógico.

La terminología padre lógico/hijo lógico, padre físico/hijo físico, se aplica tanto a tipos como a ocurrencias.

DESCRIPCIONES DE BASES DE DATOS. (DBD's)

Una base de datos lógica, al igual que una base de datos física, se define por medio de una descripción de base de datos (DBD). Se dice que esta DBD es una DBD *lógica*. Cada DBD lógica se define en términos de una o más DBD's *físicas* subyacentes, que ya deben existir. Así, en el ejemplo se requerirá de:

a) Una DBD física para la PDB AVE.
b) Una DBD física para la PDB AREA.
c) Una DBD lógica para la LDB ESTUDIO.

La figura 4.13 ilustra de manera general la DBD para la PDB AVE. La proposición SEGM para el segmento AVE va seguida por una proposición LCHILD que especifica al segmento OBSERVACION como hijo lógico de AVE (si la proposición SEGM para el tipo de segmento S va seguido por una proposición LCHILD, significa que S *posee* un hijo lógico, no que S *es* un hijo lógico).

La figura 4.14 ilustra de manera general la DBD para la PDB AREA. Considérese primero la entrada POINTER para el segmento OBSERVACION, la cual especifica que el prefijo de observación va a contener los apuntadores siguientes :

- Un apuntador al padre lógico.
- Un apuntador a gemelo físico.

Se puede especificar TWIN siempre que el segmento observación se halle en HDAM o HIDAM (estos conceptos se estudiarán en el siguiente tema).

Considérese ahora la entrada PARENT para el segmento OBSERVACION que especifica :

a) que el padre físico de este segmento es AREA y
b) que el padre lógico de este segmento es AVE, el cual se define en la DBD para la PDB AVE.

VIRTUAL especifica que la llave completamente concatenada del padre lógico es un campo virtual (el campo NOMAV) en lo que respecta a este segmento, es decir, no está almacenada físicamente como padre del segmento. La artenativa a VIRTUAL es PHYSICAL, que significa que una copia del campo NOMAV se va a almacenar físicamente como parte del segmento OBSERVACION.

```
DBD      NAME=PDBDAVE,...
SEGM     NAME=AVE,POINTER=TWIN,...
LCHILD   NAME=(OBSERVACION,PDBDAREA)
(PROPOSICIONES FIELD PARA AVE)
```

FIG.4.13 DBD PARA LA PDB AVE (esbozo).

```
DBD      NAME=PDBDAREA,...
SEGM     NAME=AREA,POINTER=TWIN,...
(PROPOSICIONES FIELD PARA AREA)
SEGM     NAME=OBSERVACION,POINTER=(LPARNT,TWIN),
         PARENT=((AREA),(AVE,VIRTUAL,PDBAVE)),...
FIELD    NAME=NOMAV,...
FIELD    NAME=FECHA,...
FIELD    NAME=NOTAS,...
```

FIG.4.14 DBD PARA LA PDB AREA (esbozo).

```
DBD      NAME=LDBDESTD,ACCESS=LOGICAL
DATASET  LOGICAL
SEGM     NAME=AREA,SOURCE=((AREA,PDBDAREA))
SEGM     NAME=OBSERVACION,PARENT=AREA
         SOURCE=((OBSERVACION,,AREAPDBD),AVE,,PDBDAVE)
```

FIG.4.15 DBD PARA LA LDB ESTUDIO.

La figura 4.15 ilustra la DBD para la LDB ESTUDIO. ACCESS = LOGICAL debe especificarse en la proposición DBD, LOGICAL también debe especificarse en la proposición DATASET. La primera proposición SEGM estipula que el segmento raíz de la LDB, AREA, es en realidad el segmento AREA definido en la DBD para la PDB AREA (si se desea, al segmento se le puede dar un nombre diferente dentro de la LDB).

La segunda proposición SEGM estipula que dentro de esta LDB, OBSERVACION es dependiente de AREA, y se compone de la concatenación del segmento OBSERVACION definido en la DBD para la PDB AREA con el segmento AVE definido en la DBD para la PDB AVE.

CARGA DE LA BASE DE DATOS LOGICA.

El proceso de cargar una base de datos lógica consiste escencialmente en cargar la(s) base(s) de datos física(s) subyacente(s) y establecer los apuntadores requeridos. Esta operación se realiza de modo directo sobre la(s) PDB(s) subyacente(s), y no sobre la LDB como tal. El procedimiento que se sigue en realidad consiste en cargar la LDB insertando segmentos directamente en la(s) PDB(s) subyacente(s), es decir, cargando cada una de las PDB's subyacentes como una operación separada.

PROCESAMIENTO DE LA BASE DE DATOS LOGICA.

Una vez cargada la LDB, el usuario puede procesarla exactamente como si fuera una PDB, al menos en lo que respecta a las operaciones de recuperación. Sin embargo, las operaciones de actualización son más complicadas. En general, el efecto de una operación de actualización sobre un segmento que participa en una relación lógica lo define la *regla* especificada por el DBA para esa operación y ese segmento. Al DBA se le exige especificar en la proposición SEGM de la DBD física adecuada, una regla de inserción, una regla de supresión y una regla de reemplazo para cada uno de tales segmentos.

Cada una de estas reglas en general se puede especificar como *física, lógica* o *virtual*. Por las reglas especificadas se rigen *todas* las operaciones de actualización sobre el segmento; la operación en cuestión se puede emitir realmente :

a) de modo directo contra la PDB pertinente o
b) indirectamente contra alguna LDB construída encima de la PDB.

NOTAS.

1.-La regla de inserción para un segmento hijo lógico siempre se dá como virtual.

2.-La regla de supresión para un segmento padre físico siempre se dá como virtual.

3.-La estructura de la LDB permite la redundancia en la vista que tiene el usuario de los datos, sin exigir una redundancia correspondiente en lo que realmente se almacena.

ALGUNAS REGLAS Y REESTRICCIONES.

1.-La raíz de una LDB también debe ser raíz de una PDB.

2.-Un segmento hijo lógico debe tener un padre físico y un padre lógico.Por lo tanto una raíz no puede ser un hijo lógico, IMS no puede soportar directamente *asociaciones n-direccionales* para n>2.

3.-Un hijo físico de un hijo lógico puede aparecer como dependiente del segmento concatenado (hijo lógico/padre lógico)en la LDB.

4.- Un hijo físico de un padre lógico puede aparecer como dependiente del segmento concatenado (hijo lógico/padre lógico) en la LDB, siempre que el padre lógico sea el primer padre lógico encontrado en la definición de la ruta jerárquica de la LDB [4][4].

5.- Un padre físico de un padre lógico puede aparecer como dependiente del segmento concatenado (hijo lógico/padre lógico) en la LDB.

6.- Si X e Y son dos segmentos en la ruta jerárquica de una LDB, todos los segmentos recorridos en la ruta entre X e Y de la(s) PDB(s) subyacente(s) también deben incluírse (en las mismas posiciones relativas) en la ruta jerárquica de esa LDB.

4.4.- LENGUAJES DE DEFINICION Y MANIPULACION DE DATOS.

4.4.1. MANIPULACION DE DATOS EN IMS.

BLOQUE DE COMUNICACION DEL PROGRAMA (PCB).

Como se mencionó anteriormente, cuando un programa de aplicación está operando sobre una LDB particular, el PCB para esa LDB se mantiene en almacenamiento para que sirva de área de comunicación entre el programa e IMS.

IMS determina el PSB y la(s) DBD(s) que se requiere(n), las trae de sus repectivas bibliotecas y las carga en el almacenamiento. IMS luego trae el programa de aplicación y le da control, pasándole las direcciones de PCB como parámetros.

[4] 4.-ESTA RESTRICCION SE DEBILITA PARA UNA LDB DEFINIDA EN TERMINOS DE UNA SOLA PDB.

```
DLITPLI: PROCEDURE(DIR_PCBCOE) OPTIONS(MAIN);
         .
         .
         .
   DECLARE 1 PCBCOE       BASED(COSPCB_ADDR),
           2 NOMDBD       CHARACTER(8),
           2 NIVSEG       CHARACTER(2),
           2 ESTADO       CHARACTER(2),
           2 OPSPROC      CHARACTER(4),
           2 RESERVADO    FIXED BINARY(31),
           2 NOMSEG       CHARACTER(8),
           2 LONGLLAVCC   FIXED BINARY(31),
           2 #SEGSSEN     FIXED BINARY(31),
           2 AREALLAVCC   CHARACTER(15);
```

FIG.4.16 EJEMPLO DE INTRODUCCION DE PROGRAMA Y DEFINICION DE PCB (EN PL/I).

GET UNIQUE (GU)	RECUPERACION DIRECTA
GET NEXT (GN)	RECUPERACION SECUENCIAL
GET NEXT WITHIN PARENT (GNP)	RECUPERACION SECUENCIAL BAJO EL PADRE ACTUAL
GET HOLD (GHU,GHN,GHNP)	COMO ARRIBA PERO PERMITE EM SEGUIDA DLET/REPL
INSERT (ISRT)	ADICIONE SEGMENTO NUEVO
DELETE (DLET)	SUPRIMA SEGMENTO EXISTENTE
REPLACE (REPL)	REEMPLACE SEGMENTO EXISTENTE

FIG.4.17 OPERACIONES DE DL/I (resumen)

```
GU CURSO      (TITULO='DINAMICA')
   OFERTA     (FORMA='F1'|FORMA='3')
   ESTUDNTE   (CALIF='A')
```

FIG.4.18 EJEMPLO DE LA SINTAXIS SIMPLIFICADA DE DL/I.

Para que el programa de aplicación pueda accesar la información en el PCB para una LDB en particular, debe contener una definición de ese PCB.

Por ejemplo, supóngase que se tiene una aplicación en PL/I que opera sobre la LDB de cursos, ofertas y estudiantes de la figura 4.7 y que ésta es la única LDB que se usa en ésta aplicación. Entonces parte del programa puede asemejarse a la figura 4.16.
La explicación de la figura 4.16 es la siguiente:

- El punto de entrada del programa es la proposición PROCEDURE (rotulada DLITPLI). El nombre DLITPLI es obligatorio (para PL/I); todos los otros nombres indicados (PCBCOE,NOMDBD,NIVSEG,etc.) son arbitrarios.

- (DIR-PCBCOE) representa los parámetros que ha de pasarle IMS al programa, en general, se compondrá de una lista de apuntadores, uno para cada PCB en el PSB, donde el primer apuntador da la dirección del primer PCB, y así sucesivamente. En el ejemplo sólo hay un PCB y, por lo tanto, un solo apuntador en la lista.

- DECLARE define una estructura (con nombre PCBCOE) para representar el único PCB usado en esta aplicación. Esta estructura está basada en el apuntador DIR-PCBCOE.

- El campo NOMDBD contiene el nombre de la DBD subyacente (en el ejemplo, PDBDEDUC) durante toda la ejecución del programa.

- El campo NIVSEG contiene el número de nivel de segmento del segmento recién accesado (donde el segmento raíz se considera que es de nivel 1, sus hijos de nivel 2 y así sucesivamente).

- El campo ESTADO es el más importante en el PCB en lo que respecta al usuario, porque después de todas las llamadas de DL/I (que es el Lenguaje de Manipulación de Datos de IMS), se coloca un valor de dos caracteres en este campo para indicar el éxito o fracaso de la operación solicitada. Un valor en blanco indica que la operación fué completada con éxito, cualquier otro valor representa una condición de error.

- El campo DPSPROC contiene el valor de PROCOPT, como se especificó en la proposición PCB cuando éste fue originalmente definido.

- RESERVADO es un campo reservado para uso propio de IMS.

- NOMSEG contiene el nombre del último segmento accesado.

- LONGLLAVCC contiene la longitud actual significativa de la llave completamente concatenada en el área de realimentación de llaves.

- SEGSSEN contiene una cuenta del número de segmentos sensibles.

- AREALLAVCC es el área de realimentación de llaves; contiene la llave completamente concatenada del último segmento accesado.

OPERACIONES DE DL/I.

La aplicación invoca una operación de DL/I por medio de una llamada a subrutina, uno de cuyos parámetros consiste en la dirección del PCB adecuado. El nombre de la subrutina lo determina IMS; para una aplicación en PL/I es PLITDLI.

Si se supone que los datos son los mostrados en la figura 4.5, el resultado de la operación de DL/I de la figura 4.18 es recuperar el segmento ESTUDNTE para "GUZMAN,O.".

GU (*obtenga único*) siempre ocasiona una búsqueda secuencial hacia adelante desde el principio de la base de datos (al menos en términos conceptuales). El segmento recuperado será el que satisfaga los tres SSA's (SSA:*Argumento de Búsqueda de Segmento*). Los SSA's se estudiarán con mayor detalle más adelante.

En el ejemplo, los tres SSA's representan la ruta jerárquica hacia el segmento deseado. El efecto de éstos SSA's es hacer que IMS busque hacia adelante la primera ocurrencia de CURSO que contenga un valor de TITULO igual a "DINAMICA", luego busca en las ocurrencias de OFERTA subordinadas de ese curso para encontrar la primera que contenga un valor de FORMA de "F" o "F3", y busca después en las ocurrencias de ESTUDNTE subortdinadas a esa OFERTA para encontrar la primera que contenga un valor 'A' de CALIF.

Si no existe el ESTUDNTE para esa OFERTA, se escudriñarán los ESTUDNTE's de la siguiente OFERTA de la FORMA "F1" O "F3" de este curso, y así sucesivamente.

En caso de que no existan más OFERTA's de la FORMA "F1" o "F3", IMS buscará la siguiente ocurrencia de CURSO que contenga un valor de TITULO de "DINAMICA" y repetirá el proceso (si no hay otro curso así, la operación de recuperación fallará y al usuario se le devolverá un estado no blanco).

En general, un SSA se compone de un nombre del segmento, seguido como opción por una condición. Si se omite la condición, cualquier ocurrencia del segmento indicado satisfará este SSA.

Si la condición se incluye, debe constar de un conjunto de expresiones de comparación unidos por medio de los operadores booleanos "y" y "o".

Cada comparación se compone de: un campo, un operador de comparación y un valor, donde el campo debe pertenecer al segmento especificado y el operador de comparación puede ser cualquiera de los usuales (=,=,<,<=,>,>=). Todas las operaciones de comparación, IMS las realiza bit a bit y de izquierda a derecha.

Las operaciones *obtenga único* e *inserte* requieren SSA's que especifiquen la ruta jerárquica entera, desde la raíz hacia abajo; las operaciones obtenga *obtenga siguiente* y *obtenga siguiente dentro del padre* pueden requerir o no SSA's; y las operaciones *suprima* y *reemplace* no implican SSA's en absoluto.

A continuación se verán algunos ejemplos donde se supondrá que la LDB es idéntica a la PDB del segundo capítulo, es decir, todos los segmentos y campos son sensibles.

a) RECUPERACION DIRECTA.-Obtenga la primera ocurrencia de OFERTA en donde el sitio sea ESTOCOLMO.

```
        GU      CURSO
                OFERTA (SITIO='ESTOCOLMO')
```

b) RECUPERACION SECUENCIAL CON UN SSA.-Obtenga todas las ocurrencias de ESTUDNTE en la LDB, a partir del primer estudiante para la oferta hallada en el ejemplo anterior.

```
        GU      CURSO
                OFERTA (SITIO='ESTOCOLMO')
                ESTUDNTE
NS      GN      ESTUDNTE
        Vaya a  NS
```

Obtenga único (GU), establece una posición actual dentro de la base de datos. La primera vez que se ejecuta *obtenga siguiente* recupera el primer estudiante después de ésta posición y establece una nueva posición actual; la segunda vez, recupera el siguiente y así sucesivamente.

c) **RECUPERACION SECUENCIAL CON UN SSA CONDICIONAL.**-Resolver como en el ejemplo anterior, excepto que sólo se van a recuperar las ocurrencias de ESTUDNTE con una calificación de A.

```
        GU      CURSO
                OFERTA (SITIO='ESTOCOLMO')
                ESTUDNTE (CALIF='A')
NSA     GN      ESTUDNTE (CALIF='A')
        vaya a NSA
```

d) **RECUPERACION SECUENCIAL CON UN SSA.**-Resolver como en el ejemplo anterior, excepto que la búsqueda empezará al principio de la base de datos.

```
        GU      CURSO
NX      GN      ESTUDNTE (CALIF='A')
        vaya a NX
```

e) **RECUPERACION SECUENCIAL CON MULTIPLES SSA's CONDICIONALES.**- Resolver como en el ejemplo c), excepto que van a recuperar solamente las ocurrencias de ESTUDNTE con una calificación de A para las ofertas de ESTOCOLMO.

```
        GU      CURSO
                OFERTA (SITIO='ESTOCOLMO')
                ESTUDNTE (CALIF='A')
NY      GN      OFERTA (SITIO='ESTOCOLMO')
                ESTUDNTE (CALIF='A')
                vaya a NY
```

f) **RECUPERACION SECUENCIAL SIN SSA's.**-Obtenga todos los segmentos.

```
        GU      CURSO
NZ      GN
        vaya a NZ
```

g) **RECUPERACION SECUENCIAL CON UN SSA DENTRO DE UN PADRE.**-Obtenga todos los estudiantes para la oferta del 13 de agosto de 1973 del curso M23.

```
        GU      CURSO (CURSO#='M23')
                OFERTA (FECHA='730813')
NP      GNP     ESTUDNTE
        vaya a NP
```

GNP es igual a GN, excepto cuando se hayan recuperado todos los segmentos que satisfagan los SSA's para el padre actual, el siguiente intento de ejecución de GNP regresará un estatus que indique este hecho.

h)RECUPERACION SECUENCIAL CON UN SSA CONDICIONAL DENTRO DE UN PADRE.-Obtenga todos los estudiantes que lograrán una calificación de A en el curso M23 (cualquier oferta).

```
        GU      CURSO (CURSO#='M23')
NQ      GNP     ESTUDNTE (CALIF='A')
        vaya a NQ
```

Cualquier segmento antecesor puede servir de "padre" para una operación GNP.

i)RECUPERACION SECUENCIAL SIN SSA's DENTRO DE UN PADRE.-Obtenga todos los segmentos subordinados para el curso M23.

```
        GU CURSO (CURSO#='M23')
NN      GNP
        vaya a NN
```

j)INSERCION DE SEGMENTOS.-Adicione una ocurrencia nueva de ESTUDNTE para la oferta del 13 de agosto de 1973 del curso M23.

```
        arme el segmento nuevo en el área de E/S
        ISRT CURSO (CURSO#='M23')
             OFERTA (FECHA='730813')
             ESTUDNTE
```

IMS introducirá la nueva ocurrencia en la posición correcta que es definida por el valor de su campo de secuencia (en éste caso EMP#).

k)SUPRESION DE SEGMENTOS.-Suprima la oferta del curso M23 del 13 de agosto de 1973.

```
GHU CURSO (CURSO#='M23')
     OFERTA (FECHA='730813')
DLET
```

El segmento que se va a suprimir, primero debe recuperarse por medio de una de las operaciones *agarre* (GHU, GHN o GHNP,. Recuérdese que una operación *suprima* exitosa suprime la ocurrencia de segmento especificada y también todos sus hijos.

1)REEMPLAZO DE SEGMENTOS.-Cambie el sitio de la oferta del 13 de Agosto de 1973 del curso M23 a Helsinki.

```
GHU     CURSO (CURSO#='M23')
        OFERTA (FECHA='730813')
Cambie el sitio a HELSINKI en el area de E/S
REPL
```

El segmento que se va a reemplazar, primero debe recuperarse por medio de una de las llamadas *agarre*. Luego se modifica en el área de E/S y se emite la operación de *reemplace*. El campo de secuencia no se puede modificar (ésto sucede también con suprime).

En los ejemplos anteriores se utilizó la sintaxis *hipotética* para simplificar; en la figura 4.19 se muestra la sintaxis *genuina* de DL/I. La codificación mostrada corresponde a la primera llamada de DL/I (GU) del ejemplo a).

CONSTRUCCION DEL ARGUMENTO DE BUSQUEDA DE UN SEGMENTO (SSA).

El proceso de construcción del SSA es un detalle de IMS, no parte del enfoque jerárquico como tal. Un SSA es en realidad, una hilera de caracteres, que forma uno de los parámetros de una llamada de subrutina.

Un valor típico de un SSA podría ser :

```
'ESTUDNTEb(CALIFbbb=bA)'
```

```
DLITPLI: PROCEDURE(DIR_PCBCOE) OPTIONS(MAIN);

  DCL 1 PCBED BASED(DIREC_PCBED),...;
  DCL AREA_ESTUDNTE CHAR(25);/* AREA DE ENTRADA */

  DCL 1 CSSA,
       2 CSEGNAME CHAR(8) INITIAL('CURSObb'),
       2 CSSAEND  CHAR(1) INITIAL('b');

  DCL 1 OSSA,
       2 OSEGNAME CHAR(8) INITIAL('OFERTAbb'),
       2 OLPAREN  CHAR(1) INITIAL('('),
       2 OFLDNAME CHAR(8) INITIAL(SITIObbb'),
       2 OCOMPOP  CHAR(2) INITIAL('=b'),
       2 OFLDVAL  CHAR(12),
       2 ORPAREN  CHAR(1) INITIAL(')');

  DCL 1 ESSA,
       2 SSEGNAME CHAR(8) INITIAL('ESTUDNTE'),
       2 SLPAREN  CHAR(1) INITIAL('('),
       2 EFLDNAME CHAR(8) INITIAL('CALIFbbb'),
       2 ECOMPOP  CHAR(2) INITIAL('=b'),
       2 EFLDVAL  CHAR(1),
       2 ERPAREN  CHAR(1) INITIAL(')');

  DCL GU CHAR(4) INITIAL('GUbb');
  DCL SEIS FIXED BIN(31) INITIAL(6);

  ........
  OFLDVAL='ESTOCOLMObb',/* EN LA PRACTICA SERIAN VARIABLES,      */
  EFLDVAL='A';              /* NO CONSTANTES                      */
  CALL PLITDLI (SEIS,GU,EDPCB,STUDENT_AREA,/*LLAME A DL/I  /*I
                CSSA,OSSA,SSSA);
  IF PCBED.STATUS='GE' THEN.../* SEGMENTO NO ENCONTRADO */

  ........
  END DLITPLI;
```

FIG.4.19 EJEMPLO DE LA SINTAXIS DE IMS.

Esta sería la sintaxis genuina de IMS, donde se deben usar caracteres en blanco (mostrados como letras b) para que cada parte del SSA tenga una longitud fija predefinida y el valor de comparación no se encierra entre comillas.

CODIGOS DE MANDATO DEL SSA.

Cada código de mandato se representa por un carácter único. Un SSA puede incluír como opción uno o más códigos de mandato.

Los códigos de mandato se especifican escribiendo un asterisco seguido por el (los) caracter(es) adecuado(s) inmediatamente después del nombre de segmento en el SSA. Los códigos de mandato pueden ser : D(tal vez,el más útil del conjunto), F ,C, L, P, Q, U, N.

*D significa datos; en general, el código de mandato D puede especificarse en algunos niveles y no en otros; el efecto es recuperar solamente los segmentos indicados de la ruta jerárquica y concatenarlos en el área de E/S.

*F ocasiona que el IMS principie la búsqueda en la primera ocurrencia del tipo de segmento especificado bajo el padre actual.

Un SSA con un código de mandato V, dirige a IMS a no alejarse del segmento actual del tipo nombrado en el SSA al buscar un segmento que satisfaga la solicitud. *V no se puede usar en el nivel más bajo de una secuencia de SSA's, ni en un SSA que incluya condiciones de calificación sobre los campos del segmento; no obstante, como proporciona casi todas las funciones de GNP y otras cosas más, siempre puede convenir usarlo con preferencia sobre GNP donde sea posible.

USO DE MAS DE UN PCB.

Cada operación de DL/I tiene como uno de sus parámetros la dirección de un PCB. El PCB especificado identifica para IMS la base de datos adecuada y también una *posición actual dentro de esa base de datos*. Es decir, IMS recuerda la *posición actual* registrándola en un campo *oculto* del PCB (es decir, un campo que no es accesible para el usuario), y las operaciones tales como GN hacen uso implícito de este campo oculto.

NIVEL INTERNO DE IMS

IMS proporciona cuatro estructuras diferentes de almacenamiento conocidas como HSAM, HISAM, HDAM e HIDAM, y una base de datos almacenada puede estar en cualquiera de los cuatro.

Veáse la figura 4.20. IMS usa varios métodos de acceso debajo de la interfaz de registros almacenados. Estos son los siguientes:

-Los Métodos de Acceso Secuenciales de OS/VS (QSAM y BSAM, que se conocen en conjunto como SAM);

-El Método de Acceso Secuencial Indicado de OS/VS (ISAM);

-El Método de Acceso de Almacenamiento Virtual de OS/VS (VSAM);

-Un método de acceso especial de IMS llamado Método de Acceso Secuencial de Desbordamiento (OSAM).

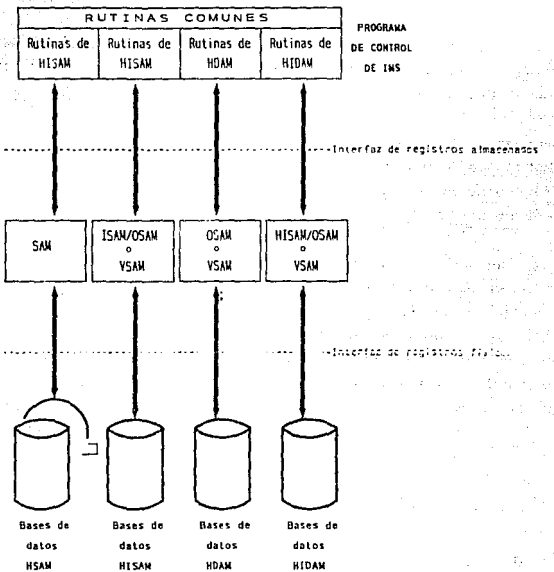


FIG.4.20. METODOS DE ACCESO Y ESTRUCTURAS DE ALMACENAMIENTO DE IMS.

El término *método de acceso* también se usa para las rutinas dentro del programa de control IMS que procesan las cuatro estructuras diferentes. Estos métodos de acceso son como sigue:

El Método de Acceso Secuencial Jerárquico (HISAM);
El Método de Acceso Secuencial Indicado Jerárquico (HISAM);
El Método de Acceso Directo Jerárquico (HDAM);
El Método de Acceso Directo Indicado Jerárquico (HIDAM).

Cada base de datos física (PDB) se representa como una base de datos almacenada en una de las cuatro estructuras. Cada ocurrencia de segmento de la PDB se representa mediante una ocurrencia de segmento *almacenado*, que contiene los datos (exactamente como el usuario los ve) junto con un *prefijo* que el usuario no ve. El prefijo contiene información de control para el segmento : señales de supresión, código de tipo de segmento, apuntadores, etc. La diferencia entre las cuatro estructuras estriba en la forma de representar la secuencia jerárquica de la PDB.

H S A M

Una base de datos HSAM se representa por medio de un único *conjunto de datos* SAM que contiene registros almacenados de longitud fija. Cada registro almacenado puede contener cualquier número de segmentos almacenados; sin embargo, cada segmento almacenado debe estar contenido por completo dentro de un registro almacenado. Esto siginifica que algunos bytes al final de un registro almacenado pueden dejarse sin uso. Cada segmento almacenado va seguido inmediatamente por su sucesor en la secuencia jerárquica.

Una base de datos HSAM se crea mediante una serie de operaciones inserte (los segmentos que se van a cargar deben presentarse en secuencia jerárquica).

Una vez creada la base de datos sólo son válidas las operaciones *obtenga* (GU, GN y GNP, no las operaciones *agarre*). Las actualizaciones se aplican a una versión existente de la base de datos para generar una versión nueva (físicamente separada).

Las operaciones *suprima* y *reemplace* no se pueden usar; pero el usuario puede suprimir un segmento existente no insertándolo en la nueva base de datos, y puede reemplazar un segmento existente modificándolo antes de insertarlo en la nueva base de datos.

H I S A M

HISAM proporciona acceso indicado a los segmentos raíz, y acceso secuencial a los segmentos subordinados. (Esta afirmación no es verdadera después de que se hayan efectuado inserciones y supresiones).

HISAM usa VSAM o una combinación de ISAM y OSAM como método de acceso de soporte.

HISAM que usa ISAM/OSAM.-

Una base de datos HISAM bajo ISAM/OSAM se compone de dos conjuntos de datos, un conjunto de datos ISAM y otro OSAM.

Para ISAM, en ambos conjuntos de datos, los registros almacenados se crean en secuencia y pueden recuperarse secuencial o directamente, y para OSAM por medio de la dirección relativa del registro dentro del conjunto de datos. La figura 4.21 muestra la estructura de uno de estos registros almacenados.

Al cargarse la base de datos, cada segmento raíz hace que se cree un nuevo registro almacenado ISAM; el segmento raíz se coloca al principio de este registro, y después de la raíz se colocan tantos segmentos dependientes como quepan. Si el segmento raíz se llena, entonces se crea un nuevo registro OSAM, y el dependiente se coloca al principio de ese, si el registro OSAM también se llena, se crea otro, y así sucesivamente.

De esta manera; cada ocurrencia de un registro de la base de datos física se representa como una cadena que contiene un registro ISAM junto con cero o más registros OSAM.

La supresión en HISAM se logra poniendo una señal en el prefijo del segmento. Los dependientes del segmento suprimido también se suprimirán. El segmento suprimido continúa ocupando espacio en la base de datos; éste espacio no está disponible para volverse a usar.

Respecto a la inserción; si es una raíz, se crea automáticamente un registro OSAM nuevo y el nuevo segmento se coloca al principio de él; si el nuevo segmento es un subordinado, entonces el método de operación cambia, los segmentos dependientes se insertan en el punto correcto de la secuencia jerárquica. Esto implica hallar el registro que contenga al predecesor del nuevo segmento. Si hay segmentos que siguen a este predecesor, entonces se desplazan a la derecha dentro del registro almacenado para hacer campo para el segmento nuevo.

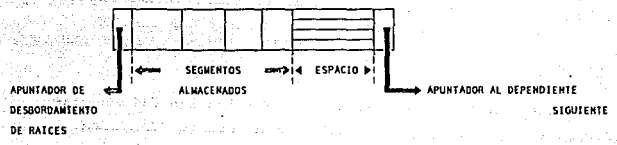


FIG.4.21. ESTRUCTURA DE UN REGISTRO ALMACENADO EN HISAM (usando ISAM/OSAM)

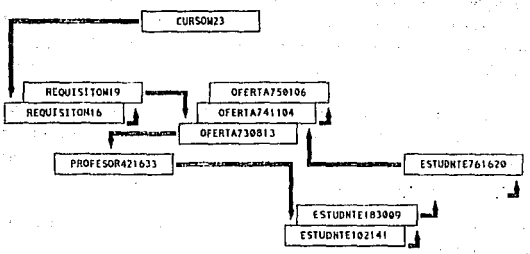


FIG.4.22. EJEMPLO DE APUNTADORES JERARQUICOS

Si se tiene el caso de que uno o más segmentos ya no quepan en el registro; todos estos segmentos desbordados se colocan en el conjunto de datos OSAM.

HISAM usando VSAM.-

Para explicar este punto, se resumen a continuación las diferencias más importantes entre HISAM usando VSAM e HISAM usando ISAM/OSAM.

-Los conjuntos de datos ISAM y OSAM se reemplazan por un conjunto de datos VSAM secuenciado por llave y un conjunto de datos VSAM secuenciado por entrada, respectivamente.

-Todos los segmentos raíz residen en el conjunto de datos secuenciado por llave, incluso los insertados. Cuando se inserta una raíz nueva, se coloca en su posición correcta en este conjunto de datos, y el índice VSAM se actualiza si es necesario.

-En ciertas circunstancias, la supresión de un segmento raíz liberará el espacio de almacenamiento del regsitro y lo contiene (en el conjunto de datos secuenciado por llave) para volverse a usar posteriormente.

El acceso a los segmentos raíz de una base de datos HISAM se hace por medio de un índice (ISAM o VSAM) sobre el campo de secuencia del segmento de raíz; sin embargo, la indicación en general es parcial por las siguientes razones:

-El índice no es denso, por lo que, en general, habrá muchas raíces en cada entrada del índice.
-Además, algunas (inserciones de) raíces pueden existir en el conjunto de datos OSAM en vez del conjunto de datos ISAM si se usa la combinación ISAM/OSAM.

El acceso a los segmentos dependientes es secuencial. Esto puede implicar recorrer los segmentos suprimidos. También puede requerir recorrer cadenas de un registro almacenado a otro.

APUNTADORES: ESTRUCTURAS HD

Las dos estructuras directas HDAM e HIDAM comprenden el uso de apuntadores para ligar los segmentos, por lo que antes de entrar a estudiar éstas estructuras, se considerarán estos apuntadores.

Los apuntadores consisten en desplazamientos en bytes dentro del conjunto de datos pertinente. Se almacenan físicamente como parte del prefijo del segmento, y se usan para:

a) representar la secuencia jerárquica de los segmentos dentro de una ocurrencia del PDBR.

b) representar la secuencia de ocurrencias del PDBR.

La secuencia jerárquica de los segmentos dentro de una ocurrencia del PDBR, se puede representar ya sea por medio de apuntadores jerárquicos o mediante apuntadores de hijo/gemelo. La figura 4.22 muestra una ocurrencia del PDBR donde se usan apuntadores jerárquicos.

Obsérvese que cada segmento incluye un apuntador al siguiente en secuencia jerárquica. Como alternativa, los apuntadores jerárquicos pueden hacerse de doble vía; es decir, cada segmento puede incluír un apuntador extra a su predecesor.

La figura 4.23 muestra cómo aparecería la misma ocurrencia del PDBR si se usarán apuntadores hijo/gemelo. Aquí cada ocurrencia de un segmento padre tiene un apuntador a la primera ocurrencia de cada uno de sus hijos, y cada ocurrencia hijo tiene un apuntador al siguiente gemelo. Estos apuntadores a gemelos también pueden hacerse como opción de doble vía. Además cada padre puede incluír un apuntador a la última ocurrencia (así como a la primera) de cualquiera de sus hijos o de todos.

Respecto a la secuencia de las ocurrencias del PDBR dentro de la base de datos, los apuntadores implicados son en realidad el (los) apuntador(es) a gemelo(s) en el prefijo de la raíz.

En HDAM, todas las raíces que colisionan en una posición K se guardan en una cadena que principia en el registro K y se mantiene en secuencia ascendente de la raíz.

En HIDAM, si se selecciona la opción de una vía, se encadenan todas la raíces dentro de un registro almacenado pero la raíz insertada más reciente estará al principio de la cadena. La cadena es para uso propio de IMS y nunca es directamente accesible al usuario.

En HIDAM, si se selecciona la opción de doble vía, todas las raíces se mantienen con la secuencia correcta sobre la cadena de doble vía, permitiendo así la recuperación secuencial de las raíces sin referencias al índice.

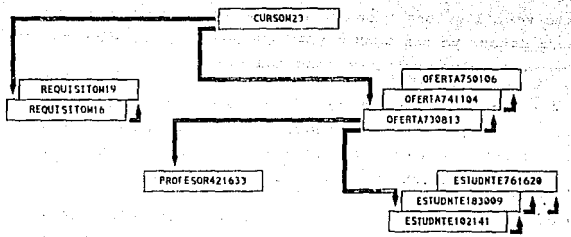


FIG. 4.23. EJEMPLO DE APUNTADORES HIJO/GEMELO

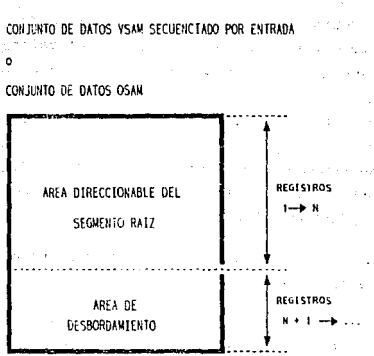


FIG. 4.24. ESTRUCTURA DE UNA BASE DE DATOS HDAM.

H D A M

En su forma más sencilla, una base de datos HDAM se compone de un solo conjunto de datos, ya sea OSAM o VSAM secuenciado por entrada, dividido en registros almacenados de longitud fija.

La figura 4.24 muestra la estructura de una base de datos HDAM, puede verse que los registros almacenados se numeran desde 1; los registros 1 a N forman el *área direccionable del segmento raíz*, y los registros restantes forman un área de desbordamiento.

En HDAM la carga inicial de la base de datos no necesita ejecutarse en secuencia, pero todos los dependientes de una raíz dada se deben presentar en estricta secuencia jerárquica antes de que sea cargada la raíz siguiente.

Cuando se carga un segmento raíz, el valor de su campo de secuencia se pasa a una rutina de dispersión que transforma el valor y genera una dirección, K, de un registro dentro del área direccionable del segmento raíz (por lo que 1<=K<=N). Si el registro K no contiene espacio suficiente para la nueva raíz, entonces, la nueva raíz se colocará en el registro más cercano del área direccionable que contenga espacio suficiente. Si no existe espacio disponible dentro de ésta área, la nueva raíz se colocará en la siguiente posición disponible del área de desbordamiento.

El procedimiento anterior es el mismo para insertar raíces nuevas en una base de datos existente.

Cuando dos segmentos raíz cuyos valores del campo de secuencia se transforman al mismo valor K, se dice que *colisionan* en K. Todas éstas colisiones en K, se mantienen en secuencia ascendente de la raíz en una cadena que principia en un *punto de anclaje* dentro del registro K.

El acceso a los segmentos raíz en la HDAM será muy rápido, siempre que las cadenas de colisiones no se alarguen demasiado. En cuanto a los segmentos subordinados, se pretende proporcionar acceso rápido a éstos, colocándolos físicamente cerca de la raíz correspondiente.

La inserción en HDAM de un segmento nuevo no hace que ningún segmento existente sea movido, lo cual es una ventaja sobre HISAM. La supresión de segmentos también se realiza de manera diferente en las dos estructuras. En HISAM, las supresiones se indican colocando una señal en el prefijo del segmento. En HDAM, el espacio ocupado por el segmento suprimido queda disponible para ser usado de nuevo, de modo que una inserción posterior puede hacer que el segmento sea físicamente sobreescrito.

H I D A M

HIDAM proporciona acceso indicado a los segmentos raíz y acceso por apuntadores a los segmentos subordinados.

Una base de datos HIDAM en realidad se compone de *dos* bases de datos: una base de datos "de datos", que contiene los datos reales, y una base de datos **INDEX** asociada, que proporciona el índice (denso).

La base de datos "de datos" se compone, en su forma más simple, de un conjunto de datos, ya sea OSAM o VSAM secuenciado por entrada, divididos en registros almacenados de longitud fija. La carga inicial de la base de datos debe realizarse en secuencia jerárquica.

A medida que se van presentando los segmentos se van colocando en el conjunto de datos en las siguientes posiciones disponibles en secuencia, tal como si ésta fuera una base datos HSAM. Las supresiones posteriores se manejan como en HDAM.

La base de datos INDEX es un par de conjuntos de ISAM y OSAM o un solo conjunto de datos VSAM secuenciado por llave. Contiene exactamente un tipo de segmento, el segmento índice (INDEX, es por tanto, solamente de segmentos raíz).

Hay una ocurrencia del segmento índice para cada ocurrencia de la raíz en la base de datos "de datos"; contiene el valor del campo de secuencia de la raíz, junto con un apuntador a esa raíz.

DEFINICION DE LA CORRESPONDENCIA

La definición de la correspondencia comprende entradas adicionales en las proposiciones DBD y SEGM y dos proposiciones extras (CHILD y DATASET), dentro de la descripción de base de datos (DBD).

Las entradas adicionales en la proposición DBD son como sigue:

ACCSESS = HSAM o HISAM o HDAM o HIDAM o INDEX

Esta entrada especifica la estructura pertinente. Nótese que INDEX es una de las posibilidades; para HIDAM se requieren dos DBD's, una para la base de datos "de datos" y una para la base de datos de INDEX.

Se puede especificar una variación de HSAM, conocida como HSAM simple (SHSAM), la cual es una base de datos HSAM que sólo contiene un tipo de segmento y no contiene prefijos almacenados. HISAM tiene una forma análoga simple conocida como SHISAM (disponible solamente con VSAM).

RMNAME

Esta entrada solo se exige si ACCESS=HDAM. Especifica el nombre de la rutina de dispersión suministrada por el DBA. También especifica los valores **N** (número de registros almacenados en el área direccionable del segmento raíz) y **M** (número máximo de bytes dentro del área direccionable del segmento raíz que se pueden asignar a una ocurrencia de PDBR durante una serie de inserciones.

Las entradas adicionales en la proposición SEGM tienen que ver con las opciones de apuntadores disponibles en HDAM e HIDAM; no se aplican a las bases de datos HSAM, HISAM o INDEX. La primera y las más importante de las entradas adicionales es el operando POINTER:

POINTER=HIER o HIERBWD o TWIN o TWINBWD

HIER especifíca que cada ocurrencia de este segmento habrá de contener un apuntador a la siguiente ocurrencia de segmento en secuencia jerárquica, excepto que la última ocurrencia de un segmento dependiente debajo de una raíz dada *no* contiene un apuntador a la raíz siguiente.

HIERBWD es lo mismo que HIER, excepto que los apuntadores son de doble vía.

TWIN especifica que cada ocurrencia de este segmento contendrá un apuntador a la siguiente ocurrencia (si la hay) del mismo tipo de segmento debajo de la misma ocurrencia del padre y un apuntador a la primera ocurrencia de cada tipo de hijo debajo de esta ocurrencia.

TWINBWD es lo mismo que TWIN, excepto que los apuntadores son de doble vía.

PARENT = ((padre, DBLE))

PARENT se requiere sólo si se están usando apuntadores hijo/gemelo (PARENT=TWIN/TWINBWD para el padre) y si se desea un apuntador de cada ocurrencia del padre a la última ocurrencia (tanto como a la primera) de este tipo particular de hijo debajo de esa ocurrencia de padre. La entrada se especifica para el segmento hijo particular en cuestión.

La proposición LCHILD se usa sobre todo con respecto a las bases de datos lógicas. No obstante, también se usa en HIDAM para ligar el segmento de índice en la base de datos INDEX con el segmento que es indicado en la correspondiente base de datos "de datos"; el segmento de datos se considera *hijo lógico* del segmento INDEX. La DBD INDEX contendrá una proposición SEGM y una proposición FIELD.

Además de éstas dos proposiciones, la DBD INDEX debe incluír una proposición LCHILD para especificar el segmento de datos y el campo dentro de el cual se va a efectuar el indicamiento. Por ejemplo:

LCHILD NAME = (CURSO,PDBDEDUC), INDEX = CURSO#

Una proposición LCHILD también debe incluírse en la DBD HIDAM, predeciendo o siguiendo a las proposiciones FIELD para la raíz de HIDAM.

Las proposiciones DATASET se usan para especificar qué segmentos se van a asignar a cada grupo de conjuntos de datos y especifican los nombres en las proposiciones de Lenguaje de Control de Trabajos (JCL) de OS/VS que se requerirán cuando se programe una aplicación para operar sobre la base de datos.

CAPITULO 5.- MODELO DE RED: LA ALTERNATIVA.

Una red, es una estructura más general que una jerarquía porque una ocurrencia de registro puede tener *cualquier número* de superiores inmediatos, es decir, no está limitado a un máximo de uno, como ocurre en una jerarquía.

De ésta manera, el enfoque de red permite modelar una correspondencia de muchos a muchos de manera más directa que el enfoque jerárquico.

Una basa de datos de red consiste en una serie de registros que están conectados entre sí por medio de *ligas (links)*. Un registro es, en muchos aspectos, similar a una entidad en el modelo de entidad-relación.

Gran parte del interés actual por el enfoque de redes se deriva de la publicación del Informe del DBTG en abril de 1971; por tanto, como en el capítulo anterior, aquí muchos de los detalles son un poco específicos del sistema, pero los conceptos fundamentales se pueden considerar característicos de cualquier sistema basado en redes. En todo caso, DBTG es el ejemplo más importante de este enfoque.

5.1.- ARQUITECTURA DEL MODELO DE RED.

La arquitectura de un sistema DBTG se muestra en la figura 5.1. La vista conceptual se define por medio del esquema. El esquema se compone de las definiciones de los diversos tipos de registro de la base de datos, de los elementos de datos que contienen y de los conjuntos DBTG en los cuales se agrupan.

La estructura de almacenamiento (la vista interna) de la base de datos se describe por medio del esquema de almacenamiento, escrito en un Lenguaje de Descripción de Almacenamiento de Datos (DSDL).

La vista externa se define por medio de un subesquema. Un subesquema se compone de una especificación de los tipos de registro del esquema en los que se interesa el usuario, los elementos del esquema que desea ver en esos registros y las asociaciones (conjuntos DBTG) del esquema que ligan a esos registros y que el usuario desea considerar.

Cada programa de aplicación *invoca* al subesquema correspondiente. Esta invocación proporciona la definición del área de trabajo del usuario (ATU) para ese programa. La ATU contiene una localidad distinta para cada tipo de registro que se defina en el subesquema.

5.2.- ESTRUCTURA LOGICA DE DATOS.

5.2.1. CONSTRUCCION CONJUNTO DBTG.-(EJEMPLOS JERARQUICOS)

El conjunto DBTG es la principal característica distintiva de la estructura de datos de DBTG. Cada *tipo* de conjunto DBTG tiene por definición un cierto tipo de registro como *propietario* y otro tipo de registro como *miembro*.

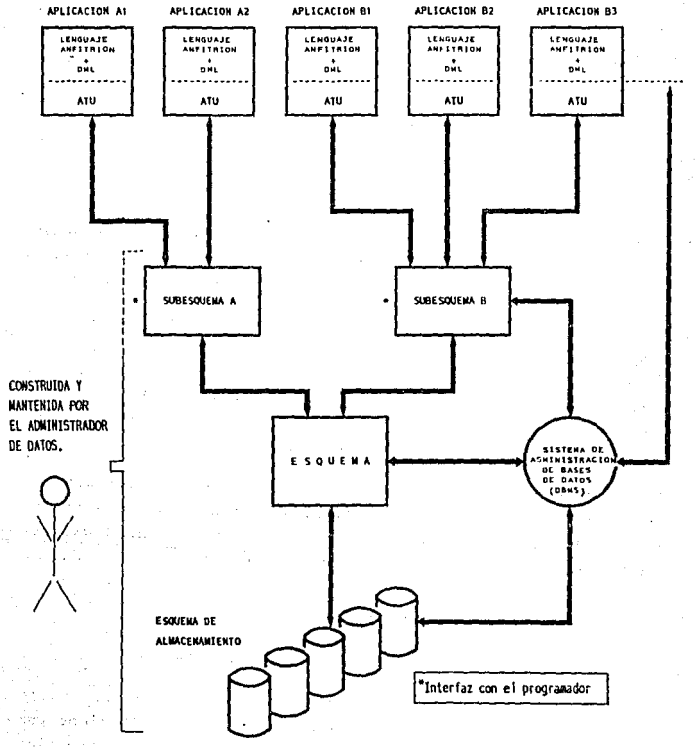


FIG.5.1. ARQUITECTURA DE UN SISTEMA DBTG.

La figura 5.2 muestra una base de datos que contiene información sobre departamentos y empleados. Hay tres ocurrencias de DEPTO y nueve de EMP. Estas ocurrencias de registro se agrupan en tres ocurrencias de un conjunto DBTG llamado EMPDEPTO. Este conjunto DBTG tiene a DEPTO como propietario y a EMP como miembro. Se trata entonces de una jerarquía con un nivel dependiente.

Cada *ocurrencia* de un tipo de conjunto DBTG dado se compone de una ocurrencia de su propietario junto con cero o más ocurrencias de su miembro.

Dentro de un conjunto DBTG dado, ninguna ocurrencia del registro miembro puede pertenecer a más de una ocurrencia de ese conjunto DBTG en cualquier instante. Por ejemplo, ninguna ocurrencia de EMP puede pertenecer a más de una ocurrencia de EMPDEPTO en cualquier instante dado.

Cada ocurrencia de un conjunto DBTG representa una asociación jerárquica entre la ocurrencia del propietario y las ocurrencias correspondientes del miembro.

La convención usada hasta ahora para representar ocurrencias de conjuntos DBTG no es la única, pero sí es una convención aceptada. Otra convención aceptada, es la técnica del diagrama de estructura de datos de Bachman, véase la figura 5.3, donde la estructura de EMPDEPTO se ha mostrado como si fuera una estructura de IMS. Las diferencias entre el simbolismo de Bachman y el usado en IMS son que:

a) La liga entre el propietario y el miembro se etiqueta con el nombre de conjunto DBTG, mientras que tales ligas son anónimas en IMS.
b) La liga *está dirigida* de modo que indique cuál es el propietario y cuál el miembro.

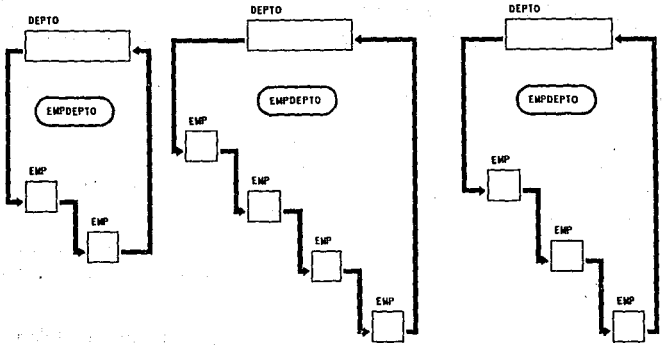


FIG.5.2. BASES DE DATOS DE EMPLEADOS Y DEPARTAMENTOS.

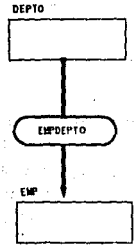


FIG.5.3.ESTRUCTURA DEL CONJUNTO DBTG EMPDEPTO.

Ahora, la figura 5.4 muestra (parte de) una base de datos que contiene información sobre las divisiones, departamentos y empleados. Este ejemplo ilustra el hecho de que un tipo particular de registro (DEPTO) se puede declarar como miembro de un tipo de conjunto DBTG (DEPTODIV) y propietario de otro (EMPDEPTO). Aquí se tiene entonces una **estructura jerárquica con dos niveles dependientes.** En general se puede armar una jerarquía de esta manera con cualquier número de dependientes.
La figura 5.5 muestra (parte de) una base de datos que contiene información sobre empleados, su historia de trabajo y su historia de educación.

Aquí cada ocurrencia de EMP es propietaria de dos ocurrencias de conjunto DBTG: una ocurrencia de TRABEMP que representa a los trabajos que el empleado ha tenido, y una ocurrencia de CURSOEMP, que representa los cursos a los que el empleado ha asistido. La figura 5.5. muestra que se pueden construír estructuras jerárquicas que no sólo contengan cualquier número de niveles, sino que también tengan cualquier número de tipos de registro en cada nivel dependiente, es decir, dicha figura muestra una **jerarquía con más de un tipo de registro en un nivel dependiente.**

La figura 5.6 muestra (parte de) una base de datos que contiene información sobre la estructura administrativa de una compañía.
No se permite que el mismo tipo de registro sea a la vez propietario y miembro de un tipo de conjunto DBTG. Aquí la asociación que se va a representar es la normal de administrador a empleados (donde un administrador también se considera empleado, y a su vez, tiene un administrador, y así sucesivamente). Para ésto es necesario introducir un nivel de mediación (ENLACE), como lo ilustra la figura 5.6; además, se definen dos tipos de conjuntos DBTG:

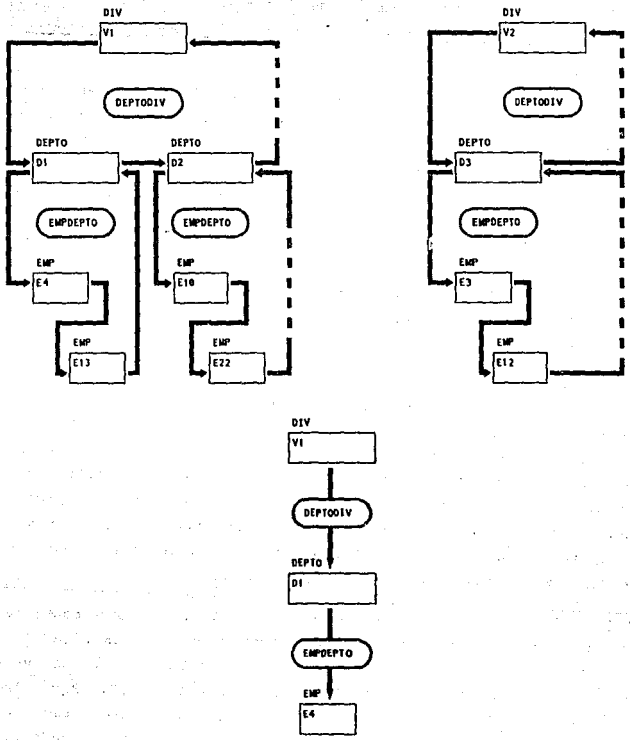


FIG.5.4. ESTRUCTURA DE LOS CONJUNTOS DBTG DEPTODIV Y EMPDEPTO.

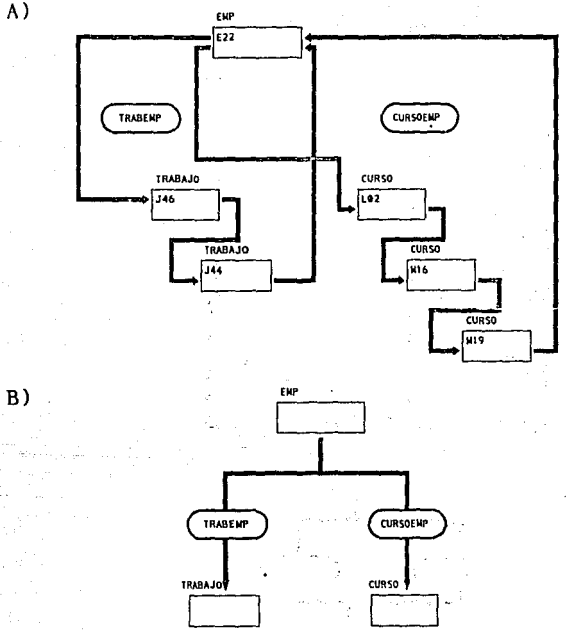


FIG.5.5. A) BASE DE DATOS DE HISTORIA DE EMPLEADOS.
B) ESTRUCTURA DE LOS CONJUNTOS DBTG TRABEMP Y CURSOEMP.

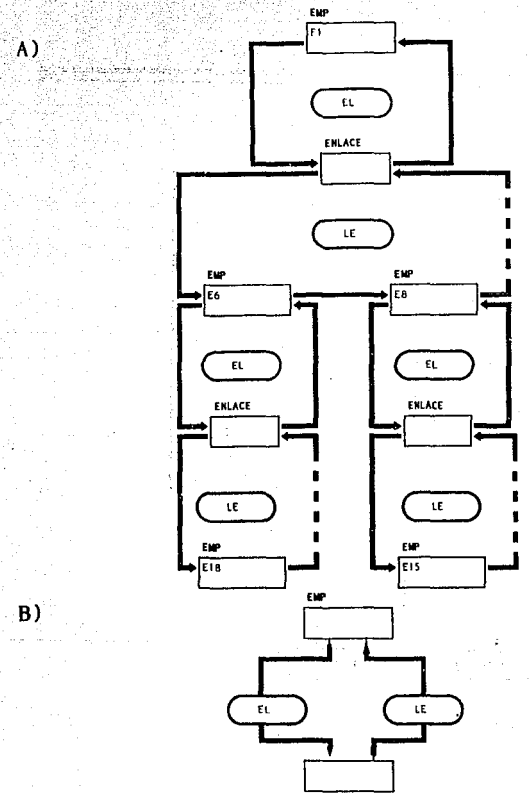


FIG.5.6. A)BASE DE DATOS DE LA ESTRUCTURA ADMINISTRATIVA.
B)ESTRUCTURA DE LOS CONJUNTOS DBTG EL Y LE.

-EL (propietario: EMP, miembro: ENLACE) y
-LE (propietario: ENLACE, miembro: EMP).

De esta manera, el empleado E1 es el administrador de los empleados E6,E8..., E6 es el administrador de E18..., y así sucesivamente. Se está hablando por lo tanto de una jerarquía con el mismo tipo de registro en más de un nivel.

5.2.2. *CONSTRUCCION CONJUNTO DBTG.-(EJEMPLOS DE REDES)*

Para ejemplificar una situación característica de red, supóngase el plan actual para una temporada de conciertos orquestales de la figura 5.7.

En general, cada concierto incluirá obras de varios compositores y cada compositor tendrá obras en varios conciertos. Se tiene una red que contiene dos tipos de entidades básicas (CONCIERTO y COMPOSITOR), por lo que se ha introducido un tipo de registro de *conexión* (OBRA), cuya función es conectar a los tipos de entidad. También se introducen dos tipos de conjuntos DBTG:

CONCIO (propietario: CONCIERTO, miembro: OBRA) y
COMPOSO (propietario: COMPOSITOR, miembro: OBRA).

En general, la ocurrencia del conjunto DBTG CONCIO para un concierto dado contiene ocurrencias de OBRA para todas las obras en ese concierto y la ocurrencia del conjunto COMPOSO para un compositor dado contiene ocurrencias de OBRA para todas las ejecuciones de las obras de ese compositor en cualquier concierto.

La figura 5.7 muestra seis ocurrencias del conjunto DBTG CONCIO y siete del conjunto DBTG COMPOSO; por razones de espacio no se muestran los nombres de los conjuntos DBTG. La figura 5.8 es el diagrama correspondiente de estructura de datos.

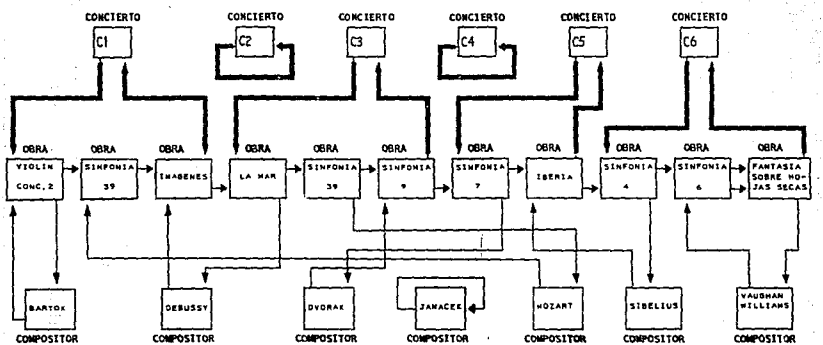


FIG.5.7. BASE DE DATOS DE CONCIERTOS ORQUESTALES.

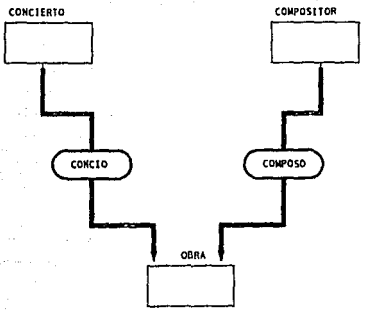


FIG.5.8. ESTRUCTURA DE LOS CONJUNTOS DBTG CONCIO Y COMPOSO

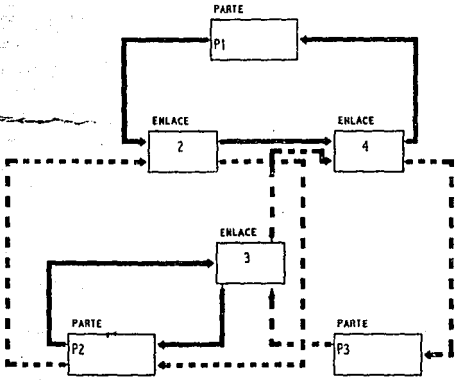


FIG.5.9. BASE DE DATOS DE PARTES

El ejemplo anterior muestra el método general de construcción de una base de datos de red en DBTG. En particular, si se van a conectar *n* tipos de entidad, se introducen : un tipo de registro de conexión y *n* tipos de conjunto DBTG; se hace entonces que cada uno de los *n* tipos de registro de entidad sea propietario de uno de los tipos de conjunto DBTG, y que el tipo de registro de conexión sea miembro de todos ellos; cada ocurrencia del registro de conexión se hace miembro exactamente de una ocurrencia de cada uno de los *n* tipos de conjunto DBTG, de ésta manera, representa la conexión entre las *n* entidades correspondientes. Se tendría así, una red con más de dos tipos de entidad.

Un caso especial de la situación de dos entidades, sería una base de datos que contenga información acerca de partes y componentes (donde un componente es en sí mismo parte y puede tener más componentes, y así sucesivamente), aquí se tendría una red que sólo comprende un tipo de entidad, las partes. La figura 5.9 ilustra éste caso. Se introduce un registro de conexión (ENLACE). Después se definen dos tipos de conjunto DBTG : LM (lista de materiales) y DU (dónde se usan), ambos con PARTE como propietario, y ENLACE como miembro.

En la figura 5.9 las ocurrencias del conjunto DBTG LM se indican con flechas continuas, y las ocurrencias del conjunto DBTG DU con flechas punteadas. De esta manera, en la figura se puede ver que P1 contiene a P2 y a P3 como componentes inmediatos, y P2, a su vez, contiene a P3 como componente inmediato; recíprocamente, P3 es componente inmediato de P1 y P2, y P2 es componente inmediato de P1. Los números dentro de las ocurrencias de ENLACE representan las cantidades correspondientes.

5.2.3. CONJUNTOS DBTG SINGULARES.

Un conjunto DBTG *singular* puede considerarse como un conjunto DBTG que tiene exactamente una ocurrencia y que no tiene registro propietario.

Considérese la figura 5.2 nuevamente, en donde, cada grupo representa el conjunto de todos los empleados de algún departamento dado. Hay un operador *halle siguiente dentro de grupo* en el lenguaje de manipulación de datos que permitirá usar un grupo como ruta de acceso a los registros correspondientes.

No obstante, no se ve en la figura 5.2 ninguna ruta de acceso que conecte *todas* las ocurrencias de registro de EMP; tampoco una que conecte todas las ocurrencias de registro de DEPTO. Este problema puede resolverse añadiendo una ruta de acceso que ligue a todas las ocurrencias de DEPTO, sin embargo, el uso de la construcción conjunto DBTG, como se ha señalado hasta ahora, resulta muy poco manejable para este propósito. Los conjuntos DBTG singulares ofrecen una solución más conveniente al problema.

En el ejemplo, se podrán reunir todas las ocurrencias de DEPTO en un conjunto DBTG singular CONJUNTODEPTO. En forma semejante se podrían reunir todas las ocurrencias de EMP en otro conjunto DBTG singular llamado CONJUNTOEMP. Estos conjuntos singulares, son muy semejantes a los archivos secuenciales ordinarios; por ejemplo, CONJUNTOEMP proporciona acceso secuencial al conjunto de todas las ocurrencias de EMP, de acuerdo con el ordenamiento que se le indique para CONJUNTOEMP. en el esquema.

5.2.4. *UN EJEMPLO DE ESQUEMA.*

La figura 5.10 indica los datos de muestra de una base de datos de proveedores y partes. Las líneas representan conexiones entre proveedores y partes, y los números que están junto a ellas son las cantidades adecuadas.

La figura 5.11 muestra el diagrama de estructura de datos para ésta red. Se ha introducido un registro de conexión, SP, con elementos de datos NROS (número de proveedor), NROP (número de parte) y CTD (cantidad) y dos conjuntos DBTG:

S-SP (propietario S, miembro SP) y
P-SP (propietario P, miembro SP).

Parte de la base de datos real correspondiente a los datos de muestra se indica en la figura 5.12.

Nótese que cada ocurrencia de registro de SP incluye los valores de NROS y NROP adecuados; por lo tanto, la estructura incluye un grado de redundancia, la cual es necesaria para poder ordenar las ocurrencias de SP por el valor de NROP dentro de cada ocurrencia de S-SP y por el valor de NROS dentro de cada ocurrencia de P-SP. El esquema se muestra en la figura 5.13 y la explicación es la siguiente:

LINEA:	EXPLICACION.
1	Asigna un nombre al esquema.
3	Define la existencia de un tipo de registro S.
4 a 5	Especifican que dos ocurrencias del tipo de registro S no puden contener el mismo valor para el elemento de datos NROS en cualquier instante dado.
6 a 9	Definen los tipos de elementos de datos que constituyen a S.

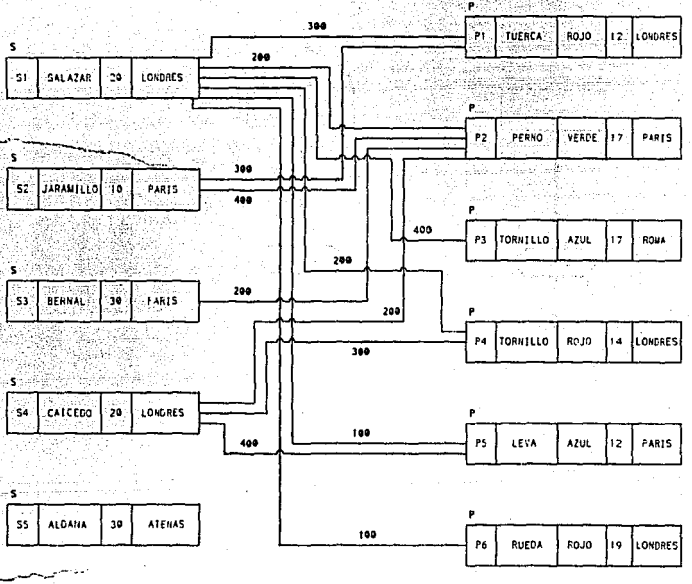


FIG.5.10. DATOS DE MUESTRA (PROVEEDORES Y PARTES).

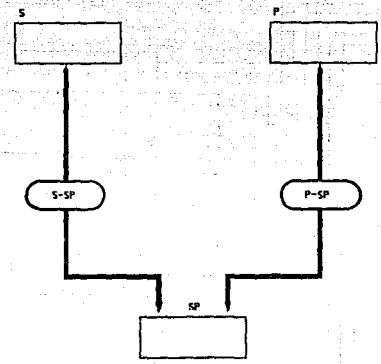


FIG.5.11. ESTRUCTURA DE LOS CONJUNTOS DBTG S-SP Y P-SP.

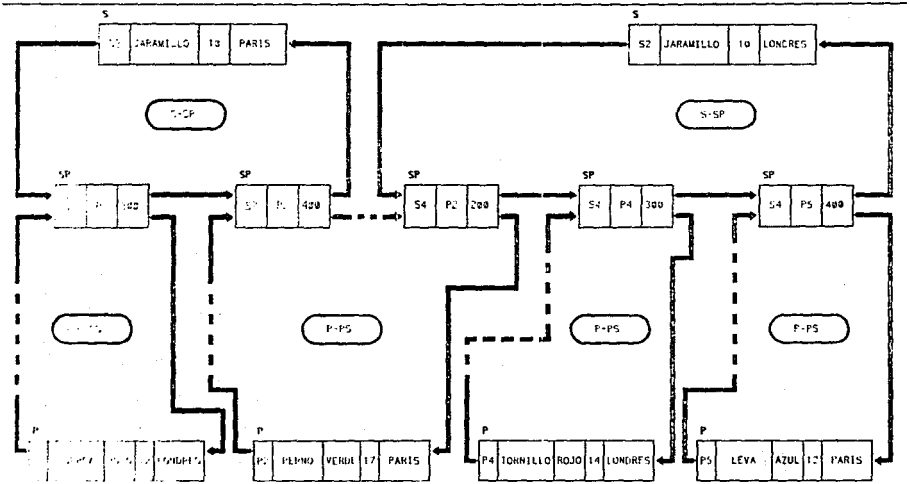


FIG. 5.12. (PARTE DE LA) BASE DE DATOS DE PROVEEDORES Y PARTES.

```
1 SCHEMA NAME IS PROVEEDORES-Y-PARTES.
2
3 RECORD NAME IS:
4    DUPLICATES ARE NOT ALLOWED
5              FOR NORS IN S.
6       NORS   : TYPE IS CHARACTER 5.
7       NOMS   : TYPE IS CHARACTER 20.
8       ESTATUS: TYPE IS FIXED DECIMAL 3.
9       CIUDAD : TYPE IS CHARACTER 15.
10
11 RECORD NAME IS P:
12    DUPLICATES ARE NOT ALLOWED
13              FOR NROP IN P.
14       NROP   : TYPE IS CHARACTER 6.
15       NOMPAR : TYPE IS CHARACTER 20.
16       COLOR  : TYPE IS CHARACTER 6.
17       PESO   : TYPE IS FIXED DECIMAL 4; DEFAULT IS -1.
18       CIUDAD : TYPE IS CHARACTER 15.
19
20 RECORD NAME IS P:
21    DUPLICATES ARE NOT ALLOWED
22              FOR NROS IN SP, NROP IN SP.
23       NROS   : TYPE IS CHARACTER 5.
24       NROP   : TYPE IS CHARACTER 6.
25       CTD    : TYPE IS FIXED DECIMAL 5.
26
27 SET NAME IS S-SP;
28    OWNER IS S;
29    ORDER IS SORTED BY DEFINED KEYS
30       DUPLICATES ARE NOT ALLOWED.
31    MEMBER IS SP;
32       INSERTION IS AUTOMATIC
33       RETENTION IS FIXED;
34       KEY IS ASCENDING NROP IN SP;
35       SET SELECTION IS BY VALUE OF NROS IN S.
36
37 SET NAME IS P-SP;
38    OWNER IS P;
39    ORDER IS SORTED BY DEFINED KEYS
40       DUPLICATES ARE NOT ALLOWED.
41    MEMBER IS SP;
42       INSERTION IS AUTOMATIC
43       RETENTION IS FIXED;
44       KEY IS ASCENDING NROS IN SP;
45       SET SELECTION IS BY VALUE OF NROP IN P.
```

FIG.5.13. ESQUEMA PROVEEDORES-Y-PARTES.

11 a 18 Definen el tipo de registro P. En la línea 17 aparece la frase DEFAULT IS-1 para el elemento de datos PESO, ésto significa que si PESO se omite de un subesquema correspondiente al registro P del esquema, y si un programa crea una ocurrencia del registro P usando ese subesquema, entoces el valor de PESO en esa ocurrencia se ajustará a -1.

20 a 25 Definen el tipo de registro SP. Aquí DUPLICATES NOT ALLOWED se ha especificado para la combinación de dos elementos de datos distintos.

27 Define la existencia de un tipo de conjunto DBTG S-SP.

28 Especifica el tipo de registro propietario para S-SP, que es S.

29 Define la secuencia de las ocurrencias de SP dentro de cada ocurrencia del conjunto DBTG S-SP que se van a clasificar por llaves definidas (SORTED BY DEFINED KEYS).

30 Especifica que dos ocurrencias de SP dentro de una ocurrencia dada de S-SP no pueden contener el mismo valor de NROP.

31 Especifica el tipo de registro miembro para S-SP, que es, SP.

32 y 33 Especifican la clase de pertenencia de SP dentro de S-SP. (la clase de pertenencia se verá más adelante).

34 La *llave de control de clasificación* se define por la cláusula KEY; en general, puede ser ASCENDING, o DESCENDING, y la *llave* implicada puede ser la combinación de cualquier número de elementos de datos en el registro.

35 Esta línea tiene que ver con la selección de conjunto DBTG (SET SELECTION).

37-45 Definen el tipo de conjunto P-SP.

5.2.5. *CLASE DE PERTENENCIA.*

En el esquema de la figura 5.13, cada subentrada MEMBER debe incluír una especificación de la *clase de pertenencia* para el tipo de registro comprendido en el tipo de conjunto DBTG de que se trate.

La clase de pertenencia se especifica por medio de la entrada INSERTION/RETENTION, por lo tanto, puede considerarse como una combinación de una clase de inserción y una clase de retención.

La clase de inserción es AUTOMATIC o MANUAL. La clase de retención es FIXED (fija), MANDATORY (obligatoria) u OPTIONAL (opcional).

Un tipo de registro dado puede tener cualquier combinación de valores de clase de inserción y clase de retención con respecto a un tipo de conjunto DBTG dado, también puede tener diferentes clases de pertenencia en diferentes tipos de conjunto DBTG.

La clase de pertenencia de un registro en un conjunto DBTG afecta a los programas que crean, modifican o suprimen instancias de la asociación jerárquica que ese conjunto DBTG representa.

Para explicar el efecto de las diferentes clases de inserción y retención, considérese la figura 5.14.

CLASE DE RETENCION

Si la pertenencia de M en PM es:
FIXED:
Una vez insertada una ocurrencia de M (m, por ejemplo) en una ocurrencia de PM, la única forma de destruír la asociación entre m y PM es eliminar a m de la base de datos por medio de una operación de ERASE. Si se borra una ocurrencia de P, entonces todas las ocurrencias correspondientes de M también deben suprimirse.

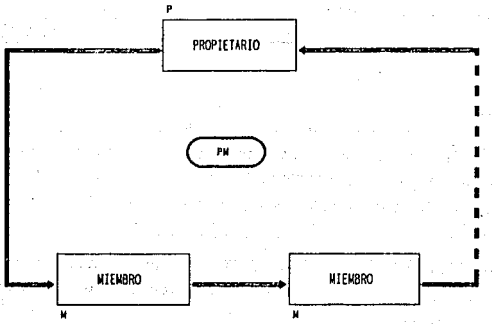


FIG.5.14. OCURRENCIA ORDINARIA DEL CONJUNTO DBTG.

	AUTOMATIC		MANUAL	
FIXED	CONNECT	X	CONNECT	
	DISCONNECT	X	DISCONNECT	X
	RECONNECT	X	RECONNECT	X
	CONNECT	X	CONNECT	
	DISCONNECT	X	DIDCONNECT	X
	RECONNECT		RECONNECT	
	CONNECT		CONNECT	
	DISCONNECT		DISCONNECT	
	RECONNECT		RECONNECT	

FIG.5.15. EFECTO DE LA CLASE DE PERTENENCIA SOBRE: CONNECT, DISCONNECT Y RECONNECT.

MANDATORY:
Una vez insertada una ocurrencia de M (m, por ejemplo) en una ocurrencia de PM, nunca podrá sacarse de la ocurrencia de PM por medio de una operación DISCONNET, pero podrá transferirse de una ocurrencia de PM a otra por medio de una operación RECONNET.

OPTIONAL:
Una ocurrencia de M *puede* retirarse de una ocurrencia de PM sin borrarse de la base de datos (por ejemplo, por medio de una operación DISCONNET).

CLASE DE INSERCION

Si la pertenencia de M en PM es:

AUTOMATIC:
Cuando se cree inicialmente una ocurrencia de M (m, por ejemplo) y se coloque en la base de datos (por medio de una operación STORE), el DBMS automáticamente la conectará en la ocurrencia adecuada de PM.

MANUAL:
El almacenamiento de una ocurrencia m no ocasiona ésta conexión automática; para conectar a m en una ocurrencia de PM, el programa debe emitir una operación CONNECT explícita.

La figura 5.15 representa un resumen parcial de lo dicho anteriormente.

5.2.6. *SELECCION DE CONJUNTO DBTG.*

Para que el DBMS pueda realizar la selección automática de una ocurrencia de conjunto DBTG, el DBA debe definir una cláusula SET SELECTION dentro de la subentrada MEMBER de la entrada del conjunto DBTG en el esquema.

Para explicar este punto, considérese el esquema de la figura 5.13 nuevamente y supóngase que se requiere almacenar la ocurrencia de SP nueva S5/P6/700 (para simplificar se ignora que SP es un miembro AUTOMATICo de S-SP y P-SP).

La manera más sencilla de utilizar la cláusula SET SELECTION para el conjunto S-SP (miembro SP) es:

```
SET SELECTION IS BY APPLICATION
```

Esto significa que al programa de aplicación le corresponde seleccionar procedimentalmente la ocurrencia correcta de S-SP antes de almacenar la ocurrencia de SP nueva. El DBMS supondrá que la ocurrencia actual de S-SP es la correcta, por tanto, para almacenar la ocurrencia de SP S5/P6/700, una posible secuencia de codificación es la siguiente.

```
construya ocurrencia de SP 'S5/P6/700' en ATU
MOVE 'S5' TO NROS IN S
FIND ANY S USING NROS IN S
STORE SP
```

Esta es solo una de las muchas posibles maneras de establecer la ocurrencia requerida de S-SP como tal ocurrencia actual.

El segundo caso es el ilustrado en la figura 5.13 :

```
SET SELECTION IS BY VALUE OF NROS IN S
```

Esto significa que cuando el DBMS va a seleccionar una ocurrencia de S-SP, lo hace localizando la ocurrencia correspondiente de su propietario (S) y usando el elemento de datos NROS IN S (el cual se debe haber especificado con DUPLICATES NOT ALLOWED en la definición de ese propietario). Por lo tanto, el programador debe dar el valor correcto al correspondiente elemento de datos de la ATU antes de almacenar la nueva ocurrencia de SP. Por ejemplo, como sigue:

```
construya ocurrencia de SP 'SS/P6/700' en ATU
MOVE 'S6' TO NROS IN S
STORE SP
```

La tercera forma de SET SELECTION es:

```
SET SELECTION IS BY STRUCTURAL NROS IN SP = NROS IN S
```

En este caso, esto significa que el DBMS seleccionará una ocurrencia del conjunto DBTG S-SP seleccionando la ocurrencia de registro de S que tenga un valor de NROS igual al valor en el registro de SP que se está almacenando, es decir, igual al valor de NROS IN SP dentro de la ATU. (El elemento de datos NROS IN S debe tener DUPLICATES NOT ALLOWED para que esta forma de SET SELECTION sea legal.) La codificación para crear la ocurrencia de SP SS/P6/700 ahora se reduce a:

```
construya ocurrencia de SP SS/P6/700 en ATU
STORE SP
```

Una cláusula SET SELECTION no es requerida ni permitida si el conjunto DBTG implicado es singular. En realidad, la única ocurrencia del conjunto DBTG siempre se considera la ocurrencia actual.

5.3.- LENGUAJE DE DEFINICION Y MANIPULACION DE DATOS

Una vista externa en DBTG se define por medio de un *subesquema*. Un subesquema es un subconjunto sencillo del esquema correspondiente. Cualquier número de subesquemas puede definirse sobre un esquema dado; cualquier número de programas puede compartir un subesquema dado.

5.3.1. DIFERENCIAS ENTRE ESQUEMA Y SUBESQUEMA

-La primer diferencia es que, como ya se mencionó anteriormente, un subesquema es un subconjunto del esquema.

-Cualquiera de las siguientes entradas del esquema puede omitirse en un subesquema dado.

* La declaración de uno o más conjuntos DBTG
* La declaración de uno o más registros.
* La declaración de uno o más elementos de datos.

-El subesquema automáticamente proporciona un nivel de seguridad de los datos, en cuanto que un programa tal vez no pueda accesar ningún dato que no se defina en el subesquema correspondiente. (excepto en el caso de ERASE).

-En el subesquema, pueden definirse nombres privados (*alias*) para conjuntos DBTG, registros y elementos de datos.

-En el subesquema, a los elementos de datos se les puede dar tipos de datos diferentes.

-En el subesquema, el orden relativo de los elementos de datos dentro del registro que los contiene se puede cambiar.

-En el subesquema, a los conjuntos DBTG se les pueden dar diferentes cláusulas SET SELECTION; ésto significa que la cláusula SET SELECTION en el esquema puede ser anulada por una del subesquema.

5.3.2. *EJEMPLO DE UN SUBESQUEMA*

Un ejemplo sencillo de un subesquema en COBOL se muestra en la figura 5.16, la cual está basada en el esquema de la figura 5.13. Este subesquema incluye información sobre proveedores y remesas del esquema subyacente pero no incluye la información de las partes. Un subesquema en COBOL se compone de:

-una TITLE DIVISION (división de título), que le da el nombre al subesquema e identifica al esquema subyacente.

-una MAPPING DIVISION (división de correspondencia), donde se definen los alias.

-una STRUCTURE DIVISION (división de estructura), la cual se compone a su vez de tres secciones:

REALM SECTION (sección de regiones)
RECORD SECTION (sección de registros)
SET SECTION (sección de conjuntos DBTG)

REALM SECTION especifica las *regiones de interés*. Una región es un subconjunto lógico de la vista de la base de datos total como la define el subesquema. Una región se compone de todas las ocurrencias de registro de uno o más tipos especificados. Un subesquema puede poner en lista cualquier número de regiones, y regiones distintas pueden tener tipos de registro en común (es decir, se pueden traslapar).

RECORD SECTION pone en lista todos los tipos de registro del esquema que sean de interés.

SET SECTION pone en lista todos los tipos de conjunto DBTG del esquema que sean de interés.

En adelante, se supondrá que el subesquema es idéntico al esquema correspondiente en todos los aspectos muy importantes.

```
TITLE DIVISION.

SS PROVEEDORES WITHIN PROVEEDORES-Y-PARTES.

MAPPING DIVISION.

ALIAS SECTION.
AD SET SUMINISTROS IS S-P.

STRUCTURE DIVISION.

REALM SECTION.
RD SUMINISTROS-REALM CONTAINS S, SP RECORDS.

RECORD SECTION.
01   S.
       02 NORS; PICTURE IS X(5).
       02 CIUDAD; PICTURE IS X(24).
01   SP.
       02 NROP; PICTURE IS X(6).
       02 NROS; PICTURE IS X(5).
       02 CTD ; PICTURE IS S9(5).

SET SECTION.
SD SUMINISTROS.
```

FIG.5.16. UN SUBESQUEMA DE MUESTRA (en COBOL).

5.3.3. MANIPULACION DE DATOS.

Al igual que en el DDL de subesquemas, la explicación de la manipulación de datos se basará en el DML definido para COBOL.

POSICION ACTUAL

Este concepto es una generalización de la noción familiar de posición actual dentro de un archivo. Para cada programa que opere bajo su control, el DBMS mantiene una tabla de *indicadores de posición actual*.

Un inidicador de posición actual es un objeto cuyo valor generalmente es una *llave de base de datos*. Las llaves de base de datos son valores generados por el sistema y que identifican de manera única a los registros de la base de datos.

Para una región R, la ocurrencia de registro accesada más recientemente dentro de R se llama *registro actual de la región R*.

Para un tipo de registro T, la ocurrencia de T accesada más recientemente se llama *registro actual de tipo T* u *ocurrencia de T actual*.

Para un tipo de conjunto DBTG S, la ocurrencia de registro accesada más recientemente se llama *registro actual del conjunto DBTG S*, el cual se refiere a una ocurrencia de *registro*, pero también identifica de manera única a una ocurrencia de *conjunto DBTG*, que es, la única ocurrencia de S que lo contiene. Esta ocurrencia de conjunto se llama *ocurrencia de S actual*.

La ocurrencia de registro accesada más recientemente se llama *registro actual de la unidad de ejecución*, que se suele abreviar como *actual de la unidad de ejecución*. Esta es la posición actual más importante.

A manera de ejemplo, considérese la siguiente secuencia de proposiciones:

```
1:   MOVE 'S4' TO NROS IN S
2:   FIND ANY S USING NROS IN S
3:   FIND FIRST SP WITHIN S-SP
4:   FIND OWNER WITHIN P-SP
```

Se han colocado números al principio de cada línea, como guías para la explicación siguiente:

1: Move da valor inicial al elemento de datos NROS IN S del Area de Trabajo del Usuario (ATU).

2: Localiza la correspondiente ocurrencia de registro de S, a saber, la del proveedor S4.

3: Localiza la primera ocurrencia de SP dentro de la ocurrencia del conjunto SP poseída por el proveedor S4, la cual sería, la ocurrencia de SP 'S4/P2/200'.

4: Localiza al propietario de ésta ocurrencia de SP dentro del conjunto P-SP, que es, la ocurrencia de P para la parte P2.

Por lo tanto, al final de la secuencia se tiene:

Actual de la unidad de ejecución:	**P 'P2'**
Ocurrencia de S actual:	**S 'S4'**

Ocurrencia de P actual:	P 'P2'
Ocurrencia de SP actual:	SP 'S4/P2/200'
Registro actual del conjunto DBTG S-SP:	SP 'S4/P2/200'(miembro)
Registro actual del conjunto DBTG P-SP:	P 'P2' (propietario)
Ocurrencia de S-SP actual	poseída por S 'S4'
Ocurrencia de P-SP actual	poseída por P 'P2'
Registro actual de la región S-SP-P:	P 'P2'

CONDICION DE EXCEPCION O ERROR.

Al ejecutar cualquier proposición del DML, el DBMS coloca un valor en el *registro especial de la base de datos* DB-ESTADO (estado de BD) para indicar el resultado de la ejecución; un valor igual a cero significa que la proposición se ejecutó con éxito; un valor diferente de cero significa que hubo algún error o condición de excepción.

El programador debe suministrar uno o más *procedimientos declarativos* por medio de la proposición USE FOR DB-EXCEPTION (use para excepción de BD) para manejar todos los casos de error o excepción.

A continuación se verá un ejemplo usando una forma simplificada de la sintaxis real de COBOL. (Estos procedimientos se especifican al principio de la Procedure Division.)

```
DECLARATIVES.

USE FOR DB-EXCEPTION ON '5502100'.
NO-HAS-PROC.
MOVE 'SI' TO NO-HAS.
USE FOR-EXCEPTION ON '0502400'.
NO-ENCONTRADO-PROC.
MOVE 'SI' TO NO-ENCONTRADO.
```

```
        USE FOR DB-EXCEPTION ON OTHER.
        OTRO-PROC.
        END DECLARATIVES.
```

El primer USE especifica un procedimiento compuesto de la sola proposición MOVE 'SI' NO-MAS, el cual se ejecutará siempre que DB-ESTADO tome el valor 0502100 (lo cual ocurrirá cuando la posición actual en una operación FINDNEXT (halle siguiente) esté en fin-de-conjunto o fin-de-región).

El segundo USE especifica un procedimiento semejante para el valor 0502400 de DB-ESTADO (*registro no encontrado*).

El tercer USE especifica un procedimiento (vacío) "*haga nada*" que se ejecutará siempre que DB-ESTADO tome cualquier otro valor diferente a cero.

En general, una proposición USE dada puede especificar OTHER o una lista de valores específicos. Además de DB-ESTADO, el DBMS también da un valor a algunos otros registros especiales (DB-SET-NAME, DB-RECORD-NAME y DB-DATA-NAME) después de ejecutar algunas proposiciones del DML.

5.7.4. PRINCIPALES PROPOSICIONES DEL DML

GET (obtenga)

Recupera el actual de la unidad de ejecución.

Ejemplos:

Obtenga todos los detalles del proveedor S4.

```
MOVE 'S4' TO NROS IN S
FIND ANY S USING NROS IN S
GET S
```

El resultado de ésta codificación es traer la ocurrencia de registro de S para el proveedor S4 a la localidad de S dentro de la ATU. La proposición FIND en sí no recupera ningún dato.

Puesto que GET opera sobre el actual de la unidad de ejecución, sería válido escribir solamente:

```
GET
```

Esto traería al actual de la unidad de ejecución ,cualquiera que sea su tipo, al sitio adecuado de la ATU. De cualquier manera, tal especificación explícita es más clara y se permite al DBMS controlar que el actual de la unidad de ejecución sea una ocurrencia del tipo adecuado. Por ejemplo, si por alguna razón el actual de ejecución no fuera una ocurrencia de S, GET fallaría y se colocaría un valor diferente de cero en DB-ESTADO.

Obtenga el nombre del proveedor y la ciudad para el proveedor S4.

```
FIND ocurrencia de S para S4
GET NOMS IN S, CIUDAD IN S
```

En este caso, sólo se traen a la ATU el nombre y la ciudad para el proveedor S4; los otros elementos de la ATU no cambian.

STORE (almacene)

Crea una nueva ocurrencia de registro y la establece como el actual de la unidad de ejecución. También actualiza otros indicadores de posición actual en la medida que sea necesario.

Ejemplo:

Cree la ocurrencia de SP 'S5/P6/700'.

```
1:
        MOVE 'S5' TO NROS IN SP
        MOVE 'P6' TO NROP IN SP
        MOVE 700 TO CTD IN SP
2:
        MOVE 'S5' TO NROS IN S
        MOVE 'P6' TO NROP IN P
3:
        STORE SP
```

Los tres primeros MOVEs del paso 1 permiten construír la nueva ocurrencia de SP en la ATU.

En el esquema de la figura 5.13, SP se declara como miembro AUTOMATICo de S-SP y de P-SP, por lo tanto, para que el proceso SET SELECTION escoja las ocurrencias correctas de conjunto DBTG en este ejemplo, el programador primero debe (paso dos) dar a los elementos de datos NROS IN S y NROP IN P de la ATU los valores correctos.

Ahora se puede almacenar en la base de datos (paso 3).

ERASE (borre)

Suprime el actual de la unidad de ejecución.

Ejemplo:

Suprima la ocurrencia de S para el proveedor S4.

```
FIND ocurrencia de S para S4
ERASE [ALL] S
```

La palabra dentro de los corchetes es opcional, ésto significa que hay dos formatos de ERASE.

La proposicion ERASE ALL suprime el actual de la unidad de ejecución y todos sus descendientes; en el ejemplo, ERASE ALL suprimiría al rpoveedor S4 y todas sus remesas.

La proposición ERASE sin ALL, opera de acuerdo con el procedimiento descrito en los pasos que se presentan a continuación:

1.-Sea R el actual de la unidad de ejecución.

2.-Si existe un registro X que sea miembro MANDATORY de un conjunto DBTG T con propietario R, entonces ERASE falla. La base de datos queda exactamente en el estado que se encontraba antes del ERASE y se retira cualquiera de las marcas *suprime* y *desconectable*.

3.-Para cada registro C que sea un miembro OPTIONAL de un conjunto DBTG T con propietario R, X se marca como *decsonectable desde T*.

4.-Para cada registro X que sea miembro FIXED de un conjunto DBTG T con propietario R, los pasos 2 a 5 se repiten con X reemplazando a R.

5.-R se marca como *suprimible*.

6.-Todo registro marcado como *desconectable* se desconecta del conjunto DBTG indicado. Todo registro marcado como *suprimible* se desconecta de todos los conjuntos DBTG de los cuales sea miembro y luego se destruye. La operacón ERASE termina con éxito. El indicador de posición actual para el *actual de la unidad de ejecución* es nulo (no identifica ninguna posición).

MODIFY (modifique)

Actualiza el actual de la unidad de ejecución.
Ejemplo:

Adiciane 10 *al estado del proveedor* S4.

```
FIND ocurrencia de S para S4.
GETS
ADD 10 TO ESTADO IN S
MODIFY S
```

Para añadir 10 al estado S4, el usuario debe recuperarlo, incrementarlo en la ATU y luego devolverlo.

Así, la proposición MODIFY reemplaza el actual de la unidad de ejecución por valores tomados de la ATU.

Si MODIFY especifica un nombre de registro, el registro entero se reemplaza. Si especifica una lista de elementos de datos, sólo esos elementos se reemplazan.

CONNECT (conecte)

Conecta el actual de la unidad de ejecución a una ocurrencia de conjunto DBTG.

Para ilustrar los ejemplos siguientes, se introduce un tipo de conjunto DBTG nuevo CONJUNTOX (propietario X, miembro S, SET SELECTION IS BY APPLICATION; aquí, S es el tipo de registro proveedor, y X es algún tipo de registro nuevo arbitrario).

Ejemplo:

Conecte la ocurrencia de S para el proveedor S4 a la ocurrencia de CONJUNTOX *poseída por la ocurrencia de X de X.*

```
1:    FIND ...X
2:    FIND ocurrencia de S para S4
3:    CONNECT S TO CONJUNTOX
```

El proceso de conexión consiste en:

1: Localizar la ocurrencia requerida de CONJUNTOX,

2: Localizar la ocurrencia requerida de S, y

3: Ejecutar una proposición CONNECT para conectar la última con la primera.

La proposición CONNECT conecta al actual de la unidad de ejecución a una ocurrencia del conjunto DBTG especificado. Nótese que la clase de pertenencia de S en CONJUNTOX no debe ser AUTOMATIC a menos que también sea OPTIONAL.

DISCONNECT (desconecte)

Desconecta el actual de la unidad de ejecución de una ocurrencia de conjunto DBTG.

Ejemplo:

Desconecte la ocurrencia de S para S4 de la ocurrencia de CONJUNTOX *que la contiene.*

```
FIND ocurrencia de S para S4
DISCONNECT S FROM CONJUNTOX
```

La proposición DISCONNECT desconecta al actual de la unidad de ejecución de la ocurrencia del conjunto DBTG especificado que la contiene; la ocurrencia de registro aun existe en la base de datos, pero ya no es miembro del conjunto DBTG especificado.

Nótese, que S debe ser un miembro OPTIONAL de CONJUNTOX y que la ocurrencia de S para S4 debe actualmente ser un miembro de alguna ocurrencia de CONJUNTOX.

RECONNECT (reconecte)

Desconecta el actual de la unidad de ejecución de una ocurrencia de conjunto DBTG y lo conecta a otra ocurrencia del mismo tipo de conjunto DBTG.

Ejemplo:

Desconecte la ocurrencia de S *de la ocurrencia de* CONJUNTOX *que la contiene y conéctela a la ocurrencia de* CONJUNTOX *poseída por la ocurrencia* X *de* X.

```
1:    FIND ...x
2:    FIND ocurrencia de S para S4.
3:    RECONNECT S WITHIN CONJUNTOX.
```

La proposición RECONNECT desconecta el actual de la unidad de ejecución de la ocurrencia del conjunto DBTG especificado que la contiene, y luego lo conecta a alguna ocurrencia , tal vez la misma, de ese conjunto DBTG.

Esta última ocurrencia se determina de acuerdo con los criterios de SET SELECTION para el conjunto DBTG especificado. Nótese en el ejemplo que S no debe ser un miembro FIXED de CONJUNTOX, a menos que la ocurrencia de CONJUNTOX de la cual el registro se desconecta y la ocurrencia de CONJUNTOX en la cual se reconecta sean una y la misma.

FIND (encuentre)

Localiza una ocurrencia de registro existente y la establece como el actual de la unidad de ejecución. También actualiza otros indicadores de posición actual en la medida que sea necesario.

El formato básico de la proposición FIND es:

FIND expresión-de-selección-de-registro

donde, *expresión-de-selección-de-registro* (*e-s-r*) designa alguna ocurrencia de registro en la base de datos.

La función de FIND es localizar la ocurrencia designada y hacerla el actual de la unidad de ejecución y también el registro actual de todas las regiones y todos los conjuntos DBTG donde participa.

Hay seis formatos generales de *e-s-r* ,y por lo tanto, seis formatos de FIND, los cuales se mencionan a continuación:

1.-*Acceso dentro de un tipo de registro*

Este formato se necesita cuando el elemento de datos utilizado admite duplicados. Por ejemplo, para hallar todas las ocurrencias de registro de S en las que el valor de cantidad es LONDRES:

```
MOVE 'LONDRES' TO CIUDAD IN S
FIND ANY S USING CIUDAD IN S
MOVE 'NO' TO NO-ENCONTRADO
PERFORM UNTIL NO-ENCONTRADO = 'SI'
      GET S
      ...
      FIND DUPLICATE S USING CIUDAD IN S
END-PERFORM.
```

El FIND básico de formato (FIND ANY) proporciona acceso directo a una ocurrencia de registro por medio del valor de algún elemento (o alguna combinación) de datos para esa ocurrencia.

Las dos primeras proposiciones del ejemplo localizan *alguna* ocurrencia de S con el valor de ciudad LONDRES.

La tercera proposición da el valor inicial 'NO' al elemento de datos NO-ENCONTRADO.

El ciclo PERFORM recupera la ocurrencia de S recién encontrada, la procesa y luego intenta hallar un *duplicado* de esa ocurrencia de S (la construcción PERFORM/END-PERFORM es parte de las extensiones de programación estructurada de COBOL). El ciclo se repite hasta que NO-ENCONTRADO se ajuste a 'SI'.

La codificación garantiza que se encuentran todas las ocurrencias de S requeridas y que ninguna ocurrencia se encuentra dos veces.

La cláusula USING puede tomar la forma "USING elemento-de-datos, elemento-de-datos,...".

2.-*Acceso al propietario.*

Supóngase que el registro actual del conjunto P-SP es una ocurrencia particular de SP; entonces, hallar la correspondiente ocurrencia de P.

```
FIND OWNER WITHIN P-SP
```

La proposición FIND OWNER halla al propietraio en el tipo de conjunto DBTG especificado de ocurrencia de conjunto DBTG actual de ese tipo.

3.-*Acceso secuencial dentro de un conjunto o una region.*

Hallar los valores de NROP para las partes suministradas por el proveedor S4.

```
MOVE 'S4' TO NROS IN S
FIND ANY S USING NROS IN S
MOVE 'NO' TO NO-MAS
FIND FIRS SP WITHIN S-SP
PERFORM UNTIL NO-MAS = 'SI'
       GET SP
       (adicione NROP IN SP a la lista de resultados)
       FIND NEXT SP WITHIN S-SP
END PERFORM
```

FIND FIRST localiza la primera ocurrencia de SP dentro de la ocurrencia actual del conjunto DBTG S-SP.

FIND NEXT localiza la siguiente ocurrencia de SP dentro de la ocurrencia actual del conjunto DBTG S-SP, para cada interacción del ciclo.

El elemento de datos NO-MAS se ajusta a 'SI' cuando no se puede hallar ninguna ocurencia siguiente de S-SP.

4.-*Acceso secuencial dentro de un conjunto DBTG (primera forma).*

Hallar la cantidad de la parte P5 suministrada por el proveedor S1.

```
MOVE 'S1' TO NROS IN S
FIND ANY S USING NROS IN S
MOVE 'P5' TO NROP IN SP
FIND SP WITHIN S-SP CURRENT USING NROP IN SP
GET SP
(imprime CTD in SP)
```

El FIND localizado en la cuarta linea opera localizando la primera ocurrencia de registro SP dentro de la ocurrencia actual de conjunto DBTG S-SP que tenga un valor de NROP igual al valor de NROP IN SP en la ATU (es decir, un valor de NROP de 'P5').

Si CURRENT se omite en un FIND de éste tipo, la ocurrencia de conjunto DBTG que se va a recorrer se determina de acuerdo con la especificación SET SELECTION para el conjunto DBTG en cuestión.

En el ejemplo anterior, la proposición FIND ANY S USING NROS IN S pudo haberse omitido, si CURRENT también se hubiera omitido del otro FIND.

5.-*Acceso secuencial dentro de un conjunto DBTG (segunda forma).*

Hallar todas las remesas para el proveedor S1 donde la cantidad sea 100.

```
MOVE 'S1' TO NROS IN S
FIND ANY S USING NROS IN S
MOVE 100 TO CTD IN SP
FIND SP WITHIN S-SP CURRENT USING CTD IN SP
MOVE 'NO' TO NO-ENCONTRADO
PERFORM UNTIL NO ENCONTRADO = 'S1'
      GET SP
      ....
      FIND DUPLICATE WITHIN S-SP USING CTD IN SP
END-PERFORM
```

Las cuatro primeras proposiciones localizan la primera ocurrencia de SP para el proveedor S1 que contenga un valor de CTD de 100, es decir, la ocurrencia de SP 'S1/P5/100'. Nótese que se hace uso de un FIND del formato anterior.

La proposición siguiente da valor inicial 'NO' a NO-ENCONTRADO.

Dentro del ciclo PERFORM, el FIND DUPLICATE recorre la ocurrencia actual de S-SP en la dirección NEXT, empezando en el registro actual del conjunto DBTG S-SP y buscando luego la ocurrencia siguiente de SP que tenga el mismo valor de CTD que el registro actual.

El FIND del formato anterior (punto 4) no se puede usar aquí porque tan sólo localizaría otra vez a 'S1/P5/100', y en cambio, el FIND de la segunda forma, recorre hacia adelante de la posición actual. El ciclo se repite hasta que ocurra una condición de *no-encontrado*.

El efecto de la cláusula USING CTD IN SP en éste tipo de formato, es hacer que el DBMS busque la ocurrencia siguiente de SP con el mismo valor de CTD que el registro de conjunto DBTG actual, no el mismo valor que el elemento de datos CTD IN SP de la ATU.

6.-*Acceso mediante llave de base de datos.*

El FIND de éste tipo de formato se usa para encontrar el registro que tenga un valor especificado de llave de base de datos. Realmente el registro en cuestión debe haberse encontrado ya, puesto que la única manera de que la unidad de ejecución pueda descubrir el valor de la llave de base de datos del registro, es hacer que ese registro sea el actual de la unidad de ejecución.

Este formato difiere de los otros formatos de FIND en que su única función es actualizar la tabla de indicadores de posición actual, no requiere ningún acceso a la base de datos.

La expresión de *selección de registro* en un FIND de éste tipo de formato adopta la forma de un *identificador de llave de base de datos*. Un identificador de llave de base de datos, a su vez, puede tener dos formas:

PRIMERA FORMA.

Establece el registro actual del conjunto DBTG S-SP como el actual de la unidad de ejecución.

La forma general del *identificador de llave de base de datos* (en la primera forma) es:

CURRENT [nombre-de-registro] [WITHIN nombre]

donde nombre es un nombre de conjunto DBTG o un nombre de región.

Si la palabra WITHIN se omite, entonces el registro que se va a encontrar es el registro actual del tipo de registro indicado.

Si tanto el nombre de registro como la palabra WITHIN se omiten, el registro que se va a encontrar es el actual de la unidad de ejecución.

Si se especifican tanto el nombre de registro como el WITHIN, entonces el registro que se va a encontrar es el registro actual del conjunto o región que especifique el WITHIN.

SEGUNDA FORMA:

Esta forma depende de la noción de *listas de guarda*.(Las listas de guarda se estudiarán un poco más adelante.)

La sintaxis de la segunda forma de *identificador de llave de base de datos* es:

posición WITHIN nombre-de-lista-de-guarda

donde posición es FIRST o LAST.

Ejemplo:

```
FIND LAST WITHIN LISTA
```

LA FRASE RETAINING

El efecto de esta frase es evitar la actualización del indicador de posición actual para el conjunto DBTG en cuestión. En general, la actualización de las posiciones actuales puede suprimirse para todas las regiones abarcadas (recuérdese que un registro dado puede pertenecer a múltiples regiones), para el tipo de registro implicado, para cualquier tipo de conjuntos DBTG en cuestión o para cualquier combinación de los casos anteriores. El único indicador de posición actual para el que la actualización nunca se puede suprimir es el de actual de la unidad de ejecución.

Una frase de RETAINING puede especificarse en cualquier posición FIND o STORE.

Ejemplo:

```
FIND NEXT SP WITHIN S-SP
     RETAINING P-SP CURRENCY
```

LISTAS DE GUARDA

Una lista de guarda es un objeto con nombre cuya función es guardar una lista ordenada de valores de llave de base de datos. Una unidad de ejecución puede tener cualquier número de listas; las listas de guarda no forman parte de la base de datos, sino son locales para la unidad de ejecución que las usa. Se definen en la Data Division del programa, siguiendo a la proposición que *invoca* al subesquema.

Ejemplo:

```
DB PROVEEDORES WITHIN PROVEEDORES-Y-PARTES
LD LISTAA LIMIT IS 15
LD LISTAB LIMIT IS 20
```

Las listas de guarda LISTAA y LISTAB contendrán cada una, una lista ordenada de valores de llave de base de datos, y de esta manera, cada una designará un conjunto de registros de la base de datos. LISTA puede guardar hasta 15 de tales valores y LISTAB hasta 20. Los valores de base de datos se añaden a una lista de guarda por medio de la proposición KEEP y se suprimen por medio de la proposición FREE.

El formato general de KEEP es:

KEEP[identificador-de-llave-de-base-de-datos]USING nombre-de-la-lista-de-guarda

donde los posibles formatos para **identificador-de-llave-de-base-de datos** son:

```
CURRENT[nombre-de-registro][WITHIN (nombre de conjunto o región)]
y
(FIRST O LAST) WITHIN nombre-de-lista-de-guarda.
```

El efecto de la proposición KEEP es añadir al final de la lista de guarda especificada el valor de la llave de base de datos representados por el identificador de llave de base de datos. (o, si no se especifica tal identificador, el valor de llave de base de datos para el actual de la unidad de ejecución.)

Un valor dado de llave de base de datos puede aparecer en varias listas de guarda distintas, o varias veces en la misma lista de guarda, o ambas cosas.

El formato general de FREE es:

```
FREE identificador-de-llave-de-base-de-datos
```

o

```
FREE ALL [FROM nombre-de-lista-de-guarda]
```

En el primer formato, si el identificador de llave de base de datos es de una de las formas CURRENT, entonces el indicador de posición actual especificado se coloca en nulo< si es una de las formas de lista de guarda (FIRST o LAST), entonces el valor indicado se suprime de la lista de guarda indicada.

En el segundo formato, todos los valores en la lista de guarda indicada (o, si se omite FROM, en todas las listas de guarda) se suprimen.

READY

Habilita una región para el procesamiento, por ejemplo:

```
READY S-SP-P USAGE-MODE IS UPDATE
```

donde USAGE-MODE es RETRIEVAL (recuperación) o UPDATE (actualización). Cada uno de éstos puede calificarse como PROTECTED (protegido) o EXCLUSIVE (exclusivo), indicando los requerimientos de la unidad de ejecución con respecto al compartimiento de la región con unidades de ejecución concurrentes. PROTECTED significa que la región puede compartirse con recuperadores, mas no con actualizadores; EXCLUSIVE significa que no se puede compartir en absoluto.

FINISH

La proposición FINISH hace que una región disponible quede como no disponible, por ejemplo:

```
FINISH S-SP-P
```

COMMIT

Establece un puntro de sincronización para la unidad de ejecución. (La iniciación de la unidad de ejecución también se considera un punto de sincronización).

Todos los cambios a la base de datos hechos por la unidad de ejecución desde el punto de sincronización anterior se hacen visibles a las unidades de ejecución concurrentes y se garantiza que nunca se *deshagan*.

La última proposición ejecutada en la unidad de ejecución normalmente debe ser un COMMIT, porque en la terminación de la unidad de ejecución el DBMS automáticamente *deshace* todos los cambios de la base de datos hechos por la unidad de ejecución desde su último punto de sincronización.

ROLLBACK

Deshace todos los cambios a la base de datos efectuados por la unidad de ejecución desde su último punto de sincronización. La unidad de ejecución por si misma puede decidir emitir ROLLBACK si descubre algún error en la mitad del procesamiento de algún lote de entrada. La unidad de ejecución se sigue ejecutando después del ROLLBACK, por lo tanto, puede seguir haciendo mas actualizaciones que no serán deshechas.

5.4.-COMPARACION ENTRE LOS TRES MODELOS

En éste último tema se pretende dar una visión general de las caraterísticas de cada uno de los tres enfoques, así como mostrar sus ventajas y desventajas.

Un sistema de bases de datos debe ser capaz de representar dos tipos de objetos: *entidades y asociaciones*. No hay ninguna diferencia básica real entre los dos tipos de objetos. Los tres enfoques (relacional, jerárquico y de red) difieren en la manera en que permiten al usuario ver y manipular las asociaciones.

En el enfoque relacional por ejemplo, las asociaciones se representan como tuplas de relaciones, es decir con una serie de tablas. En los enfoques jerárquicos y de red, ciertas asociaciones (no todas) se representan por medio de ligas.

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

La diferencia entre los enfoques de red y jerárquico estriba en que con el primero, las ligas pueden combinarse para modelar asociaciones más complejas de muchos a muchos, mientras que ésto no es posible con el segundo; es decir, en el enfoque jerárquico los registros se organizan para formar conjuntos de árboles, en vez de gráficas arbitrarias.

Una relación puede considerarse como caso especial de jerarquía, es decir, una que es sólo raíz. Asimismo, una jerarquía puede considerarse como caso especial de red, es decir, una en la cual cada registro hijo tiene exactamente un registro padre.

En seguida se hace una comparación entre las relaciones y las redes porque están en competencia directa (a las jerarquías no se les mencionará de manera explícita, porque se les considerará simplemente cono formas restringuidas de redes).

Una gran diferencia entre las relaciones y las redes es que en un esquema relacional, el contenido de información total de la base de datos se representa por medio de una sola construcción de datos (la relación). En cambio, en un esquema de red, hay al menos dos construcciones de datos esenciales: el conjunto base y el conjunto abanico. En DBTG, en particular, hay cinco construcciones de datos:

-el tipo de registro (corresponde al conjunto base),
-el conjunto DBTG (corresponde al conjunto abanico),
-el conjunto DBTG singular,
-el ordenamiento,
-el grupo de repetición.

Las redes son menos fáciles de entender que las relaciones. Una posible explicación del aumento en la complejidad es el incremento en el número de construcciones básicas que el usuario debe entender y manejar.

A continuación se describen de manera general algunas características de cada enfoque por separado, así como sus desventajas.

JERARQUICO.

Caracterìsticas:

*Se representa a los datos como estructuras de ARBOL.

*El ARBOL representa una jerarquía de registros de datos.

*Procesamiento TOP-DOWN, navegacional.

*Relaciones entre registros: 1 padre, múltiples hijos. (Fig.5.17)

Desventajas:

*Anomalías de inserción.

*Anomalías de borrado.

*Anomalías de Actualización.

*Asimetría en la consulta.

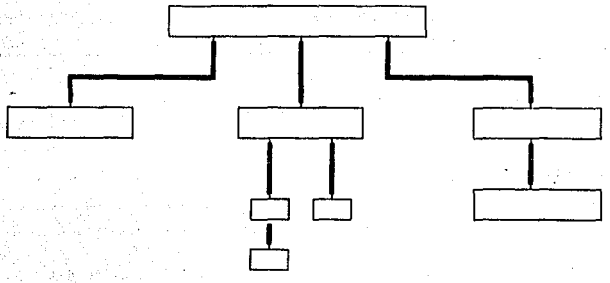


FIG.5.17. ESTRUCTURA DEL ENFOQUE JERARQUICO.

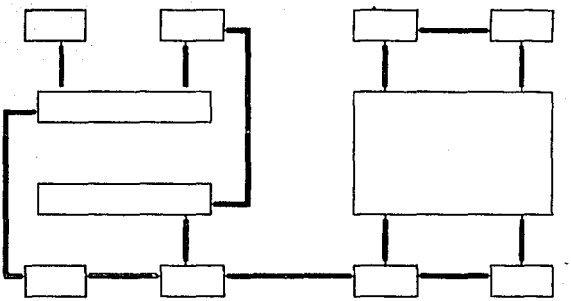


FIG.5.18. ESTRUCTURA DEL ENFOQUE DE RED.

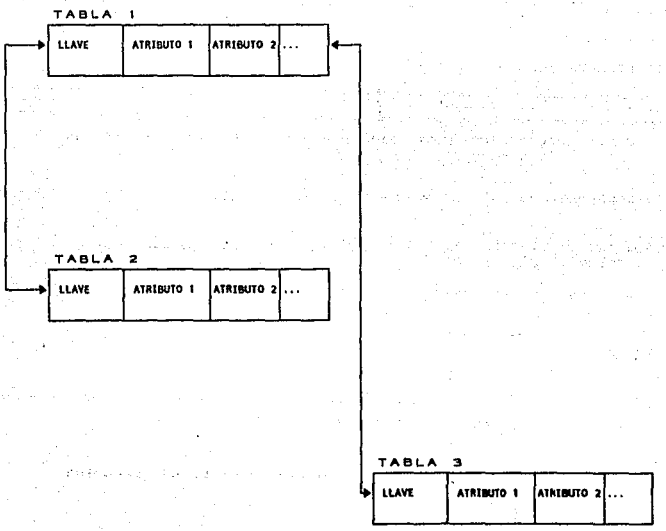


FIG.5.19.- ESTRUCTURA DEL ENFOQUE RELACIONAL

R E D

Características:

*Se representan los datos como registros ligados formando un conjunto de datos intersectados.

*Procesamiento multidireccional, navegacional.

*Cualquier tipo de relación entre registros puede ser modelada. (Fig. 5.18)

Desventajas:

*Difícil definir nuevas relaciones.

*Difícil de mantener. (Cualquier cambio requiere una descarga de los datos)

*Mucho overhead. (Desperdicio de recursos)

RELACIONAL

Características:

*Representación de datos a través de tablas.

*Desarrollo a través de herramientas de alta productividad.

*Capacidades relacionales completas.

*Flexibilidad:
-en el mantenimiento de las estructuras.
-en el mantenimiento de los datos.
-en el tipo de consultas.

*Diccionario de Datos Integrado.

Ventajas:

*Fácil de usar.

*Fácil obtener respuestas.

*Fácil insertar y actualizar datos.

*Fácil cambiar la estructura de los datos.

*Todas las consultas son posibles.

*Redundancia controlada con la normalización.

Sistemas que pueden considerarse representativos de los tres enfoques:

EL ENFOQUE JERARQUICO

-Sistema de Manejo de Información, IMS, de IBM.
-Mark IV, de Informatics.
-System 2000, de MRI.
-Sistema de Administración de Datos de Tiempo Compartido, TDMS, de SDC.

EL ENFOQUE DE RED

-Grupo de Trabajo de Bases de Datos (DBTG) de CODASYL y sus diversos comités sucesores.
-DMS 1100, de UNIVAC.
-IDMS, de Cullianane.
-TOTAL, de Cincom.
-DBOMP, de IBM.
-Integrate Dara Store, IDS, de Honeywell.

EL ENFOQUE RELACIONAL

-MAGNUM, de Tymshare.
-Query By Example, de IBM.
-System R, de IBM Research.
-INGRES, de la Universidad de California en Berkeley.
-ORACLE, DBASE, FOXPRO y en general los desarrollados para microcomputadores.

Los enfoques jerárquicos y de redes merecen atención por su gran importancia y porque ofrecen la ventaja de haber sido utilizados durante varios años, pues el modelo de datos relacional es relativamente reciente.

Los primeros sistemas de bases de datos estaban basados en el modelo jerárquico o en el de red; sin embargo, considerámos al enfoque relacional como la estructura de bases de datos con mayores ventajas y más fácil de manejar, ya que toda la información en la base de datos se representa usando una tabla, la cual es sencilla y muy familiar, además de que la explotación simétrica es posible porque toda la información se representa de la misma manera uniforme.

Aunque a largo plazo el enfoque relacional parece el mejor candidato como base de un lenguaje de bases de datos de propósito general, no hay duda de que las redes y las jerarquías seguirán existiendo por algún tiempo, por la sencilla razón de que ya se ha hecho una gran inversión en esos sistemas.

Muchas autoridades piensan que en el futuro se crearán uno o más sistemas a gran escala basados en el enfoque relacional.

La idea de usar un solo lenguaje bien estructurado como interfaz de programación común para diversos sistemas parece muy atractiva, podría simplificar mucho los problemas de comunicación entre los usuarios de los diferentes sistemas, podría atenuar los problemas de educación y podría ayudar a la migración de programas y programadores de un sistema a otro.

CAPITULO 6.- PLANEACION DE LA BASE DE DATOS. EL DISEÑO LOGICO.

6.1.- ANTECEDENTES.

Muchas de las primeras computadoras utilizadas en los negocios fueron instaladas para procesar unos cuantos trabajos de gran volumen. Cada aplicación tenía su propio archivo maestro, datos de entrada y su propio programa de proceso para actualizar el archivo y suministrar la información.

Este método de archivo creó muchos problemas, por ejemplo:

a) Redundancia de los datos, ya que muchos de los datos básicos se incluían en muchos archivos diferentes;

b) Problemas de actualización de los archivos, ya que cuando los datos sufren cambios, cada archivo debe ser actualizado. Esto puede crear confusión cuando un archivo es actualizado y otro no., y

c) Falta de independencia de programas-datos, esto es porque los programas de aplicación orientados a los archivos normalmente contienen sentencias como "picture","format" o "data"., que definen exactamente cada dato que va a ser procesado, esto ocasiona que un elemento de datos tenga diferentes formatos y cuando exista la necesidad de modificar, borrar o sustituír el formato de los datos, el programa de aplicación también debe de ser cambiado.

A finales de la década de 1960, algunos diseñadores de sistemas insatisfechos con los problemas ocasionados por el método de archivos, empezaron a buscar diversas formas de consolidar las actividades utilizando un método de bases de datos. La Base de Datos tiene la finalidad de reducir éstas dificultades. Así, se solucionaron muchos de los problemas, ya que esta base de datos se localiza en un dispositivo de acceso directo. Las transacciones se introducen al sistema una sola vez. Los datos son un recurso neutral respecto a cualquier programa y los elementos específicos de los datos están al alcance de todas las aplicaciones autorizadas y los usuarios de la base de datos. Todos los registros de la base de datos que pueda afectar una transacción se actualizan en el instante en que entra la transacción al sistema.

A un conjunto integrado de programas de apoyo y estructuras de archivo que apoyan a una base de datos, a sus necesidades lógicas y a una interface entre usuarios y programas se le denomina *sistema de base de datos.*

Un sistema de base de datos es la combinación de programas y archivos que se utilizan conjuntamente. Un conjunto integrado de programas para dar apoyo a bases de datos puede formar un *sistema de manejo de base de datos.*

Diseñar los sistemas para dar servicio eficiente a varios usuarios y a la vez reducir el uso de los recursos de cómputo es una tarea difícil que implica complejas consideraciones.

El objetivo del diseño de una base de datos es generar un conjunto de esquemas que nos permitan almacenar la información con un mínimo de redundancia y que además la información sea fácil de recuperar.

Es muy importante el realizar un buen diseño de la base de datos, ya que de lo contrario se tendrán algunas propiedades indeseables como: repetición de la información, incapacidad para representar cierta información y pérdida de información.

Para diseñar y desarrollar un sistema se requieren las siguientes habilidades:

1.-La capacidad para examinar una petición del usuario y determinar si debe ser considerada la oportunidad para resolver el problema con la computadora.

2.-El conocimiento para recopilar e interpretar los datos, su relación con los sistemas de información y el cómputo.

3.-El criterio para determinar dónde es aconsejable utilizar la computadora y dónde son más efectivos los procedimientos manuales.

4 -El conocimiento para seleccionar los mejores métodos de entrada de datos, almacenamiento, acceso, proceso y salida.

5.-Tener un conocimiento interno del desarrollo de software, métodos de prueba y estrategias de puesta en marcha.

El diseño de un sistema de información produce los elementos que establecen cómo el sistema cumplirá los requerimientos. A menudo los especialistas se refieren a esta etapa como el *diseño lógico*, en contraste con el desarrollo del software de programas, que se conoce como *diseño físico*.

En la generalidad de las empresas en que se utilizan y manejan las bases de datos, un grupo del personal de manejo de base de datos revisa su diseño y desarrollo. El grupo define el esquema, mantiene el diccionario de datos y refuerza los estándares para los mismos (como nombre, tipo, longitud y utilización).

Para revisar la necesidad de un sistema de base de datos es posible comparar sus beneficios y costos con los beneficios y costos que se presentan cuando se sobrepone una estructura de archivo a un sistema de hardware.

Un *sistema de archivo* organiza la capacidad de almacenamiento de datos que proporciona el hardware. El hardware se divide en archivos los cuales se asocian con un usuario particular. Ahora estos usuarios pueden trabajar en aparente aislamiento.

Un *sistema de base de datos* organiza la capacidad de almacenamiento de archivos que proporcionan los sistemas de archivo. Los vínculos entre los elementos de las relaciones se vuelven accesibles. Los datos pueden ser compartidos por usuarios en cooperación.

No es esencial que el almacenamiento de base de datos en una computadora sea controlado mediante un sistema de base de datos.

Cuando se almacenan grandes cantidades de datos interrelacionados que sean de interés para diversos grupos de usuarios, se vuelve indispensable un sistema de base de datos.

A continuación se muestra una lista de los objetivos generales para un diseño de sistema de bases de datos:

1.-La capacidad de hacer referencia a elementos dato sin tener conocimento de la estructura de registro o de archivo.

2.-La capacidad de cambiar el contenido de registros o de archivos y la estructura sin afectar los programas existentes en la base de datos.

3.-La capacidad para manejar archivos relacionados dentro de un marco general, de manera que los datos en archivos separados puedan seguir siendo consistentes y pueda evitarse la redundancia excesiva en actualización y almacenamiento.

4.-Una descripción de la base de datos integrando diversos puntos de vista (visiones), de manera que ésta descripción pueda volverse un medio de comunicación entre quienes generan datos y quienes buscan información.

En el momento de la implantación de los sistemas, el problema central es siempre el mismo:

" Los usuarios dicen que los diseñadores de sistemas no programaron lo que ellos (los usuarios) querían. Los encargados del procesamiento de datos dicen : les hemos dado a ustedes exactamente lo que deseaban, pero esos requerimientos ya no son válidos."

Lo ideal es que los usuarios expresen sus puntos de vista y trabajen en cooperación con el personal de procesamiento de datos para lograr un sistema óptimo.

Las ventajas y desventajas que deben ser consideradas por los diseñadores de los sistemas de bases de datos se resumen a continuación.

VENTAJAS

1.-Pueden necesitarse menos programas de aplicación y menos informes regulares que contengan información de referencia si los usuarios pueden tener acceso directo a la base de datos.

2.-Es factible una mejor integración (y menos duplicidad) de los datos que se originan en los diferentes puntos.

3.-Es posible la preparación más rápida de la información para soportar las tareas no recurrentes y las condiciones cambiantes.

4.-Es posible ahorrar en el costo del desarrollo de nuevas aplicaciones, así como en los costos de entrada de los datos y en su almacenamiento.

5.-Pueden ocurrir menos errores (e incrementarse la integridad de los datos) cuando varios registros pueden actualizarse en forma simultánea.

LIMITACIONES

1.-Se necesita hardware y software más complejos y caros.

2.-Puede requerirse un largo período de conversión, elevados gastos de captación y habilidades mayores en quienes son responsables del sistema de base de datos.

3.-La gente puede negarse a adoptar cambios significativos en los procedimientos de proceso de datos.

4.-Los datos sensibles en los dispositivos en línea pueden llegar a personas sin autorización.

5.-Las fallas de hardware o de sotware pueden ocasionar la destrucción de información vital en la base de datos.

6.2.- DISEÑO CONCEPTUAL.

6.2.1. PLANEACION Y ANALISIS DEL SISTEMA

Antes de que pueda considerarse el diseño de un sistema, el proyecto requerido debe examinarse para determinar precisamente lo que desea la empresa, ya que muchas veces los requeriminetos no están establecidos claramente.

Es necesario conocer más acerca de cómo se manejan las operaciones en el sistema actual; por ejemplo, saber qué formas se utilizan para almacenar la información en forma manual, además de saber que informes (si existen algunos) se producen y para qué se utilizan; por lo tanto, se debe buscar la información de dichos informes.

También se necesita encontrar de dónde se origina ésta información. Es decir, se debe comprender la forma en que trabaja el sistema actual, cuál es el flujo de información por el que atraviesa el sistema. También es importante saber por qué la empresa desea cambiar sus operaciones actuales.

Sólo después de recabar todos estos datos se puede comenzar a definir cómo y dónde se puede beneficiar un sistema de información basado en la computadora y que sirva a todos los usuarios del sistema.

Cuando el proyecto está claramente establecido, el siguiente paso es hacer un estudio de factibilidad, es decir, la posibilidad de que el sistema sea benéfico a la empresa. Existen tres aspectos a considerar:

1.-Factibilidad técnica.-¿Puede desarrollarse el sistema con el equipo actual, tecnología de software y personal disponible?. Si se requiere nueva tecnología, ¿qué probabilidades hay de que pueda desarrollarse?

2.-Factibilidad económica.-¿Existen beneficios suficientes en la creación del proyecto para hacer que los costos sean aceptables? o ¿son tan altos los costos como para que el proyecto no deba llevarse a cabo?

3.-Factibilidad Operativa.- ¿Se utilizará el sistema si se desarrolla y pone en marcha?,¿habrá resistencia de los usuarios, que los posibles beneficios se reducirán?.

No todos los proyectos requeridos son factibles. De hecho, algunas compañías reciben tantas peticiones de proyectos de los empleados que solamente se llevan a cabo unas cuantas. Después de que se aprueba la requisición de un proyecto; se estima su costo, la prioridad, el tiempo de terminación y los requerimientos de personal.

Algunas veces, el tiempo que lleva desarrollar una opción, comparada con otras, será el aspecto más difícil. Los costos y beneficios financieros son importantes de determinar. Los analistas de sistemas pueden recomendar, pero la gerencia que va a pagar y utilizar los resultados es la que realmente decide.

Una vez que se toma la decisión, se desarrolla un plan para poner en marcha la recomendación. El plan incluye todas las características de diseño de sistemas, como son :

-Necesidades nuevas de captación de datos,
-Especificaciones de archivos,
-Procedimientos de operación y
-Necesidades de equipo y personal.

En una base de datos la documentación más importante es su modelo. El modelo de la base de datos determinará los procesos necesarios para la creación y el mantenimiento de archivos, y para la recuperación de información. El esquema ampliado con observaciones acerca de los vínculos en el mundo real, las reestricciones de conexión y las definiciones de los dominios de variable y su representación, son el depósito formal para la documentación del modelo de base de datos, conforme se afina el diseño.

El diseño de un sistema generalizado de base de datos se vuelve mucho más complicado debido a la ausencia de objetivos del usuario y de modelos específicos de base de datos.

Por lo general, los problemas en una base de datos no se reconocen hasta que se ha recolectado cierto volumen de datos. Es importante la recolección de la información utilizada para generar los programas de procesamiento que trabajarán con la base de datos. Esta información puede utilizarse para los procesos de planteamiento de modelos.

Ya que las transacciones también se incluirán en el posible diseño, también se documentan los pasos de procesamiento aplicados para información. Deberán estudiarse muchos procedimientos existentes a fin de obtener una imagen compuesta de las acciones que se realizan. Se recolecta información acerca de las fuentes y el destino de los datos, la frecuencia de uso, el tiempo deseado de respuesta y de ser posible, las condiciones de precisión.

El análisis de procedimientos tiene que realizarse con la debida atención al flujo real de la información. El hecho de que ciertos informes producidos contengan un elemento dato específico no significa necesariamente que el elemento se está utilizando.

A menudo existen medios informales de distribución de datos que no son obvios dentro de un análisis de sistema que dependa por completo de documentos existentes. Hablar es un importante y flexible tipo de comunicación no formal.

El tipo, la actividad y la cantidad de usuarios y de datos deben cuantificarse antes de que pueda realizarse cualquier diseño específico del sistema. La determinación de éstos parámetros es una responsabilidad de la gerencia.

Un problema que constantemente se presenta es que el grupo de programadores emite opiniones acerca de costos y efectividad de las alternativas sin realizar un análisis, utilizando a menudo un enfoque dogmático, basando algunas veces las opiniones en el interés personal.

El análisis de sistemas es el proceso que sirve para recopilar e intérpretar los hechos, diagnósticar problemas y utilizar estos hechos a fin de mejorar el sistema.

El análisis especifica qué es lo que el sistema debe hacer y cómo alcanzar el objetivo.

El punto clave del análisis de sistemas se consigue al adquirir un conocimiento detallado de todos los factores importantes dentro del área del sistema a desarrollar.

Los analistas deben estudiar el proceso que actualmente se efectúa para contestar estas preguntas clave :

1.-¿Qué se está haciendo ?
2.-¿Cómo se está haciendo ?
3.-¿Qué tan frecuentemente ocurre ?
4.-¿Qué tan grande es la cantidad de transacciones o decisiones?
5.-¿Qué tan bien se lleva a cabo la tarea ?
6.-¿Existe algún problema ?
7.-¿Si el problema existe, qué tan serio es ?
8.-¿Si el problema existe, cuál es la causa principal ?

6.2.2. *FORMULACION DE ESQUEMA Y SUBESQUEMAS*

El diseño general de la base de datos se llama *esquema de la base de datos*. Estos esquemas se alteran muy raras veces.

Para transformar un modelo en un sistema que opere es necesario describir el modelo en una forma que se preste a implantación. A tal descripción se le denomina *esquema* y al lenguaje empleado para describirlo se le llamará *lenguaje de esquema*.

Es necesario que los esquemas incluyan detalles prácticos que pudieron haberse ignorado en los modelos.

Un esquema define inicialmente la estructura de la base de datos y pone esta reestricción a la disposición de los usuarios de la misma base de datos. Si se utiliza un sistema de manejo de base de datos, el esquema se empleará para controlar automáticamente la ejecución de los programas de transacción que operan a dicha base de datos.

La información referente al tipo de dato almacenada en el esquema puede utilizarse en las transacciones de procesamiento para dirigir el cálculo. Las consultas pueden utilizar las especificaciones de conexión en el esquema, para localizar datos sucesores.

A los predecesores de los esquemas se les denomina *diccionarios de datos* y *directorios de base de datos*. Son una importante parte del esquema las descripciones de los elementos dato.

El esquema estará codificado de manera que el sistema de base de datos pueda leerlo y utilizar los programas generalizados para controlar el flujo de los datos a los archivos que contiene la base de datos. El esquema estará almacenado en el sistema, para que resulte accesible cuando se necesite.

A fin de crear un esquema, se necesitarán servicios de lenguaje de esquema separados de los servicios de lenguaje que se utiliza cuando se maneja la base de datos. A menudo los procesadores para el lenguaje de esquema y el lenguaje de manejo de datos son diferentes.

En la figura 6.1 se presenta en perspectiva la idea de esquema. Durante los cálculos, el esquema se utiliza tanto para colocar adecuadamente los datos de llegada en los archivos como para localizar los datos solicitados. Los usuarios de la base de datos no modifican el esquema durante las operaciones de dicha base de datos.

UN EJEMPLO DE ESQUEMA

Un esquema muy simple para una base de datos que utilice sólo un archivo secuencial se muestra a continuación. Este ejemplo utiliza el sistema TYMSHARE RETRIEVE. El sistema opera en forma interactiva con una terminal. En la figura 6.2 se muestra un ejemplo de la definición inicial de un esquema. En este punto no existen datos en el archivo de datos.

El esquema se ha guardado en un archivo separado, de manera que pueda ser utilizado por algún proceso subsecuente. Ahora el sistema está listo para recibir datos y en la figura 6.3 se muestra esta fase.

La información proveniente del esquema se utiliza para colocar en el archivo cada campo en forma adecuada. Ya pueden manipularse estos datos utilizando instrucciones de cómputo y declaraciones de selección. En la figura 6.4 se encuentra la lista de comandos de que dispone el usuario. Una sesión completa incluiría la recuperación del esquema, el agregado de registro, el cambio de campos de datos, el cálculo selectivo de campos resultantes y una impresión final. (ver figura 6.5).

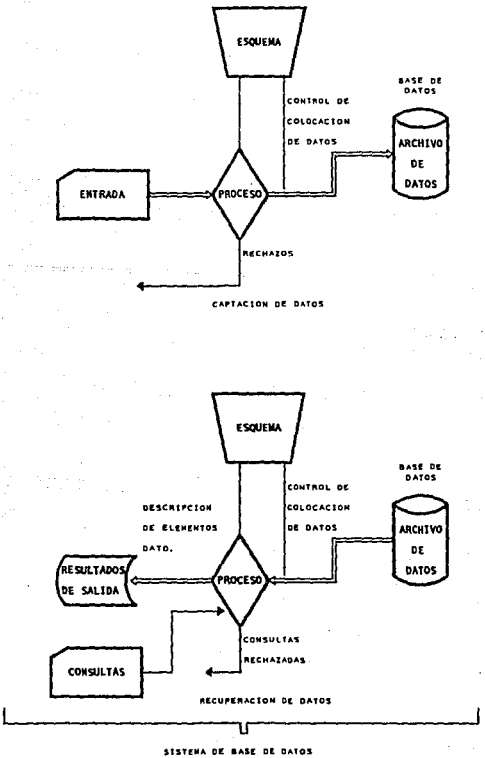


FIG.6.1- LUGAR DE UN ESQUEMA EN ENTRADA Y SALIDA.

```
.RETRIEVE ←┘

POR FAVOR INDIQUE EL NOMBRE DE SU BASE DE DATOS: EMPLEADO ←┘

NECESITO CONOCER LA ESTRUCTURA DE SU BASE DE DATOS.
POR FAVOR DESCRIBA CADA ELEMENTO.

ELEMENTO    NOMBRE          TIPO

    1       EMPLEADO,       20,C ←┘
    2       TRABAJO,        15,C ←┘
    3       SUELDO,          9,N ←┘
    4       DIRECCION,      40,C ←┘
    5       CIUDAD,         20,C ←┘
    6       ←┘              EL CAMBIO DE LINEA (RETURN) CONLCUYE LA DESCRIPCION DE LA ESTRUCTURA.

EMPLEADO CONTIENE AHORA 0 REGISTROS DE 7 CARACTERES.
EMPLEADO 'STR.D' CONTIENE AHORA LA ESTRUCTURA BASE.
```

FIG. 6.2.- CREACION DE ESQUEMAS.

```
.APPEND ←┘

EMPLEADO    TRABAJO  SUELDO    DIRECCION              CIUDAD
GOMEZ LUIS,MENSAJERO,831983,NORTE 58 3536 RIO BLANCO,MEXICO D.F. ←┘
PEREZ ANGEL,ANALISTA,1922990,YURIRIA 28-A REYES IXTACALA,EDO.MEXICO ←┘
RUIZ NORMA,PROGRAMADOR,1453242,ZARAGOZA 124 CONSTITUCION,MEXICO D.F. ←┘
MORA JULIO,ANUNCIADOR,791726,VALLEJO 1830 PRADOS VALLEJO,MEXICO D.F. ←┘
←┘          EL COMANDO APPEND CONCLUYE CON UN CAMBIO DE LINEA (RETURN).

4 REGISTROS PROCESADOS
```

FIG. 6.3.- INTRODUCCION DE DATOS A LA BASE.

COMANDOS DE MANIPULACION EN UNA BASE DE DATOS

```
APPEND
CHANGE ATRIBUTO (FOR condiciones)
COUNT (FOR condiciones)
DELETE (FOR condiciones)
LIST (FOR condiciones)
PRINT (atributos) (FOR condiciones)
REPLACE atributo WITH expresion (FOR condiciones)
REPORT (FOR condiciones)
SAVE archivo
SORT BY atributo llave (FOR condiciones)
SUM expresion (FOR condiciones)
```

LAS CLAUSULAS (FOR condiciones) SON OPTATIVAS.

FIG.6.4.- COMANDOS RETRIEVE.

```
RETRIEVE ←┘

POR FAVOR INDIQUE EL NOMBRE DE SU BASE DE DATOS: PERSONAL ←┘   LA BASE DE DATOS SE CREO EN UNA SESION
                                                               ANTERIOR DE RETRIEVE.
PERSONAL CONTIENE AHORA 4 REGISTROS DE 48 CARACTERES.

.STRUCTURE ←┘   SE EXHIBE LA ESTRUCTURA DEFINIDA ANTERIORMENTE.

ELEMENTO      TIPO    ANCHO   NOMBRE

  1             C       20    EMPLEADO
  2             C       11    SEG.SOCIAL
  3             N        6    SUELDO
  4             N        3    HORAS
  5             N        7    PAGO

.APPEND ←┘   SE AGREGAN DOS REGISTROS A LA BASE DE DATOS.

EMPLEADO   SEG.SOCIAL  SUELDO  HORAS PAGO

TORRES NOE,347-72-6528,1558,40,0 ←┘

MONTES ALAN,462-99-3369,855910,40,0 ←┘

←┘

2 REGISTROS PROCESADOS

.SORT BY EMPLEADO ←┘   LA BASE DE DATOS SE CLASIFICA
                       ALFABETICAMENTE PARA EMPLEADO

PERSONAL 'OLD' CONTIENE SU BASE DE DATOS NO CLASIFICADA.
PERSONAL AHORA ESTA CLASIFICADO.
SE CONSERVA PERSONAL 'OLD'? NO

.LIST ←┘   SE EXHIBE TODA LA BASE DE DATOS.

NUMREG  EMPLEADO          SEG.SOCIAL     SUELDO   HORAS    PAGO

  1     ALZUA ANDRES      469-20-9531      1666     40       0
  2     BRAVO JOSE        202-46-9277      2010     40       0
  3     LOPEZ ANGEL       519-45-6218      3112     40       0
  4     MONTES ALAN       462-99-3369    855910     40       0
  5     MURILLO ISAAC     357-48-3158    965420     40       0
  6     TORRES NOE        347-72-6528      1558     40       0

  6 REGISTROS PROCESADOS
```

FIG.6.5.- SESION DE BASE DE DATOS (PRIMERA PARTE)

```
.IN 1,3 CHANGE HORAS <┘     EL ELEMENTO HORAS SE MODIFICA EN LOS REGISTROS 1 Y 3.
HORAS
 40
48 <┘
 40
44 <┘
2 REGISTROS PROCESADOS
.IN 2 CHANGE SUELDO <┘     EL ELEMENTO SUELDO SE MODIFICA EN EL REGISTRO 2.
SUELDO
 2010
 2594 <┘
1 REGISTRO PROCESADO
.REPLACE SUELDO WITH (SUELDO*40)+(SUELDO*2*(HORAS-40)) FOR SUELDO < 500000 <┘
                                   SE CALCULA UN NUEVO VALOR DE PAGO
                                   PARA LOS EMPLEADOS QUE TRABAJAN
                                   POR HORAS EMPLEANDO EL COMANDO
                                   REPLACE.
4 REGISTROS PROCESADOS
.REPLACE PAGO WITH SUELDO FOR SUELDO > 500000 <┘   NUEVAMENTE SE EMPLEA EL COMANDO REPLACE PARA OBTENER
                                                   EL PAGO DE LOS EMPLEADOS QUE TRABAJAN POR MES.
2 REGISTROS PROCESADOS
.PRINT EMPLEADO,PAGO <┘    SE EMPLEA PRINT PARA LISTAR SOLO LOS ELEMENTOS
                           EMPLEADO Y PAGO. OBSERVESE QUE SE IMPRIMEN ENCABEZADOS.
EMPLEADO                PAGO

ALZUA ANDRES            93296
BRAVO JOSE              80400
LOPEZ ANGEL             149376
MONTES ALAN             855910
MURILLO ISAAC           965420
TORRES NOE              62320

6 REGISTROS PROCESADOS

.SUM PAGO <┘     EL COMANDO SUM CALCULA EL VALOR TOTAL
                 DE PAGO PARA LA BASE DE DATOS.
SUM IS  2206722

6 REGISTROS PROCEADOS
.QUIT <┘     EL COMANDO QUIT REGRESA EL CONTROL AL EXECUTIVE.
```

FIG. 6.5.- SESION DE BASE DE DATOS (CONTINUACION)

El *subesquema* es la definición lógica de los datos a partir de la base de datos que utilizará el programa, consiste en los nombres de los datos y descripciones y es un subconjunto del esquema. Para cada base de datos existe un esquema individual, pero puede haber muchos subesquemas. Cada aplicación de sistemas de información que utilice la base de datos puede tener un subesquema diferente.

En una gran base de datos, el esquema puede ser de grandes proporciones. Ya que un usuario o programa no requieren toda la información en el esquema, los datos contenidos en él pueden categorizarse y seleccionarse de acuerdo con varias dimensiones :

-Nivel funcional del usuario del esquema.
-Adaptaciones del lenguaje de la computadora anfitriona.
-Responsabilidad y propiedad de los datos.
-Función de procesamiento.
-Localización de fragmentos almacenados de la base de datos.

Todas estas dimensiones pueden utilizarse para dividir los esquemas en subesquemas que restrinjan al usuario o a la base de datos a un subconjunto de datos y funciones, dependientes de la responsabilidad, necesidad de saber y localización.

Un usuario interesado en obtener información de la base de datos tal vez sólo necesite una descripción externa del contenido de dicha base.

En la figura 6.6 se muestra un ejemplo de una definición simple de subesquema. Una VIEW (visión) de las relaciones TABLE (tabla) en IBM SQL/DS permite la proyección y el reacomodo de las columnas de atributos, la selección de renglones y la especificación de variables obtenidas. Las VIEW en SQL/DS nunca crean relaciones, todo el mapeo se realiza en el momento en que se utiliza la visión.

Un *esquema externo* (subesquema basado en visión) contiene las anotaciones necesarias para el procesamiento de datos en el área de interés. Los esquemas externos pueden trasladarse para reflejar traslapes en los modelos de datos.

Este concepto de esquema externo se obtiene a partir del esquema conceptual general que representa todo el modelo de la base de datos. Es aquí donde se describen todos los vínculos. Su función principal es el diseño y la generación de esquemas.

El manejo operativo de los archivos almacenadops hace necesario ubicar más información en un *esquema interno*. Este esquema define en dónde están colocados los valores de atributos de la base de datos y cómo se obtiene acceso a ellas.

Un esquema de base de datos puede organizarse por atributo o por característica: *nombres, dominios, títulos, control de acceso a los datos, etc.* Ciertos procesos de la base de datos no requerirán todas las características que se han asociado con cada elemento dato., por ejemplo, si los datos sólo van a moverse, lo único que se requiere es la longitud y el conteo.

DADA UNA RELACION NINOS CON DATOS EN MEDIDAS TRADICIONALES :

```
NINOS : RELATION

 id_nino, tutor, clase, rango, edad_n, peso, ano_ingreso;
```

Se define una vision para la enfermera de la escuela, que piensa en unidades metricas modernas.

```
CREATE VIEW Enfermera

          (nino, edad, estatura_cm, peso_kg   )

   AS SELECT  id_nino, edad_n, estatura * 1.54, peso * 0.4536

         FROM NINOS;
```

FIG.6.6.- UNA DEFINICION SOL DE SUBESQUEMA EXTERNO.

La intención es que el esquema y el lenguaje de descripción del esquema puedan ser utilizados por programas escritos en distintos lenguajes de computación. Se diseñan variaciones de lenguajes en subesquemas que se ajusten a la sintaxis, la semántica y la potencia del lenguaje principal o lenguaje de la computadora anfitriona.

El administrador de la base de datos tiene la responsabilidad de la operación exitosa del sistema. Aun cuando la integridad estructural y la adecuación de las interfases entre esquemas externos, conceptuales e internos, hacen necesaria una visión central, puede resultar indispensable delegar la responsabilidad del contenido de datos a aquellos más próximamente asociados con un submodelo particular de la base de datos.

El uso de múltiples esquemas externos de la base de datos está relacionado con la asignación de responsabilidades en lo referente al mantenimiento de los datos almacenados. La *propiedad de los datos* es otro atributo que posiblemente deba especificarse en el lenguaje de esquema o ser parte de la especificación de privilegios de acceso.

Una vez que la base de datos se encuentra en operación, cualquier modificación al esquema podría provocar que muchos programas fallaran.

La *independencia de los datos* es objetivo importante del empleo de las bases de datos. El usuario de una gran base de datos está comunmente atado al hardware que da apoyo a los archivos y programas de esa base de datos. Por ejemplo, si se ha utilizado en una base de datos referencia apuntador, resulta sumamente difícil traducir en forma directa estas referencias a otro sistema. Se presentan dificultades más serias cuando en un sistema el acceso a los datos se hace mediante índices y reside en archivos secuenciales o aleatorios, y en otro sistema esto se realiza mediante enlaces.

A la independencia deseable de las aplicaciones con respecto a los sistemas de hardware y software se le ha denominado *independencia de los datos*. Sin embargo, es posible distinguir tres dependencias que resulta necesario resolver :

Dependencia de los datos.-

Las instrucciones de programa dependen de la puesta en código de los elementos dato. Son factores determinantes el tamaño de los campos elementales, representaciones de punto flotante y valores de apuntador.

Dependencia de la estructura.-

La lógica del programa depende de las facilidades para segmentar registros, manejar nidos y proporcionar y seguir enlaces entre archivos.

Dependencia del programa.-

No se refleja en los esquemas la dependencia en las consideraciones realizadas en los programas referentes al empleo y estructura de los datos. Aquellos aspectos de los datos que no son esenciales para los problemas de almacenamiento y manipulación de la información pero que proporcionan conveniencia y eficiencia de programación, pueden asignar estructuras de datos en forma inesperada.

El esquema es una herramienta que ayuda a lograr la independencia de los datos. Para lograr independencia en la estructura es necesario describir esta estructura en forma estandarizada. La independencia de programación se logra sólo si los programas utilizan en su totalidad la capacidad de independencia de los datos y de estructura. Aún no está claro qué tanto de independencia puede lograrse conservando la eficiencia máxima en grandes bases de datos.

6.2.3. EVALUACION DEL SISTEMA

El esquema, como lo escribió quien organizó la base de datos, debe traducirse para que lo emplee un sistema de manejo de base de datos. Las elecciones de asignación son las alternativas usuales de compilación comparada con la interpretación.

En el medio de la base de datos, la compilación es equivalente a emplear toda la información contenida en el esquema cuando se crean los programas de aplicación; entonces puede descartarse el esquema. Por otra parte, la interpretación es equivalente al uso de un programa general que, cuando se le llama para que realice manipulaciones de base de datos, consulte el esquema para encontrar elementos dato y determinar sus vínculos.

El usuario y quien desarrolla el sistema, perciben la organización de los datos a través del esquema que se les presenta. El uso de la base de datos dependerá de los conceptos apoyados por el esquema, ya que éstos conceptos son necesarios para que el usuario pueda manipular la base de datos.

EVALUACION DE LAS ALTERNATIVAS PARA LA TRADUCCION DEL ESQUEMA.

Los beneficios de *compilar* la información del esquema son los siguientes:

1.-Alta eficiencia durante el tiempo de procesamiento, debido al empleo de código real de máquina.

2.-Pueden escribirse y compilarse programas especiales que se ocupen de situaciones especiales. Se dispone de todo el repertorio de procesamiento de la computadora, incluyendo múltiples lenguajes de compilación y sus bibliotecas.

3.-No se requiere espacio explícito para contener el esquema durante el tiempo de procesamiento.

4.-En los lenguajes existen facilidades de programación para apoyar este enfoque. Una de ellas es la capacidad de incluír módulos fuentes provenientes de un archivo de esquema, y otra es la adecuación de programas proporcionados mediante el empleo de las capacidades de macroexpansión.

Las desventajas del enfoque compilado son las siguientes :

1.-Todos los programas tendrán que regenerarse cuando cambia el esquema y será necesario escribir nuevos programas cuando se especifiquen nuevos datos y vínculos.

2.-Es difícil controlar cuáles son los programas que utilizan completamente el esquema y que no dependen de conocimiento interno específico de la organización de datos.

3.-El beneficio de no requerir almacenamiento para el esquema es solo aparente, ya que toda la información requerida se almacena dentro de los programas de procesamiento.

4.-La mayoría de los compiladores y lenguajes de alto nivel no tienen la capacidad de definir estructuras de base de datos demasiado complejas.

5.-Puede resultar difícil adaptar un compilador existente para que genere código para algunas de las nuevas características de hardware que se están desarrollando para ayudar a las bases de datos.

Los beneficios de la *interpretación* son los siguientes :

1.-Los cambios orientados a los datos necesitan traducirse sólo al esquema y no requieren esfuerzo de programación.

2.-Existe mejor control sobre el contenido de la base de datos y el acceso a ella.

3.-Es posible realizar la programación antes de que se conozcan las especificaciones exactas de los datos.

4.-Debido a que se dispone de más información de proceso, es más fácil mantener la sincronización de los accesos a un archivo compartido.

5.-Existe mayor independencia entre procesos y archivos., lo que protegerá al sistema de los cambios de hardware y software.

6.-Las proposiciones en lenguaje fuente tal vez requieran menos espacio de almacenamiento que las instrucciones necesarias para acceso a archivo.

Las desventajas del método interpretativo se listan a continuación:

1.-El intérprete general tiene menos flexibilidad cuando es necesario tomar en cuenta consideraciones especiales.

2.-Al interpretar existe un alto costo de procesador central.

3.-Es difícil construír y depurar buenos programas generales.

4.-El programa único generalizado de interpretación puede ocupar más espacio que los programas separados de procesamiento que se llaman eclusivamente de acuerdo con la demanda.

Los detalles del sistema aprendidos en la planeación y análisis se unen de manera que se pueda estudiar y evaluar el sistema actual. Esta etapa se centra en la eficiencia con que se efectúan ciertos pasos y cómo contribuyen al resultado deseado.

Se debe determinar si se cometen errores u omisiones excesivas, con qué frecuencia suceden, por qué y cuál es el costo de ésto.

La evaluación del sistema produce detalles que describen las operaciones actuales y señalan las áreas en donde se necesitan mejoras o dónde son posibles.

6.3.- MAPEO DEL MODELO DE DATOS.

El principal método para el diseño de bases de datos es la construcción de modelos que representan la estructura de dicha base en forma tal que permita la manipulación de los bloques de construcción para la base de datos.

Un modelo es la herramienta más importante para el diseñador de bases de datos.

La herramienta empleada para describir bases de datos es el *modelo estructural*, el cual se basa en el modelo relacional; en donde las relaciones entre archivos se denominan *conexiones*. Se emplean tres tipos de conexiones para clasificar las relaciones en cinco tipos (las cuales se estudiarán más adelante).

Cuando se han recolectado todas las definiciones para la base de datos, se tiene un modelo estructural o un *modelo de la base de datos*.

En las empresas modernas las computadoras realizan muchas funciones, por lo tanto, están en uso muchos archivos y existen también muchas relaciones entre éstos archivos. Cuando se habla de una estructura para la base de datos se consideran todas las relaciones entre todos los archivos.

Una base de datos que se utilice para dos fines diferentes puede tener dos modelos distintos según la visión de los usuarios. el manejo simultáneo de varias visiones puede casi imposibilitar la construcción de un modelo válido de base de datos.

Mientras más usos tenga una base de datos, más conflictos se presentarán entre las distintas visiones. Por lo tanto, primero se definirá la base de datos desde un sólo punto de vista y posteriormente se definirá desde otros puntos de vista. El modelo estructural proporciona el medio para *integrar* dos o más visiones diferentes.

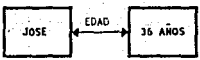
6.3.1. MODELOS DE VISION

Un modelo de visión representa un pequeño subconjunto de la realidad, para una aplicación del contenido de la base de datos. La mayoría de las bases de datos para especificarse, requerirán de varios modelos de visión.

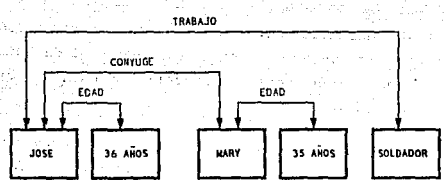
La mayoría de los conceptos de planteamiento de modelos se aplica tanto a modelos de bases de datos como a modelos de visión. En muchos textos de computación el término de *modelo de bases de datos* a menudo se aplica a lo que en realidad es un modelo de visión ya que no se ocupan directamente del problema de múltiples visiones.

Un modelo de visión se construye a partir de los elemento dato y de sus vínculos. Los elementos dato representan valores de atributos de entidades. La expresión básica de un vínculo es el vínculo binario.

Se encuentran vínculos múltiples cuando los datos se recolectan y organizan para su procesamiento. Los pasos de la figura 6.7 proporcionan ejemplos de los bloques iniciales de construcción de los modelos de visión.



a)



b)

NOMBRE	EDAD	CONYUGE	EDAD DEL CONYUGE	TRABAJO
JOSE	36	MARY	35	SOLDADOR

c)

PERSONA:

NOMBRE	EDAD	ESPOSA	EDAD_CONYUGE	TRABAJO
JOSE	36	MARY	35	SOLDADOR
PEDRO	28	PATY	32	ESCRITOR
ANDRES	30	NONE	N.A.	SOLDADOR
MARY	35	JOSE	36	PROGRAMADOR

PARTE RECTORA

PARTE DEPENDIENTE

d)

FIG.6.7.- VINCULOS, ENEADAS Y RELACIONES.

a)VINCULO BINARIO. b)VINCULOS VINARIOS MULTIPLES
c)ENEADA. d)RELACION.

Una *eneada* representa múltiples atributos de un objeto. Conjuntos de eneadas semejantes se ensamblan en *relaciones*. Cada eneada en una relación representa algún objeto diferente y expresa vínculos semejantes entre sus atributos.

Los valores para los atributos es el *dominio*. El *atributo* es el nombre que sirve de etiqueta a las columnas de la eneada. Ahora una columna de una relación contiene elementos dato semejantes. La *parte rectora* está formada por la columna o columnas de atributos que definen al objeto. Los atributos restantes son la *parte dependiente*.

En la figura 6.8 se compara la terminología de los cuatro niveles de abstracción que se encuentran en el diseño de archivos y bases de datos.

Con cada atributo se define un *dominio* específico a partir del cual es posible seleccionar valores. El número de valores en un dominio puede ser muy grande o bastante limitado. Por ejemplo :

-El dominio de un atributo para la cantidad de una factura puede variar de millones de dolares a un crédito igualmente grande.

-El dominio de un atributo temperatura que describa a personas puede ser {97.0,97.1,...,105.0}.

Para construír un modelo es necesario definir la estructura interna de las relaciones y los vínculos entre las relaciones.

HARDWARE	SOFTWARE	MODELO	SEMANTICA
DISCO	ARCHIVO	RELACION	CONJUNTO ENTIDAD
DOCUMENTACION	ESQUEMA	RELACION-ESQUEMA	DESCRIPCION
BLOQUE	REGISTRO	ENEADA	OBJETO
DIRECCION	LLAVE	PARTE RECTORA	NOMBRE OBJETO
CONTENIDO	OBJETIVO	PARTE DEPENDIENTE	ATRIBUTOS
NUMERO DE BYTE	NOMBRE DEL ATRIBUTO	ATRIBUTO	ATRIBUTO
FORMATO DE DATOS	TIPO DE DATO	DOMINIO	INTERVALO DE VALORES
BYTE O PALABRA	VALOR DEL ATRIBUTO	ELEMENTO DATO	VALOR

FIG.6.8.- TABLA DE TERMINOLOGIA.

6.3.2. *BLOQUES DE CONSTRUCCION DE MODELOS*

Anteriormente se mencionó que un modelo de base de datos puede describirse en términos de 5 tipos relativamente simples de relación y sus subrelaciones :

1.-Relaciones de entidad
2.-Relaciones anidadas
3.-Léxico
4.-Relaciones de entidad referidas
5.-Relaciones asociativas

y tres tipos de conexión :

1.-de propiedad
2.-de referencia
3.-de subconjunto

Un tipo de relación define las conexiones en que participa, y un tipo de conexión define los tipos de relación que conecta. Ambos conceptos son útiles al manejar modelos y su implantación, y ambos se encuentran en sistemas de bases de datos.

Relaciones de Entidad.

A una relación que define un conjunto de elementos o entidades independientes se le denomina *relación de entidad*. La elección de tipos de entidad es un aspecto fundamental del diseño de un modelo de visión.

Las entidades son comúnmente elementos que pueden tocarse, contarse, moverse, adquirirse o venderse, y que no pierden su identidad durante tales manipulaciones. Los atributos pueden ser valores básicos o referencias a eneadas en otras relaciones,como se muestra en el caso del empleado de la figura 6.9.

Relaciones Anidadas.

Durante la normalización se trasladan los nidos o grupos de atributos a relaciones separadas a las que se denomina *relaciones anidadas*.

Las eneadas de relaciones anidadas tienen una parte rectora que es la composición de la parte rectora de la relación propietaria y un valor atributo que es único dentro del nido. En la figura 6.10 se muestra una relación anidada.

Para continuar existiendo las eneadas de estas relaciones anidadas, dependen de las eneadas propietarias. Esto significa que es posible insertar eneadas en una relación anidada sólo si existe una eneada propietaria coincidente y que las eneadas en una relación anidada tienen que quitarse cuando se quita la eneada propietaria.

La parte dependiente de una relación anidada está regida adecuadamente por la concatenación de la parte rectora de los padres y el atributo rector de eneada espicificada, pór ejemplo, en la figura 6.11 : *padre, hijo*.

Las relaciones anidadas pueden ser los padres de otras relaciones anidadas, como se muestra en la figura 6.11. Pueden existir muchos niveles de anidamiento. Además, los modelos también pueden requerir relaciones múltiples anidadas en el mismo nivel.

Empleado : RELATION

nombre :>	fecha_de_nacimiento,	estatura,	peso,	trabajo,	num_dep,	num_salud;
AGUSTIN	1938	1.70	80	SOLDADOR	38	854
GABRIEL	1947	1.82	83	ASIST_SOLDADOR	36	2650
MANUEL	1921	1.65	70	GERENTE	38	12
OSCAR	1956	1.75	78	EN ENTRENAMIENTO	38	1366
PABLO	1938	1.69	76	SOLDADOR	32	855

FIG.6.9.- RELACION DE ENTIDAD.

Hijos : RELATION

padre,	hijo :>	edad_h;
AGUSTIN	ISAAC	5
MANUEL	EDUARDO	17
MANUEL	RAFAEL	19
SERGIO	ISRAEL	1

FIG.6.10.- RELACION ANIDADA.

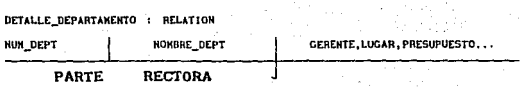
Educacion : RELATION

padre,	hijo,	tipo_de_escuela	:> nombre_escuela,	tema;
MANUEL	EDUARDO	BACHILLERATO	C.C.H.	CIENCIAS
MANUEL	RAFAEL	BACHILLERATO	C.C.H.	CIENCIAS SOCIALES
MANUEL	RAFAEL	UNIVERSIDAD	ENEP ARAGON	CIENCIAS POLITICAS
...	...	...		
...				

FIG.6.11.- RELACION ANIDADA DE SEGUNDO NIVEL.

Léxicos.

En cualquier relación las partes rectoras y dependientes deberán estar identificadas con claridad. A menudo se encuentran vínculos que tienen más de una *posible parte rectora*, por ejemplo :



Cuando existen varias partes rectoras se eliminan las que sean redundantes de una relación R a un tipo muy específico de relación, el *léxico* L.

Un léxico define una correspondencia uno a uno entre dos conjuntos A y B mediante la relación binaria L : A $\Leftrightarrow$ B. En este caso, la parte dependiente lo hace en forma funcional de la parte rectora y la parte rectora depende funcionalmente de la parte dependiente. Puede decirse que un léxico implanta equivalencia.

Cada eneada de un léxico deberá hacer referencia a una eneada correspondiente en la relación de entidad y viceversa. En el modelo de visión se observa esta dependencia estructural utilizando *conexiones de referencia* entre relaciones.

En la figura 6.12 se muestra una relación de léxico, dando una lista de nombres de departamento por números de departamento.

Departamentos : RELATION

num_depto (:)	nombre_depto;
23	CONTABILIDAD
27	AUDITORIA
31	FUNDICION
32	FORJA
34	ESTAMPADO
36	ACABADO
38	ENSAMBLE
50	PRUEBAS
33	CONTROL DE CALIDAD
24	VENTAS INTERNAS
25	VENTAS EXTERNAS

FIG.6.12.- LEXICO.

Descripcion_del_trabajo : RELATION

trabajo	:) Educacion_necesaria,	Experiencia_necesaria;
ASISTENTE DE SOLDADOR	SECUNDARIA	2 ANOS
GERENTE	UNIVERSIDAD	12 ANOS
EN ENTRENAMIENTO	SECUNDARIA	NINGUNA
SOLDADOR	SECUNDARIA	8 ANOS

FIG.6.13.- RELACION DE ENTIDAD PARA REFERENCIA

Los léxicos se presentan con frecuencia en bases de datos operativas. Los léxicos pueden tratarse desde el punto de vista conceptual, como si fueran un solo atributo al diseñar un modelo de bases de datos.

Relaciones de Entidad Referidas.

Es posible establecer nuevas relaciones de *entidad referenciada*. Este tipo de relación obedece a todas las reglas establecidas para las relaciones. Ya que es un conjunto, se elimininarán de él las eneadas redundantes y el número de éstas será igual o menor (a menudo mucho menor) que el número de eneadas en la relación primaria a que se hizo referencia. El atributo de la referencia aparecerá en ambas relaciones y se volverá la parte rectora de la relación referida.

La especificación del trabajo de un empleado puede contener muchos atributos subsidiarios. Entre éstos se encontrarán valores que son funcionalmente dependientes del nombre del trabajo. Para eliminar esta redundancia se creará una nueva relación de entidad referenciada que describa las propiedades del trabajo, como se muestra en la figura 6.13.

Un atributo único de referencia en la relación de entidad de empleado hará referencia en la relación de trabajo utilizando la parte rectora de las eneadas de trabajo.

Relaciones Asociativas.

Una *relación asociativa* tiene una parte rectora compuesta de dos o más atributos. Los atributos relacionan cada eneada de la relación asociativa con eneadas en dos o más relaciones propietarias.

Los atributos de la parte rectora que sean redundantes se eliminarán durante la normalización. El número de propietarias es arbitrario, aun cuando durante la normalización a menudo se encuentran atributos dependientes de las *asociaciones poseídas por dos relaciones*.

Un ejemplo clásico de relación asociativa es el vínculo entre las partes empleadas por una compañía manufacturera y los proveedores de dichas partes, como lo muestra la figura 6.14.

Es posible que las relaciones asociativas no tengan parte dependiente, por ejemplo para una relación que describa la capacidad de un Proveedor específico para entregar ciertas Partes.

Conexiones de Propiedad.

Se crean conexiones de propiedad al describir la dependencia de las relaciones anidadas con respecto a las relaciones propietario y al describir las asociaciones entre relaciones.

Se utilizan para describir un tipo específico de *dependencia multivaluada* (cuando un valor atributo determina un conjunto de valores múltiples); es decir, el caso en que los atributos dependientes de cada eneada forman conjuntos o relaciones semejantes en vez de conjuntos arbitrarios. A estos conjuntos se les denomina conjuntos poseídos. La relación poseída tendrá un conjunto poseído por cada eneada de la relación propietaria.

Proveedores : RELATION

Pro_id :>	nombre,	etc. ;
S1	ACME SCREW CO.	...
S2	BAY BOLT CO.	
S3	CONSOLIDATED NUTS	
S4	OZUS FASTENERS	
S5	ILIUM WORKS	

Partes : RELATION

par_id :>	nombre,	tamano,	peso,	etc.;
P1	PERNO DE MAQUINA	11 X 4	0.31	...
P2	TUERCA	4	0.07	
P3	RONDANA DE SEGURO	4	0.04	
P4	RONDANA	4	0.02	

Suministro : RELATION

pro_id,	par_id :>	cantidad ;
S1	P1	160
S1	P3	60
S2	P2	140
S2	P3	50
S2	P4	90
S4	P4	100

FIG.6.14.-RELACION ASOCIATIVA CON SUS 2 PROPIETARIOS.

REGLAS DE PROPIEDAD.-

LA PARTE RECTORA DE LA RELACION POSEIDA ES LA CONCATENACION DE LA PARTE RECTORA DE LA RELACION PROPIETARIA Y UN ATRIBUTO PARA DISTINGUIR A LOS INDIVIDUOS EN LOS CONJUNTOS POSEIDOS. ES POSIBLE INSERTAR UNA NUEVA ENEADA EN LA RELACION POSEIDA SOLO SI EXISTE UNA ENEADA PROPIETARIA QUE COINCIDA Y QUE PERTENEZCA A LA RELACION PROPIETARIA. LA ELIMINACION DE UNA ENEADA PROPIETARIA IMPLICA LA ELIMINACION DE SU CONJUNTO POSEIDO.

Conexiones de Referencia.

Se utilizará el término *relaciones primarias* para todas las relaciones que efectúan referencias.

En una relación referenciada se espera encontrar eneadas que coincidan con cualquier valor tomado por el atributo que efectuó la referencia, la ausencia de una eneada referenciada constituye un error. Por lo tanto, una relación referenciada define el dominio para el atributo que efectúa la referencia. En la figura 6.16 se muestran alternativas para relizar referencias.

REGLAS DE REFERENCIA.-

LA PARTE RECTORA DE UNA RELACION REFERIDA COINCIDE CON EL ATRIBUTO AL QUE SE HIZO REFERENCIA EN LA RELACION PRIMARIA, O QUE REALIZA LA REFERENCIA. LAS ENEADAS EN RELACIONES REFERENCIADAS NO PUEDEN ELIMINARSE MIENTRAS EXISTA CUALQUIER REFERENCIA. LA ELIMINACION DE ENEADAS QUE REALIZAN REFERENCIAS, DE LA RELACION PRIMARIA, NO IMPLICA LA ELIMINACION DE LA CORRESPONDIENTE ENEADA REFERIDA.

Si no se cumplen estas reglas es posible que la información vital no esté disponible durante el procesamiento, provocando la falla de transacciones.

TIPOS DE CONEXION EN EL MODELO ESTRUCTURAL		
CONEXION DE PROPIEDAD	——*	DE ENEADAS DE PROPIETARIO UNICO A MULTIPLES PROPIETARIOS.
CONEXION DE REFERENCIA	——→	DE ENEADAS DE REFERENCIA MULTIPLE A UNICA.
CONEXION DE SUBCONJUNTO	——)	DE ENEADAS GENERALES UNICAS A ENEADAS DE SUBCONJUNTO UNICO.

FIG.6.15.- TIPOS DE CONEXION.

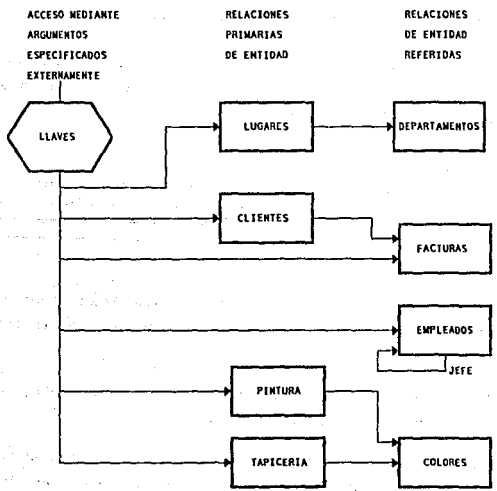


FIG.6.16.- RELACIONES REFERIDAS.

Una conexión de referencia se basa en un atributo de la relación primaria. Los valores dato que se estén introduciendo para atributo deberán estar restringuidos por el dominio definido por las eneadas de la relación referida.

Conexiones entre Subconjuntos.

Es necesaria una conexión entre subconjuntos cuando se encuentran relaciones con partes rectoras idénticas desde el punto de vista formal, pero que difieren en sus atributos o dominios.

Las relaciones de subconjunto ocurren con frecuencia. Es posible que se desee recolectar atributos para los gerentes o personal de ventas que otros empleados no necesiten.

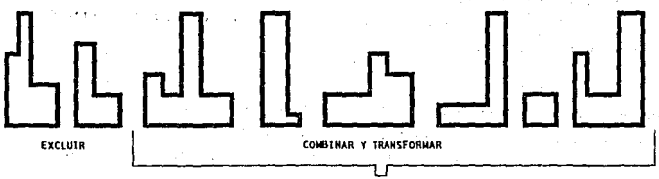
REGLAS DE LOS SUBCONJUNTOS.-

LA PARTE RECTORA DE UNA SUBRELACION COINCIDE CON LA PARTE RECTORA DE SU RELACION GENERAL CONECTADA. CADA ENEADA DE SUBCONJUNTO DEPENDE DE UNA ENEADA GENERAL. UNA ENEADA GENERAL PUEDE O NO TENER UNA ENEADA EN CUALQUIER RELACION CONECTADA DE SUBCONJUNTO.

Hasta ahora se han considerado una por una las relaciones y sus conexiones. El modelo de visión tendrá muchas relaciones y conexiones.

Cualquier relación puede participar en múltiples conexiones, por lo que tendrá múltiples reglas de restricción.

Si un modelo de visión evita la redundancia, los únicos atributos que aparecerán en más de una relación serán los que definan conexiones.



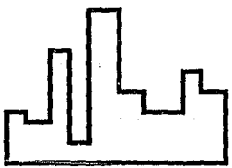


FIG.6.17.- CONSTRUCCION DEL MODELO DE BASES DE DATOS.

Cuando se aprueba la visión, resulta factible el procesamiento automático de la información en los modelos de visión para llegar a un modelo de base de datos.

Cuando se ha establecido un conjunto comprensivo de modelos de visión es posible establecer la construcción de un modelo para toda la base de datos. Se combinan relaciones provenientes de modelos separados de visión con base a los atributos que tengan en común.

Si los modelos de visión no tienen atributos en común, no se obtiene ningún beneficio al unir estos datos en un solo modelo de base de datos. En la figura 6.17 se presenta gráficamente el concepto.

Aunque haya atributos comunes podría no haber conexiones. La falta de conexiones indica que las visiones pueden mantenerse independientemente unas de otras.

6.4.- MEDIO AMBIENTE

En un medio ambiente en línea es de suma importancia tener presente todos los problemas que pueden afectar la base de datos. Si se desea asegurar una base de datos es necesario :

-Proporcionar un mecanismo de protección para lograr el control deseado de los accesos a los datos.

-Asegurarse de que no se presentará interferencia destructiva cuando múltiples usuarios compartan el acceso a la base de datos.

-Lograr un modo de operación confiable y predecible para que en un momento determinado se pueda detectar fallas y arreglarlas.

Estos tres puntos se estudiarán en este tema como :

Seguridad.-

Comprender, organizar y controlar el acceso a los datos de acuerdo con derechos específicos.

Concurrencia.-

Conservar la seguridad en un sistema en que se permite a múltiples usuarios el acceso al sistema y compartir la base de datos.

Recuperación de caídas anormales.-

Tener la posibilidad y los medios para corregir fallas.

6.4.1. SEGURIDAD DE LA INFORMACION

Uno de los riesgos y problemas para los diseñadores de la base de datos es la seguridad de la información, ya que personas conocedoras (o intrusos hábiles) pueden robar datos o programas y venderlos, o alterar las transacciones en los datos con el fin de cometer un fraude. Esto lo pueden hacer donde se encuentra la computadora o en una terminal lejana. Por lo tanto, deben adoptarse medidas para proteger el sistema contra el robo.

Un sistema de base de datos es vulnerable al ataque y penetración de diversos orígenes si no se establecen los controles adecuados. Se deben diseñar los controles de tal forma que cada vez hagan más difíciles el ataque y la penetración.

Un aspecto importante es la seguridad física, es decir, la protección del hardware y el software contra daños o destrucción ocasionados por incendio, inundación o sabotaje. Por lo tanto, deben aplicarse las medidas necesarias para almacenar programas y datos importantes en otro lugar, a manera de respaldo, y también para proteger de éstos desastres el sitio donde se encuentra el hardware.

Un paso importante para lograr más seguridad es limitar el acceso al sistema; así como prestar atención especial a los procedimientos de control para identificar a los usuarios autorizados del sistema. Dicha identificación puede ser una palabra clave, una tarjeta con un número de identificación codificado en forma magnética, sus huellas digitales o su tono de voz o bien alguna combinación de todos estos elementos. Las palabras claves son las que más se utilizan, pero cuando se emplean con frecuencia pierden su valor como elemento de seguridad.

El término seguridad se emplea para describir la protección contra la destrucción de los sistemas y su contenido. No se abarcarán aquellos aspectos de la seguridad que se refieren a edificios y seguridad del personal, aun cuando resulta obvio que sin seguridad física la mayoría de los esfuerzos orientados al sistema de computación no llegarán a ninguna parte.

A continuación se definirán algunos términos a fin de que el análisis subsecuente de los mecanismos para lograr protección resulte claro.

Accesores.-

Los usuarios con acceso a la base de datos.

Entorno.-

El área con un perímetro bien definido, conocido como sistema de la base de datos.

Intrusos.-

Individuos disfrazados de usuarios válidos.

Alcance Limitado.-

El sistema desconoce la identidad de los individuos en el mundo exterior.

Privilegios.-

Existen varios privilegios de acceso a los datos, relacionados con la identificación de un individuo. La descripción de estos privilegios se mantiene como parte del sistema de la base de datos.

Protección.-

Todos los elementos dato están protegidos hasta cierto punto mientras se encuentren dentro del área del sistema de la base de datos, y perderán toda la protección que proporciona el sistema al sacarse del área.

Confiabilidad.-

Un prerequisito para lograr la protección de la base de datos es un alto nivel de confiabilidad del sistema.

La protección de los datos requiere ejercer control sobre la lectura, escritura y empleo de esa información. La protección de los datos siempre es limitada.

Mientras más protección se implante para reducir las violaciones accidentales y deliberadas de acceso, mayor será el costo del sistema. Cuando el costo de la protección excede el valor de los elementos dato protegidos, se ha llegado a un limite.

El valor de la protección de un elemento dato puede determinarse de acuerdo al beneficio obtenido por un intruso deliberado o de acuerdo a la pérdida sufrida por el propietario en caso de una exposición accidental.

El método empleado para autenticar la identificación de un individuo depende mucho de la tecnología disponible en un caso específico.

Todas las autorizaciones y privilegios dados a los usuarios con acceso a la base de datos dependerán de la *llave de acceso (password)*. Para asegurarse de que las llaves de acceso no estén a disposición de accesores no autorizados, el password debe ser muy difícil de imitar o copiar, por ejemplo, el nombre de un individuo no es una llave adecuada ya que es fácil que cualquiera que observe a quienes tienen acceso al sistema lo copie.

A fin de proteger el proceso de obtención de una llave del sistema, cuando un usuario realiza la entrada (LOGIN) solicita una clave de acceso con el nombre del usuario. La clave de acceso se introduce sin exhibirla a fin de protegerse de los observadores.

Un intruso podría utilizar un método de ensayo y error para introducir posibles claves de acceso y lograr entrar, sin embargo, el tiempo necesario para realizar este método es el principal elemento para desanimar a posibles intrusos.

El tiempo esperado para abrir un seguro específico sin ningún conocimiento previo es :

$$T_{(entrar)} = \frac{1}{2} C^{d} t_{(ensayo)}$$

en donde :

C^{d} es el número de posibles combinaciones.

$t_{(ensayo)}$ es el tiempo necesario para ensayar o probar una combinación.

Por ejemplo, para una clave de acceso de 3 letras, d = 3 y C = 26, el tiempo para la interacción con el sistema podría ser $t_{(ensayo)}$ = 3 segundos; de manera que :

$$T_{(entrar)} = \frac{1}{2} 26^{3}(3 \text{ seg}) = 26,364 \text{ seg.} = 7.3 \text{ horas}$$

Un método para complicar un ensayo de este tipo consiste en permitir sólo dos o tres intentos de introducir una clave de acceso. En este caso, el intruso tendría que recorrer de nuevo el procedimiento completo de acceso, lo cual probablemente le tomará 20 segundos y entonces :

$$T_{(entrar)} = \frac{1}{2} 26^{3}(20 \text{ seg}) = 26,364 \text{ seg.} = 48.8 \text{ horas}$$

Otra alternativa para aumentar la seguridad consiste en ampliar el número de combinaciones. Un problema práctico es que las claves de acceso largas son más difíciles de recordar; otro problema es que con frecuencia se eligen como claves de acceso iniciales, nombres, diminutivos, nombres de amigos u otros auxilios mnemónicos, por lo que resultan fáciles de adivinar.

La responsabilidad del manejo de la llave corresponde tanto al accesor como al sistema operativo.

Otro método para aumentar la protección consiste en colocar en el sistema una rutina de interrogación que solicita parámetros al accesor durante el proceso de autenticación. Un ejemplo sería, presentar un número al accesor, quien deberá multiplicarlo por 2 y sumar 4; este procedimiento es la 'clave de acceso' que el usuario debe recordar.

La existencia de intrusos puede determinarse posteriormente mediante un informe de todos los intentos de acceso en un período dado al propietario de un archivo.

Hasta este momento, se ha caracterizado al accesor como un individuo, pero algunas veces la autoridad de acceso se refiere a clases de individuos; por ejemplo, el personal de oficina podría no estar identificado en forma individual. La autenticación por clase no es muy segura, ya que no existe responsabilidad individual.

Pueden formarse categorías de acceso a los datos por tipo. Por lo común, las distinciones se realizan entre autorización para leer y autorización para escribir los datos. Algunas unidades de disco tienen interruptores de protección de escritura. Muchos sistemas operativos compartidos tienen agregado un privilegio de ejecución exclusivamente.

Es posible distinguir los siguientes tipos de objetos o elementos que se proporcionarán o se restringirán : datos, rutas de acceso, programas de la base de datos y anotaciones de esquemas.

Es necesario considerar cuidadosamente cuál será la acción a realizar cuando se detecte una violación del mecanismo de protección. Gran número de violaciones de acceso se debe a depuraciones que realizan los programadores o a oficinistas que se equivoquen al teclear una consulta.

Por otra parte, deberá llevarse una bitácora adecuada del comportamiento incorrecto en el acceso, para permitir detectar las acciones ilícitas y atrapar en su caso a quienes las realicen.

Los desarrolladores actuales en los sistemas operativos están tomando cada vez más en cuenta la existencia y necesidades de las bases de datos. Sigue siendo importante que el diseñador de la base de datos revise las facilidades disponibles y entienda sus debilidades. A continuación se listan 10 principios que deben considerarse en el análisis de sistemas de protección.

1.-Economía de Funcionamiento.

Un sistema excesivamente complicado de protección no permite verificar si es correcto y completo.

2.-Omisiones de Falla de Seguridad.

Si algo sale mal, el acceso deberá negarse.

3.-Mediación Total.

Cada ruta de acceso a la base de datos deberá ser a través del mecanismo de protección.

4.-Diseño Abierto.

La protección no aumenta al conservar secreto el mecanismo de protección.

5.-Separación de Privilegios.

Los datos críticos deberán protegerse, de manera que sea imposible que un solo usuario autorizado burle el mecanismo.

6.-Menor Privilegio.

A un programa se le deberá otorgar sólo el privilegio esencial de acceso.

7.-Mecanismo menos común.

Se debe evitar compartir en forma innecesaria. En las bases de datos, compartir es el mayor beneficio pero, evítese conservar en una base de datos aquellos datos que no se puedan compartir.

8.-Aceptabilidad Psicológica.

Los mecanismos deben coincidir con el modelo de datos del usuario y con la protección que necesitan. No hacer esto crea errores en el uso y propicia intentos de evadir el sistema.

9.-Factor de Trabajo.

El esfuerzo para burlar el mecanismo deberá ser mayor que el beneficio que pudiera obtenerse.

10.-Compromiso de llevar registros.

El llevar una bitácora de los accesos puede crear la ruta de auditoría necesaria para corregir problemas y atrapar a los ilegales.

Cuando se determina que los sistemas son inadecuados, el diseñador de la base de datos puede aumentar las facilidades, pero también deberá proporcionar a los usuarios una evaluación realista del nivel disponible de protección.

Existe una técnica alternativa de protección llamada CRIPTOGRAFIA, la cual consiste en transformar los datos en una forma que no proporcione información al ser interceptada; por ejemplo, hacer que el valor "1" signifique "masculino" y el valor "2" indique "femenino" es una forma de proteger este campo de alguien que no tenga acceso al libro de código o al esquema.

Las técnicas criptográficas proporcionan sólo un aspecto de la protección necesaria de acceso. La protección de la destrucción, intencional o accidental, no se obtiene mediante ésta técnica.

6.4.2. C O N C U R R E N C I A

Existe un área en la que es posible que se generen fallas, aunque todo lo demás aparentemente funcione en forma satisfactoria. Esta área problemática existe debido a la interferencia de múltiples transacciones que están activas dentro de un sistema de computación.

La *integridad* de una base de datos indica la ausencia de datos inconsistentes. Mientras se está realizando una transacción de actualización, la consistencia de una base de datos puede estar temporalmente alterada.

Un error en una sola transacción puede producir una falla si una actualización de la base de datos no se realiza hasta concluírla, sin embargo, aun las transacciones de actualización libres de errores pueden provocar problemas cuando realizan acceso a datos compartidos en un lapso en el que exista otra actividad. Los accesores que compartan los datos pueden recoger inconsistencias temporales provocadas por una actualización concurrente.

Debe recordarse que una transacción es un cálculo que inicia un usuario, que se permite sea procesado tan rápidamente como el hardware disponible lo permita y después se concluye.

La tarea principal de un sistema operativo orientado a las transacciones consiste en conservar, hasta donde sea posible, una transacción sin que se vea afectada por la presencia de otras. Esta independencia no puede lograrse por completo, habrá competencia para ocupar los procesadores (especialmente si solo se dispone de uno), para el empleo de los discos y de los canales de acceso a ellos, y para el uso de los dispositivos de entrada y salida.

En una operación de base de datos existen recursos adicionales que deben compartirse. Todos los usuarios desearán tener acceso al esquema; otros tal vez seleccionen un archivo en particular, o un índice o un nivel jerárquico, y posiblemente varios deseen tener acceso al mismo elemento dato.
El empleo de seguros proporciona los medios para impedir que las transacciones interfieran unas con otras.

Cuando más de una transacción obtiene alguna copia de algún elemento dato para actualizarlo, el valor final de la base de datos para ese elemento será el resultado de la última transacción que escribió el valor actualizado. El resultado de la otra transacción se ha escrito de más.
La solución a los problemas de concurrencia consiste en proporcionar un mecanismo de *seguro* que asegure la exclusión mutua de transacciones interferentes. La transacción que solicite primero el objeto no compartible se vuelve temporalmente la propietaria de dicho objeto. Otras transacciones que deseen lograr acceso al objeto se bloquean hasta que el propietario libere el objeto.

El derecho a un objeto se realiza colocando un *semáforo*. Un semáforo binario simple consiste en una variable X protegida, que indique si el recurso asociado está desocupado u ocupado. Puede colocarse un semáforo que indique ocupado sólo si anteriormente indicaba que el recurso estaba desocupado.

Se considera que cualquier computadora tiene algún tipo de instrucción que pueda realizar la función de semáforo. Algunas versiones permiten posicionar X mediante un código que identifica al propietario actual, otras versiones pueden contar el número de solicitudes que se están deteniendo.

A la elección del tamaño de los objetos que se van a asegurar se le denomina *granularidad* del seguro. La menor unidad que puede identificarse para asegurarse mediante un esquema es un campo o un segmento; para acceso de archivo es un registro y la menor unidad disponible para manejo de buffer comúnmente es un bloque. Muchos sistemas operativos proporcionan seguros sólo para archivos completos.

El empleo de seguros también tiene una dimensión temporal. El intervalo de seguros puede acortarse liberando los seguros de un objeto en cuanto se realice la actualización. Para obtener un alto nivel de uso en un sistema de computación, resulta conveniente excluír a los otros usuarios por un tiempo tan corto como sea posible.

No asegurar la transacción de inmediato significa que existe cierta posibilidad de que otra transacción haya modificado los datos de manera que sea necesario volver a leer todos los datos que contribuyan al resultado. Cuando la transacción esté lista, todos los datos se vuelven a leer con seguros. El tiempo que los otros usuarios se ven excluídos se reduce, pero es necesario leer dos veces cada registro. Este modo de operación también tiene implicaciones para evitar caer en punto muerto, como se verá más adelante.

Se han mencionado problemas de interferencia debidos a actualización concurrente de un sólo objeto atómico; pero también es posible que otros problemas de interferencia necesiten asegurar colecciones de objetos o *regiones*. Si se incluyen muchos objetos o si estos son muy grandes en la región que se va a asegurar para una transacción, la probabilidad de que otras transacciones se vean afectadas aumenta rápidamente. La técnica de conservar los objetos hasta que la actualización se ejecute falla cuando es necesario considerar regiones. El programador debe definir las regiones de la transacción, o tendrán que retenerse todos los recursos a los que se realizó el acceso hasta que se concluya la transacción.

No existiría ninguna interferencia si cada transacción esperará hasta que su predecesora concluyera, sin embargo, el desempeño del sistema se vería muy disminuído. Se desea permitir el traslape de las transacciones que sigan dando respuestas correctas y dejen a la base de datos en un estado correcto. A la propiedad de las transacciones que se traslapan de tener una programación cronológica correspondiente en serie se le denomina *serializabilidad*.

El empleo de seguros tiene un fuerte efecto en el desempeño de la base de datos, ya que los seguros colocados por una transacción hacen que la base de datos no esté disponible para muchas otras transacciones. El grado en el que el desempeño se ve afectado depende del tamaño del objeto o región que esté asegurado y de la longitud del intervalo del seguro.

Puede evitarse el empleo de seguros durante el procesamiento diario, si las modificaciones de archivo que se realicen a causa de actualizaciones se conservan en un archivo bitácora de actualizaciones. El archivo bitácora de actualizaciones se une a la base de datos durante periodos de poca o ninguna actividad simultánea, normalmente durante la noche.

La interferencia en las rutas de acceso puede provocar que los programas de transacción fallen. Trasladar un objeto puede provocar que una transacción intente localizar este objeto mediante un apuntador sólo para tomar datos no válidos. Estos problemas pueden minimizarse o evitarse por completo al diseñar cuidadosamente los procedimientos que manipulan las rutas. Para evitar localizar y recuperar datos incorrectos se coloca un *epitafio* en el punto anterior.

Ya que las bases de datos compartidas requieren muchos seguros, el manejo de estos seguros con frecuencia se vuelve responsabilidad del sistema de manejo de la base de datos.

Es claro que múltiples transacciones se mueven en una base de datos, reclamando la propiedad de objetos y regiones a fin de bloquear a otras para que no interfieran con las actividades que pretende llevar a cabo, por lo tanto, existe una clara posibilidad de problemas, los cuales pueden clasificarse en dos clases :

Hibernación.-

La transacción se encuentra en hibernación cuando está bloqueada o dormida y el sistema está demasiado ocupado para despertarla dentro de un tiempo razonable.

Punto Muerto.-

Las transacciones están mutuamente en punto muerto cuando una transacción está bloqueada por otra y ésta a su vez está bloqueada por la primera. El círculo puede involucrar a muchas transacciones y ninguna de ellas puede ser activada sin posibles violaciones a la integridad.

La diferencia entre hibernación y punto muerto no es visible para el usuario, ya que simplemente : el sistema no trabaja de nuevo. En cambio, para un diseñador de sistemas la diferencia es importante :

-El punto muerto puede entenderse y resolverse o su ocurrencia puede reducirse a niveles aceptables.

-Los problemas de hibernación pueden deberse a actividades únicas y en el momento que se realiza un análisis todas las transacciones están activas y no queda evidencia.

La hibernación ocurre cuando una transacción H no recibe recursos durante un periodo excesivamente largo; puede deberse a que otra transacción contenga un recurso reclamado por H y no se concluya, porque H tiene una prioridad cronológica de programación menor. Por lo tanto, tener transacciones con distintas prioridades provoca que las transacciones que tengan baja prioridad entren en hibernación.

Los procesos que tienen fuertes demandas de archivo pueden tener asignada una alta prioridad a la CPU de manera que realicen un avance aceptable. Sin embargo, un número excesivo de tales procesos puede limitar demasiado a los otros usuarios.

Cuando no es posible evitar estas prioridades, se puede aumentar en cierta cantidad la prioridad del proceso periodicamente, de manera que eventualmente el proceso retrasado tenga una prioridad adecuada para proceder.

Los puntos muertos ocurren cuando dos o más transacciones solicitan recursos en forma incremental y se bloquean mutuamente impidiéndose una a otra la conclusión. La frecuencia de los puntos muertos dependerá de la interacción de las transacciones. A continuación se muestra una lista de 4 condiciones que permiten que ocurran los puntos muertos.

1.-Seguros.
La interferencia de acceso se resuelve posicionando y respetando los seguros.

2.-Bloqueo.
Un propietario de un objeto está bloqueado cuando solicita un objeto asegurado.

3.-Garantía de Conclusión.
Los objetos no pueden quitarse de sus propietarios.

4.-Circularidad.
Existe una secuencia circular de solicitud.

Ahora, se muestra una lista de cuatro enfoques para evitar los puntos muertos.

1.-Reparación posterior.
No utilizar seguros y arreglar después las fallas por inconsistencia.

2.-No Bloquear.
Solamente informar a quienes efectuaron solicitudes que provocaron reclamaciones en conflicto.

3.-Asignación Previa.
Si existe algun conflicto, quitar los objetos a sus propietarios.

4.-Secuencia Previa.
No permitir secuencias circulares de solicitud.

5.-Aseguramiento de dos fases.
Se realizan primero todas las reclamaciones y si ninguna está bloqueada se inician todas las modificaciones.

La detección de puntos muertos es una tarea importante en el mantenimiento de la integridad del sistema.

El mantenimiento de la integridad de la base de datos es un problema importante en cualquier operación de bases de datos.

6.4.3. *RECUPERACION DE CAIDAS ANORMALES*

Se empleará el término *falla* para denotar la causa y la palabra *error* para la manifestación de una falla. El hecho de que algo no esté totalmente correcto se demuestra al ocurrir un error.

El hardware, el diseño de una base de datos, los programas y los mecanismos de entrada de datos, todos contienen fallas. Todos estamos familiarizados con las fallas de programa o *errores ocultos.*

Para producir resultados correctos se requieren datos y algoritmos correctos, y el sistema debe efectuar correctamente los algoritmos. En cada una de estas tareas el problema se reduce a dos subproblemas :

1.-Es necesario detectar la existencia de una falla o demostrar la ausencia de fallas.

2.-Cuando se detecta un error es necesario disponer de un medio para corregirlo y recuperarse.

Con frecuencia una sola técnica puede proporcionar tanto detección como cierta capacidad de restauración.

Es importante tomar en cuenta la existencia continua de fallas y proporcionar recursos para manejar los errores, así como prevenirlos. La detección temprana es necesaria para evitar que los errores se propaguen. Para aquellos tipos de errores que ocurran con frecuencia se requiere la correción o recuperación automática.

Ya que el mismo error puede deberse a más de una falla, el procedimiento adecuado de recuperación tal vez no resulte obvio. Detener el sistema completo siempre que ocurran errores puede tener un costo muy alto, no sólo en términos de falta de disponibiladad del sistema, sino también en términos de la confusión provocada para todos los afectados.

El éxito en un periodo de tiempo se logra si ninguna de las partes del sistema de computación falla. Por ejemplo, para q partes cada una con una probabilidad de falla de P_f, la probabilidad de operación con éxito es :

$$P_{éxito} = (1 - P_f)^q$$

Si se emplean 10 partes, cada una con una probabilidad de falla de 1% para una operación dada en un arreglo lógico en serie, la probabilidad de lograr el resultado correcto es :

$$P_{éxito} = 0.99^{10} = 0.904$$

la cual da una probabilidad de falla de casi 10%. Para 100 de tales partes, la probabilidad de operación del sistema disminuye a :

$$P_{éxito} = 0.99^{100} = 37\ \%$$

El tiempo necesario para hacer que un sistema se encuentre de nuevo en marcha se denomina *tiempo promedio de reparación (mean time to repair: MTTR)*. Una reparación puede variar desde la identificación y corrección total de una falla, a simplemente anotar en la bitácora la ocurrencia de un error y volver a iniciar el sistema.

Los componentes de un sistema de computación tienen probabilidades muy altas de confiabilidad. Un *tiempo medio entre fallas (mean time between failure : MTBF)* común para un transistor es de 40 mil horas. Este tiempo equivale a una probabilidad de error de sólo 0.000025 por hora. Un sistema moderadamente grande que utilice 100 mil de estos componentes, tendría una probabilidad de 0.78 de evitar un error en un intervalo de una hora, lo cual equivale a un MTBF de 1.3 horas.

El intercambio entre un MTBF menor debido a la complejidad del sistema y un MTTR mayor tiene que valuarse cuidadosamente en cualquier sistema en el que se desee una alta disponibilidad de la base de datos.

Una instalación duplicada de computadora puede utilizarse totalmente en paralelo o emplearse solo para proporcionar respaldo, y procesar trabajo menos crítico hasta que se presente una falla. En el primer caso el MTTR puede ser tan pequeño como el tiempo que se requiere para detectar la condición de error y desconectar la computadora en falla.

En el segundo caso, los programas que corran en la computadora de respaldo tendrán que descontinuarse y los programas de procesamiento de la máquina en falla tendrán que iniciarse en un punto adecuado. Será necesario un análisis de tipo de falla para decidir cuál de estas dos alternativas será mejor. La toma misma de la decisión puede aumentar el MTTR.

En la práctica, muchas de las fallas de circuitos de computadoras son transitorias y no se repetirán durante largo tiempo. Tales errores pueden deberse a una rara combinación del estado del sistema, variaciones de la energía, ruído eléctrico o acumulación de electricidad estática.

Aquellos componentes que se emplean solo para proporcionar capacidad de recuperación deberán estar aislados de manera que sus fallas no afecten a la producción de todo el sistema.
Los errores en los componentes de recuperación deberán generar una señal de aviso, de manera que puedan emprenderse acciones para repararlos. La respuesta de reparación debe especificarse en forma cuidadosa y formal, tomando en cuenta los posibles efectos debido al respaldo insuficiente durante el periodo de reparación.

El mejor control de calidad consiste en la generación frecuente de salida útil de los diferentes niveles de personal implicados en la recolección, preparación y captación de datos. Ni los medios mecánicos, ni las recomendaciones constantes conservarán el contenido de la base de datos en un nivel aceptable cuando los datos no se emplean con regularidad.

Después de una caída del sistema, cualquier transacción que haya emitido un comando de conclusión deberá terminarse. Esto significa que el *manejador de transacciones* (módulo o programa *que se espera* puede incluír el sistema operativo o el sistema de manejo de la base de datos) necesita tener información suficiente para concluír la transacción. esta puede volverse a iniciar desde el principio o terminarse a partir del punto de conclusión.

Para terminar una transacción a partir del punto de conclusión, cualesquiera datos que deban escribirse a la base de datos deberán conservarse seguros y disponibles.

Si no es posible concluír una transacción comprometida, la transacción deberá eliminarse para evitar dejar a la base de datos en estado inconsistente. Deberá informarse a quien solicitó la transacción que ésta no se realizó. Un manejador de transacciones deberá informar siempre al usuario cuando se ha concluído exitosamente una transacción, ya que el usuario en la terminal puede haber detectado la falla del sistema y estará tentado a volver a introducir una transacción presentada antes de una falla.

Para cancelar o deshacer una transacción es necesario que se haya preservado el estado anterior de la base de datos. Las dos siguientes técnicas son factibles para apoyar transacciones de cancelación:

Cancelación de una versión.

Todos los datos nuevos o actualizados que se vayan a escribir, se colocan en almacenamiento no utilizado previamente. No se destruyen datos anteriores. Un apuntador a los datos válidos mas recientes, se actualiza sólo después de que se hayan escrito con éxito los nuevos datos. a las copias periódicas de versiones de la base de datos se les denomina *copias de respaldo o vaciados*.

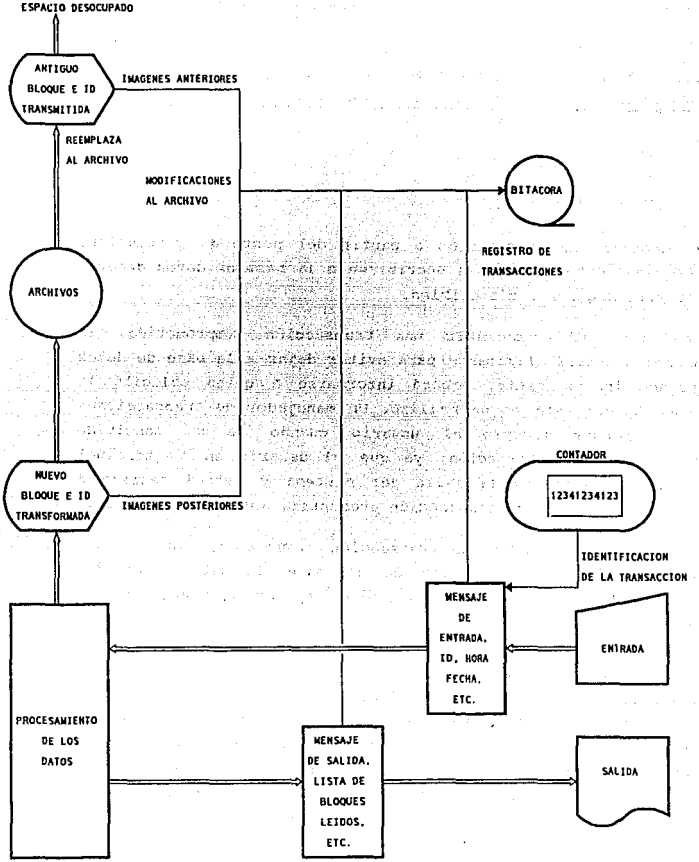


FIG.6.18.- FLUJO DE DATOS, CON BITACORA DE ACTIVIDADES.

Bitácora y Respaldo.

Todos los datos anteriores se conservan en un archivo de bitácora de transacciones. Cuando un error requiere una eliminación, los datos anteriores se localizan y los archivos se restauran a un estado que elimine el efecto de la transacción. Los bloques anteriores guardados se denominan *imágenes anteriores*. Deben escribirse antes del punto de conclusión.

En la restauración adecuada de una base de datos dañada es necesario volver a establecer los archivos de la base de datos sin los efectos de un error. Se mencionaron dos métodos para obtener el estado de la base de datos previo a alguna falla única de transacción : manejo por versiones y por bitácora.

Ahora bien, se tienen también fallas secundarias de transacción y fallas de dispositivos. Es decir, es posible que las transacciones subsecuentes hayan utilizado datos erróneos colocados en la base de datos provenientes de un error primario y que puedan generar ahora errores secundarios; por ejemplo, si un nuevo empleado se asignó al departamento equivocado, cualquier nómina y los cálculos de presupuesto subsecuentes serán erróneos.

Por otro lado, si hubo una falla de hardware provocando daño real en el disco, tal vez, no exista una versión actual a la cual aplicarle las imágenes anteriores.

Para permitir la restauración total puede conservarse una bitácora más amplia de todas las actividades que afectan la base de datos. La finalidad de estas bitácoras es permitir volver a crear los archivos. Dependiendo del tipo de error, la recuperación de la base de datos puede realizarse mediante :

Restauración.

Consiste en la combinación de versiones correctas anteriores con actualizaciones debidas a entrada correcta recibida después. La restauración procede hacia adelante a partir de una versión pasada hasta llegar al estado actual adecuado. Aún cuando el manejo por versiones no se utiliza, puede crearse periódicamente una versión o copia de respaldo.

"Volver a enrollar".

Eliminando o deshaciendo los efectos de los errores en el estado actual hasta llegar a un estado libre de errores. Después de "volver a enrollar" puede realizarse cierta restauración.

En cualquiera de los dos casos deberá colocarse considerable información en la bitácora de transacciones.

Para conservar el estado anterior de la base de datos y emplearlo al "volver a enrollar", cualquier bloque adquirido con la intención de actualizar y volver a escribir deberá copiarse a la bitácora antes de escribir la versión actualizada. Al bloque anterior se le denomina *imágen anterior*. Todas las imágenes anteriores deberán escribirse antes de llegar al punto de conclusión.

Estas imágenes anteriores pueden utilizarse para eliminar los efectos de cualquier transacción errónea. Si no ha habido transacciones subsecuentes, tal transacción podría borrarse, en lo que a la base de datos se refiere, reemplazando todos los bloques afectados con sus imágenes anteriores.

También se escribe en la bitácora el resultado completo de todas las transacciones en forma de *imágenes posteriores*, y es posible restaurar un archivo a partir de su estado inicial o intermedio copiando en la base de datos todas las imágenes posteriores.

Las imágenes posteriores son muy útiles cuando se han destruído archivos independientes de cualquier transacción específica. No es necesario volver a ejecutar ninguna transacción. El costo de la restauración puede reducirse clasificando las imágenes posteriores a la secuencia del archivo y omitiendo todas las copias angtiguas.

En la práctica, la restauración mediante imágenes posteriores depende de la disponibilidad de una copia de respaldo de una versión anterior de la base de datos. Es posible generar periódicamente copias de respaldo y conservar versiones anteriores. Cada copia de respaldo estará identificada por tiempo y fecha y por la última transacción incluída. Una copia de respaldo debe generarse mientras la base de datos está en reposo, ya que las actualizaciones durante el copiado pueden provocar que la copia sea inconsistente.

PASOS EN LA RECUPERACION.

A continuación se describirá una secuencia de acciones que podrá utilizarse cuando ocurra una falla del sistema.

a) Detección de error.
b) Determinación de la fuente de error.
c) Ubicación de errores secundarios.
d) Aplicación de correcciones.

El proceso de recuperación se inicia al detectar la existencia de un error. Puede haber una caída del sistema, observado de inmediato por los operadores de la computadora, quienes notarán que el equipo se ha detenido o que las terminales ya no responden en la forma esperada. Una caída tiene la ventaja de que la causa del error a menudo resulta obvia y que el periodo*p+51%de tiempo de operac incorrecta ha sido pequeño.

Si el error se debe a un registro dato incorrecto obtenido a partir del archivo, una búsqueda inversa a través de la transacción puede localizar el punto en el que el dato se colocó en el archivo. Este tipo de error ha tenido tiempo de propagarse.

Si el error se debe al mal funcionamiento del dispositivo mismo de almacenamiento, resulta suficiente con localizar la imágen posterior que se guardó cuando el bloque se escribió o iniciar la restauración reemplazando el bloque incorrecto.

Si las fallas son provocadas por errores en la ruta de acceso, tal vez sea posible reconstruír la estructura del archivo sin hacer referencia a las porciones de datos de los registros.

Si la falla se detecta sólo por la aparición de *salida incorrecta*, puede transcurrir un intervalo de tiempo muy largo entre la ocurrencia y la detección del error. Una búsqueda inversa a través de la bitácora de transacciones puede determinar todas las actualizaciones y encontrar las transacciones sospechosas.

Para decidir cuál es la mejor acción correctora es necesario determinar la extensión del daño. Después de una caída o cuando el procesamiento sea interrumpido debido a una señal de error, es necesario determinar las áreas del archivo de datos que sean sospechosas y cuál fue la transacción que no se concluyó.

La búsqueda inversa a través de la bitácora puede proporcionar una lista de las transacciones iniciadas y no concluídas. Si estas transacciones realizaran algunas actualizaciones de archivo, las imágenes anteriores pueden restaurar los archivos a su estado anterior.

Cuando se ha detectado un error que provocó una modificación inadecuada a un archivo, un rastreo a través de las listas de actividad encontrará aquellas transacciones que emplearon el bloque incorrecto. La identificación de todos los bloques y los mensajes afectados puede producirse mediante una búsqueda interactiva en la bitácora de transacciones.

Cuando un error se detecta a tiempo, las transacciones afectadas de manera secundaria serán pocas.

Si la extensión del daño es limitada, puede utilizarse un proceso de "volver a enrollar". Las porciones dañadas del archivo se restauran aplicando primero aquellas imágenes anteriores a los bloques en error y reemplazando después las transacciones incompletas.

Si el daño es grande, tal vez sea necesario empezar haciendo una copia de respaldo del archivo y aplicarle todas las imágenes posteriores en secuencia cronológica, excepto aquellas asociadas con transacciones incompletas.

En la figura 6.19 se presenta una panorámica del proceso.

Un programa para el proceso de recuperación deberá ser capaz de volver a iniciarse a si mismo en cualquier etapa, de manera que las fallas durante la recuperación no se vuelvan desastres.

6.5.- OPERACION DEL SISTEMA

Los beneficios de los sistemas de bases de datos se obtienen sólo después de que se concluyen el diseño y la implantación.

La buena operación del sistema de bases de datos requiere la cooperación de un gran número de posibles usuarios, de especialistas en análisis de datos, científicos de la información, expertos en varios aspectos de la computación, especialistas en comunicaciones, el administrador de la base de datos y representantes de la gerencia, quienes invertirán recursos de la empresa en el esfuerzo.

Los componentes terminantes de una operación de bases de datos son los usuarios y los datos; ya que los usuarios determinan los objetivos de sus aplicaciones y la semántica de los datos determina la forma en que éstos objetivos pueden satisfacerse.

Los usuarios son las personas que interactuan con los sistemas de información. El grado de participación puede variar dependiendo del tipo de usuario :

Los usuarios directos.

Son quienes realmente interactúan con el sistema. Ellos alimentan datos o reciben salidas, quizás por medio de una terminal.

Los usuarios indirectos.

No interactúan directamente con el hardware o el software, pero se benefician de los resultados o informes producidos por el sistema. Estos usuarios pueden ser gerentes de algún área de los negocios que utilicen el sistema.

Los usuarios administrativos.

Son quienes tienen responsabilidades en la administración de los sistemas de aplicación. Ellos tienen la autoridad para aprobar o desaprobar la inversión en el desarrollo de aplicación; también tienen la responsabilidad de organización para la efectividad de los sistemas.

Los tres tipos de usuarios son importantes. Cada uno tiene información esencial en relación con la empresa.

Cuando el personal de sistemas instala la nueva aplicación y construye los archivos de datos que se necesiten, entonces se dice que el sistema está *puesto en marcha*.

Los desarrolladores del sistema pueden escoger una prueba piloto para la operación del sistema solamente en un área de la compañía (de acuerdo con el tamaño de la empresa y el riesgo asociado con el uso de la aplicación).

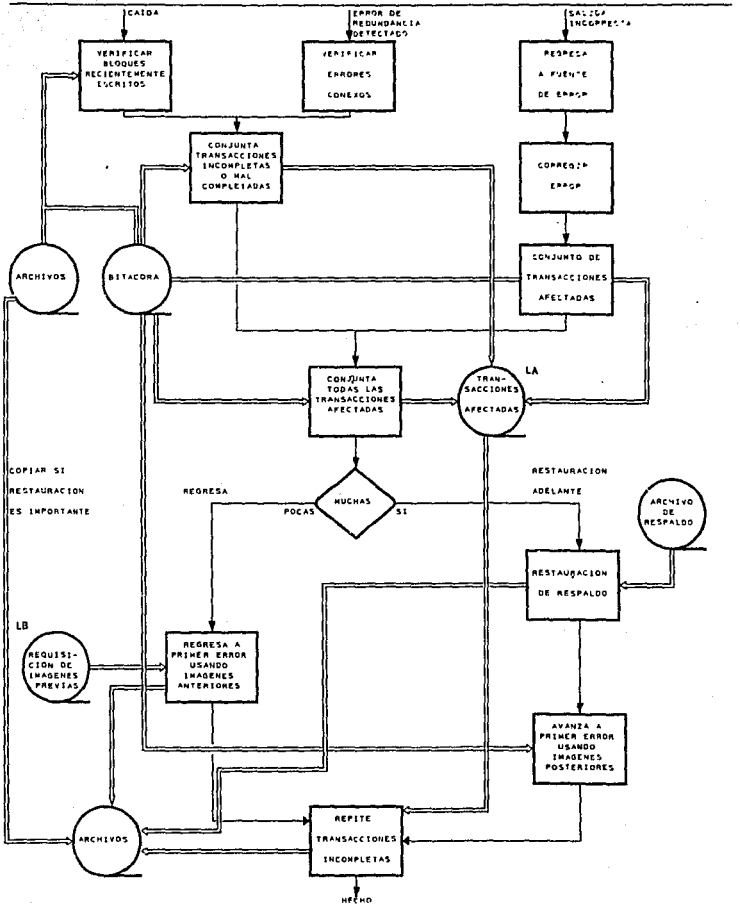


FIG.6.19.- PROCESO DE CORRECCION.

Los sistemas bien diseñados y técnicamente elegantes pueden tener éxito o fallar debido a la forma en que se operan y se utilizan; por lo tanto, las personas que trabajarán con el sistema o que se verán afectados por éste deben conocer con detalle las funciones que desempeñarán, cómo utilizarán el sistema y lo que éste hará o no. Tanto los operadores como los usuarios necesitan capacitación.

Muchos sistemas dependen del personal del centro de cómputo, quien tiene la responsabilidad de mantener el equipo en buenas condiciones, así como de proporcionar el servicio de apoyo necesario. Su capacitación debe garantizar que estan en condición para manejar todas las operaciones posibles, tanto las de rutina como las extraordinarias.

Si el sistema requiere la instalación de equipo nuevo, la capacitación del operador deberá incluír aspectos fundamentales, como la manera en que debe encender el equipo o la forma de operarlo; cómo apagarlo y qué es lo que constituye la operaciön y el uso normal. Los operadores también deben recibir instrucciones sobre los errores comunes que pueden ocurrir, cómo detectarlos y tomar las medidas necesarias cuando se presenten.

Cuando un sistema está concluído, depurado adecuadamente y poblado de datos, empieza el verdadero trabajo. Para el usuario, el valor de la base de datos no radica en el sistema de la base de datos, sino en el contenido y particularmente en los resultados de las consultas planteadas por usuarios y por los programas de éstos. Un mal sistema puede frustrar todos los aspectos positivos.

Después de que el sistema se pone marcha, se lleva a cabo una revisión del sistema. Esta revisión debe ser un proceso formal para determinar qué tan bien está trabajando el sistema, cómo se ha aceptado y si son necesarios varios ajustes.

Seguramente habrá necesidad de modificaciones a causa de cambios que se presentaron cuando el sistema se estaba implantando. En este caso, será mejor que los cambios se hagan en una copia de la base de datos, de manera que el uso normal no se vea afectado hasta que se verifiquen los cambios.

Puede existir mucha diferencia entre lo esperado y el producto entregado para aquellos usuarios que no participaron en el esfuerzo de desarrollo, es conveniente advertir al usuario que la instalación del sistema de base de datos no proporcionará todos los beneficios posibles.

A continuación se describen las etapas que se presentan una vez que se ha desarrollado un sistema de bases de datos.

-Prueba operativa del sistema de bases de datos.

Se aprueban el hardware y el software utilizando datos piloto.

-Prueba funcional de los procedimientos de captación de entrada de datos.

En este punto es necesario asegurarse de que la entrada fluye a la base de datos uniformemente y sin errores.

-Verificación del contenido de datos.

Los beneficios de la integración de los datos sólo puede probarse si los usuarios determinan que la cantidad de los datos almacenados es la misma que la de sus colecciones locales de datos.

-Generación de informes.

El empleo de la base de datos para generar informes adecuados para el medio proporciona la experiencia operativa inicial y la retroalimentación de los datos.

-Construcción de Modelos.

Es posible probar hipótesis y realizar proyecciones basadas en datos pasados. Esta es la etapa en la que en realidad el sistema puede generar información y datos útiles para la toma de decisiones y la planeación.

El tiempo necesario para llegar a la etapa operativa final puede ser de muchos años, ya que las diferentes etapas requieren herramientas cada vez más complejas y de personal más especializado.

A los usuarios se les puede dificultar plantear las causas por las que no estén satisfechos con aspectos del desempeño de los sistemas. Es importante la sensibilidad hacia las necesidades de los usuarios.

Ahora, se verán las medidas de la utilización del sistema que pueden vigilarse.-

Estadísticas de utilización de dispositivos.

Tiempos de porcentaje de actividad de procesadores, canales, controladores y discos. Es conveniente tener tanto valores promedio como valores que pertenezcan a los periodos de mayor actividad de operación del sistema.

Estadísticas de utilización de archivo.

La recuperación, obtención del siguiente registro y actualización, son importantes en la toma de decisiones acerca de la organización del archivo. Otra medida es la densidad de archivo (espacio utilizado comparado con espacio asignado).

Estadísticas de utilización de registros.

La frecuencia con la que se efectúan accesos a los registros para lectura o actualización proporciona una medida que puede emplearse para lograr optimización en sistemas en los que los registros están encadenados.

Estadísticas de utilización de atributos.

La frecuencia con la que los valores atributo se solicitan, actualizan, utilizan como llaves o se emplean como argumentos posteriores de búsqueda, proporciona los datos para la selección óptima de membresía de eneada.

Las medidas pueden obtenerse por vigilancia o mediante muestreos continuos. El depósito más obvio es una cinta común de bitácora.

Los datos de vigilancia listados anteriormente proporcionan datos referentes a la actividad para registros y atributos, los que juntos describen la actividad de cualquier elemento en términos de promedios.

Los datos y el sistema pierden valor con el tiempo. Conforme los datos envejecen, pierden valor; llegará el momento en que su valor hará inconveniente continuar almacenándolos en línea. Los datos antiguos pueden conservarse en cinta fuera de línea mientras exista la probabilidad de que puedan llegar a necesitarse.

Para el almacenamiento archivado a largo plazo lo mas conveniente es generar cintas en formato de salida, las cuales podrán leerse cuando sea necesario mediante procedimientos de entrada.

Llegará un momento en que la base de datos se vuelve obsoleta. Este hecho a menudo está asociado con el hardware que se ha vuelto anticuado, pero lo peor es cuando el software es inadecuado.

El costo de mantenimiento de hardware y software tiende a disminuír conforme se eliminan los errores, pero comienza a aumentar si se está forzando el empleo de viejas facilidades de hardware y software para que sean compatibles con nuevos desarrollos.

Una adaptación necesaria puede costar más que una nueva implantación. Debido a ésto puede hablarse del *ciclo de vida* de un sistema: comenzando con el diseño, desarrollo, implantación, carga de datos, operación y mejoría, operación y mantenimiento; y concluyendo con una transferencia de servicios para renovar un sistema. Una vez que se determina el ciclo de vida de un sistema, es posible tomar otras decisiones, por ejemplo, pueden rechazarse inversiones en mejoras al sistema que no producirán beneficios durante lo que quede del ciclo de vida.

6.6.- ADMINISTRACION DE BASE DE DATOS.

En una organización en la que muchos usuarios y programadores comparten la base de datos, se requirá cierto control conjunto de la organización de los datos. Esta puede implantarse mediante un sistema automático de protección que impida cambios en los sistemas que pudieran afectar a los demás.

Los conflictos comunes se presentan cuando una mejora general del sistema provoca importantes molestias a unos cuantos usuarios, o cuando nuevas necesidades de algunos de los usuarios afectan ligeramente a muchos otros. A este conjunto de decisiones de control se le denominará *manejo de la base de datos* y pueden ser acciones automáticas o manuales.

Se requieren parámetros que controlen el desempeño del sistema y la asignación de privilegios de acceso. El mantenimiento de copias adicionales de los datos, de enlaces de índices y el mantenimiento de los resultados reales, mejora el desempeño de recuperación de los sistemas, pero lleva a la redundancia en los datos almacenados.

Para evaluar qué tanta redundancia es efectiva en cuanto a costo, el administrador de la base de datos vigilará continuamente la operación del sistema de base de datos, en distintos niveles.

La administración de la base de datos puede estar a cargo de varias personas, o ser un trabajo de tiempo parcial; esto depende del número de funciones que se asignen al DBA. En la figura 6.20 se muestra un posible papel para el administrador de la base de datos.

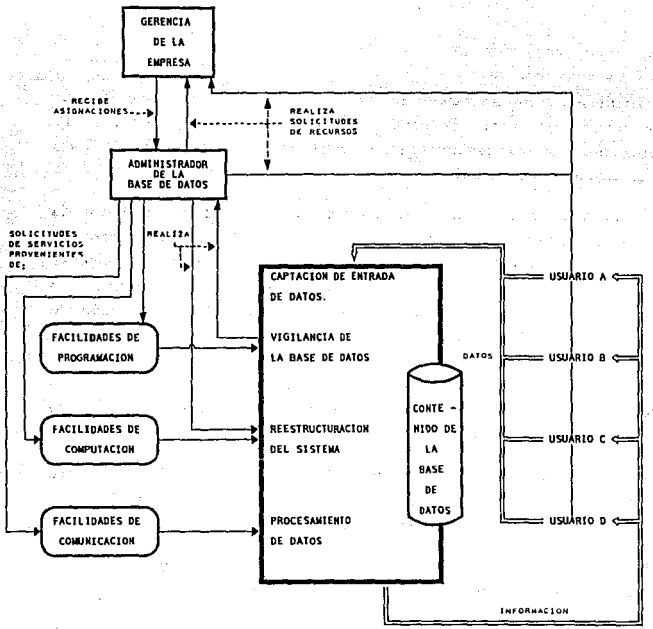


FIG.6.20.- UBICACION DEL ADMINISTRADOR DE LA BASE DE DATOS

El administrador de sistemas se preocupará constantemente de que exista una proporción adecuada entre desempeño y costo. Un administrador estará desarrollando contínuamente herramientas para medir la productividad del sistema.

El DBA actúa como mecanismo de control de un sistema que maneja ciclos de información de los usuarios. Para realizar la función de control, el DBA puede reestructurar la base de datos y obtener facilidades adicionales para análisis, procesamiento o comunicación. Los recursos adicionales requieren la asignación de capital de inversión que el DBA tendrá que obtener de la gerencia.

Si el DBA no puede satisfacer a un usuario, éste podría dirigirse a la gerencia para solicitar que se aumenten las asignaciones, de manera que se satisfagan las necesidades.

El esquema es la herramienta principal de que dispone el DBA para la asignación de recursos. El esquema interno proporciona control sobre la eficiencia y la responsabilidad de los procesos.

La selección del esquema externo determina la visión operativa que un usuario tiene del modelo de la base de datos.

Además, el DBA tendrá herramientas de vigilancia y medios para reestructurar la base de datos. El DBA debe comprender las necesidades del usuario y la capacidad de manejar estas necesidades.

El administrador de la base de datos tiene privilegios; debido a ésto, él deberá tener un conocimiento que está más allá del alcance de la mayoría de los usuarios y programadores. El usuario trabaja con un modelo simplificado y no es capaz de mejorarlo o hacerlo funcionar mal.

Por otra parte, el administrador de la base de datos tiene conocimiento, acerca de la estructura interna actual. Para salvaguardar el concepto de comunicación bien estructurada entre niveles, es posible que a un administrador de base de datos se le niegue el privilegio de escribir programas de manipulación de datos.

Conforme las bases de datos se vuelven más grandes y complejas, el control centralizado por un solo administrador podría dejar de ser aceptable. El conocimiento necesario para realizar la función, y el costo de no lograr controlar adecuadamente una base de datos es muy grande, puede hacer insostenible la posición. También es dudoso que un comité pueda hacerlo mejor. Un problema asociado es la falta de disposición de los gerentes de empresas de ceder mucho de su poder. Estas consideraciones pueden limitar el crecimiento de las bases de datos y propiciar la distribución de sus funciones.

CAPITULO 7.- IMPLEMENTACION DE LA BASE DE DATOS. EL DISEÑO FISICO.

INTRODUCCION.

Uno de los factores importantes para que un usuario quede satisfecho con un sistema de Base de Datos es su rendimiento. Dado que el objetivo de un sistema de Base de Datos es simplificar y facilitar el acceso a la información, si el tiempo de respuesta es demasiado largo, se va en contra de ésta premisa, por lo que el valor que el usuario da al sistema disminuye. En consecuencia, debe plantearse que el rendimiento de un sistema depende de la eficiencia de las estructuras de datos que utiliza para representar la información en la base de datos, y de que el sistema pueda operar de manera eficiente con éstas estructuras. Es claro que el optimizar resultados, depende de la aplicación que espera dársele al sistema, por lo que también de esto depende el tipo de implantación.

7.1 ESTRUCTURAS FISICAS.

7.1.1. EVALUACION DE DISPOSITIVOS DE ALMACENAMIENTO MASIVO DE ACCESO DIRECTO.

El estudio y análisis de los diferentes dispositivos de almacenamiento masivo de acceso directo se justifica por el hecho de que es en éste renglón donde la eficiencia es el factor más importante, puesto que los tiempos de acceso a la información dependen en gran medida del comportamiento mecánico del dispositivo de almacenamiento, así como de la manera en que están organizados los datos en éste.

DIFERENTES TIPOS DE DISPOSITIVOS.

De cualquier manera el objetivo es lograr almacenamiento masivo a bajo costo, para lo cual se siguen dos enfoques.

- Capacidad aumentada de almacenamiento por unidad para reducir el costo de almacenamiento por caracter.

- Reducción de la dependencia en operaciones manuales y partes mecánicas para lograr mayor confiabilidad y almacenamiento compacto.

Discos Magnéticos.

Los discos magnéticos se están acercando a sus límites de capacidad de almacenamiento debido a los límites de tamaño de su área magnetizable, la cual está determinada por las dimensiones de las cabezas de lectura escritura. Las limitaciones mecánicas se deben al posicionamiento de los ensambles de cabezas sobre las pistas, aun cuando se utilizan controles de retroalimentación

que alinean dinámicamente la cabeza sobre la pista que se lee. En contraparte, se han desarrollado técnicas de fabricación de éstos dispositivos que hace que su costo de producción sea muy bajo, con lo que se logra el objetivo inicial, además los adelantos en materia de superficie magnetizable le han dado un segundo aire a la tecnología de discos magnéticos.

Tecnología óptica.

En este tipo de tecnología, el área ocupada en almacenar un bit es mucho menor que la utilizada por los dispositivos magnéticos, sin embargo, ésta tecnología basada en microfichas tiene el inconveniente del alto costo de revelado de las mismas.

Una alternativa actual es el uso de tecnología láser en el almacenamiento. En este sistema, un bit se registra quemando una minúscula perforación en el recubrimiento opaco de una banda de Mylar. Para leer los datos, un haz de luz de menor energía seguirá a la posición del bit y se transmitirá si está registrado un "1".

El primer inconveniente de estos dispositivos es que sólo podían ser grabados una vez, aunque permiten todas las lecturas posibles, es decir, se comportaban como ROM's (más comunmente conocidos como CD_ROM), lo cual los hacía inconvenientes para aplicaciones de Bases de Datos, a no ser que fueran utilizados como simples catálogos para aplicaciones muy específicas, sin embargo, es tanto el avance tecnológico en materia de láser, que ya están en estudio los primeros modelos de CD-RAM, aunque los costos de fabricación todavía son muy altos, sin embargo, es la tendencia del futuro.

No se consideran medios secuenciales (como las cintas) por que tienen demasiadas restricciones para ser útiles en un sistema de Bases de Datos.

Otra cuestión a considerar es que el hecho de implementar tecnología avanzada, está en estrecha relación con la independencia de los datos, en la cual el avance es tan grande, que podemos utilizar hoy en día cualquier tipo de almacenamiento de acceso directo (desde diskettes para microcomputadoras, hasta grandes discos en sistemas de hardware muy diferentes, y bajo sistemas operativos muy diferentes) y solamente tenemos que utilizar software de conversión de códigos para poder utilizar ésta información en cualquier máquina, por difícil que esto parezca.

Es claro que el elegir un medio de almacenamiento está íntimamente ligado a la naturaleza de la aplicación que se le quiera dar al sistema de Base de Datos, ademas del tamaño que se prevea tenga la misma y el tipo de hardware con que se cuente para soportar la aplicación.

Como un ejemplo de éste análisis , supongamos una aplicación para una oficina, en la cual se quiere llevar el control de asistencia y puntualidad de los empleados, y cuyo número no pasa los 50. Dado que en una oficina de ésta naturaleza es muy factible que se tenga una microcomputadora, la atención se centraria en dispositivos tales como el Diskette y el Disco Duro (HD). Ningún costo mayor podría ser justificado, por lo que queda descartada la posibilidad de manejar otro tipo de hardware de almacenamiento.

El caso contrario se presenta en entidades con grandes volúmenes de información, y con necesidades de tiempo de proceso muy pequeño, como el caso de la S.E.P , donde las Bases de datos más pequeñas son de 300 Mb y sus tiempos de proceso por registro deben ser de segundos, por lo que en éstas circunstancias no se usan microcomputadoras, sino máquinas más grandes, y con dispositivos de almacenamiento de velocidades superiores al Disco Duro de una microcomputadora.

Es en éstas aplicaciones donde la independencia de datos se vuelve más patente, pues podemos ver que alguna información procesada en la gran computadora, es transferida a microcmputadoras para su presentación final, lográndose esto en un tiempo relativamente corto y con un costo relativamente bajo.

7.1.2 ORGANIZACION DE ARCHIVOS.

Para ésta sección se supondrá que tenemos como medio de almacenamiento masivo de acceso directo al más común de estos dispositivos: el disco.

Un archivo está organizado lógicamente como una secuencia de registros. Estos registros se mapean a bloques del disco. Los archivos son construcciones básicas con que cuentan los sistemas operativos, por lo que se supondrá que existe un sitema de archivos correspondiente.

Aunque los bloques en el disco (dependientes de las propiedades físicas del disco y del sistema operativo) son de tamaño fijo, el tamaño de los registros varia. En una Base de Datos relacional las tuplas pertenecientes a distintas tablas son de diferente tamaño, en una de red, el registro dueño (owner) tal vez sea mayor que el registro miembro (member), etc.

Una forma de enfocar el mapeo de la Base de Datos a los archivos de sistema operativo, es emplear varios archivos y almacenar en cada uno de ellos registros de una misma longitud (fija). Una alternativa es estructurar los archivos de tal manera que puedan manejar registros de varias longitudes diferentes. Los archivos de registros de longitud fija son más fáciles de implantar que los archivos de registros de longitud variable, sin embargo, las técnicas de implantación en muchos casos son similares.

Registros_de_Longitud_fija.

Consideremos el registro de la fig. 7.1.

Si consideramos que normalmente un caracter ocupa un byte, un entero 4 bytes y un real 8 bytes, tendríamos que cada registro ocuparía 52 bytes, y sería fácil la localización de un registro específico, sin embargo, se presentan dos problemas para la implementación :

- Es difícil eliminar un registro de ésta estructura. El espacio que ocupa el registro que se va a eliminar debe llenarse con otro registro del archivo, o marcarse para que el sistema lo ignore.

- A menos que el tamaño físico del bloque en disco sea múltiplo de 52, algunos registros quedarán divididos en dos bloques con lo que el acceso a éste registro requerirá accesar dos bloques.

Cuando se elimina un registro podría moverse el registro que le seguía al lugar que ocupaba el registro eliminado, y así sucesivamente hasta mover todos los restantes hacia adelante. Este enfoque requiere mover una gran cantidad de registros. Podría ser más conveniente mover el último registro del archivo al espacio vacío.

Sin embargo, no es conveniente mover registros para ocupar el espacio libre, ya que esto también requiere acceso a bloques adicionales, lo aceptable sería dejar éste espacio disponible para posteriores inserciones de registros ya que estas son más comúnes que las eliminaciones. Por lo tanto, es necesario incluír una estructura extra, llamada *encabezado de archivo*.

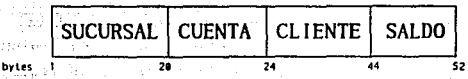


FIG. 7.1 REGISTRO DEPOSITTO

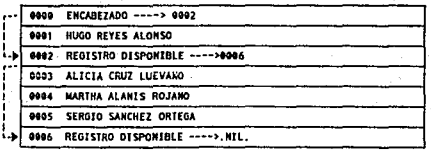


FIG. 7.2 ARCHIVO CON ENCABEZADO

El encabezado de archivo contiene información diversa acerca del archivo. En este momento nos basta con poner ahí las direcciones del primer registro borrado, dado que de no hacerlo sería muy difícil su localización. Este primer registro se utiliza para almacenar la información del segundo disponible, etc.. De cierta manera son los primeros tipos de *apuntadores* [1] que se implementan.

La Fig. 7.2 muestra esta estructura. Al insertarse un nuevo registro se hace en la posición a la que apuntaba el encabezado, y el contenido del encabezado se modifica para que apunte al registro al que apuntaba el registro utilizado. Si el encabezado apuntaba a tierra (es decir, si no hay registros disponibles y el encabezado apunta a un registro *nulo*), el registro se añade al final del archivo.

El uso de apuntadores requiere de cierto cuidado, puesto que si se elimina o modifica un registro que apuntaba hacia otro, ahora la información es incorrecta, y puede ocasionar desde *registros colgantes* (registros disponibles que no pueden ser utilizados por no poder encontrarse su dirección), hasta grandes inconsistencias en la Base de Datos, como el caso de que se modificara el primer registro libre y se pusiera ahí la dirección de otro que estuviera ocupado. Cuando el manejador de Base de Datos quisiera utilizar los registros libres, se toparía con que en la serie de registros denominados libres, existen registros ocupados, lo cual para la mayoría de los manejadores de Base de Datos representa un error fatal que hace abortar cualquier proceso.

[1] apuntador.- que indica el lugar donde un registro esta almacenado

Por último señalaremos que es fácil implementar operaciones de inserción o eliminación con este tipo de archivos, puesto que la longitud del registro libre es exactamente igual a la del registro a insertar, caso contrario al de los registros de longitud variable.

Registros de Longitud Variable.

Pueden presentarse registros de longitud variable en una Base de Datos por varias razones :

- Almacenamiento de varios tipos de registros en un archivo.
- Tipos de registros que permiten uno o más campos variables.
- Tipos de registros que permiten campos repetidos.

Existen varias técnicas para implantar registros de longitud variable, para ilustrarlas se utilizará la sig. descripción de registro variable.

```
REGISTRO-SUCURSAL.
        CAMPO1 = SUCURSAL
        CAMPO2 = ARREGLO DE N REGISTROS-CUENTA
                        REGISTRO-CUENTA.
                                        CAMPO1 = CUENTA
                                        CAMPO2 = CLIENTE
                                        CAMPO3 = SALDO.
                        FIN-REGISTO-CUENTA
FIN REGISTRO-LISTA
```

Se define registro-cuenta como un arreglo con un número de cuentas limitado sólo por la capacidad del disco, de tal manera, el registro no tiene límites de longitud.

Representacion como cadena de caracteres.
Un método sencillo para implantar los registros de longitud variable es agregar un símbolo especial de fin de registro al final de cada uno. Así puede almacenarse cada registro como una cadena de bytes consecutivos. La Fig. 7.3 muestra éste tipo de representación.(⊥ es la marca de fin de registro)

Esta representación tiene varias desventajas. Las más grandes son:

- No es fácil volver a utilizar el espacio que deja un registro eliminado puesto que los tamaños de los registros a insertar no se ajustan al de los registros suprimidos. Todo ello resulta en muchos pequeños espacios desperdiciados .

- En general, los registros no disponen de espacio para crecer. Si un registro de longitud variable se hace más largo, debe moverse, si el registro está *atado* (*o sujeto*)[2], el movimiento resulta costoso.

Por ésta razón no es conveniente implementar ésta representación.

2 SE DICE QUE UN REGISTRO ESTA ATADO CUANDO CONSERVA UNA DIRECCION HACIA OTRO REGISTRO, POR LO QUE SE ESTABLECEN POLITICAS QUE INDICAN QUE ESTE REGISTRO NO DEBE MOVERSE O MODIFICARSE, ES DECIR ESTA SUJETO.

Representacion de Longitud Fija.
Una alternativa para implementar registros de longirud variable sin tener el problema de costos de movimiento, es la representación de longitud fija, mediante la cual se utilizan uno o más registros de longitud fija para representar uno de longitud variable.

Existen dos técnicas comunes para ésta operación :

- Espacio reservado. Se parte de la premisa de que los registros, por muy variables que sean, nunca van a exceder un límite, por lo tanto se utilizan registros de longitud fija del tamaño límite. El espacio que no se utilize se llena con símbolos de fin de registro (⊥).

- Apuntadores. El registro está compuesto por una cadena de registros de longitud fija, encadenados por medio de apuntadores.

El método de espacio reservado es útil cuando la mayoría de los registros se acercan al tamaño límite, de otra manera se desperdiciaría mucho espacio.

En el método de apuntadores existe la desventaja de que se desperdicia espacio en todos los registros, menos en el primero, ya que sólo éste requiere de la información que se repite en todos los registros, sin embargo, si es necesario que se considere el espacio requerido para almacenar ésta información en cada registro ya que se trata de registros de longitud fija.

Para evitar este inconveniente, se utiliza una estructura de bloqueaje de registros en un archivo :

- Bloque ancla, que contiene el primer registro de una cadena.
- Bloque de desborde, que contiene todos los registros que no son el primer registro de una cadena.[3]

De esta manera, *todos los registros de un bloque* tienen el mismo tamaño.

Organización de registros en bloques.
Dado que los datos se transfieren entre el disco y la memoria por bloques, es conveniente asignar los registros a los bloques de tal manera que cada bloque contenga registros relacionados entre si.

Si se asignan al azar los registros a los bloques, puede darse el caso de que se deba tener acceso a un bloque diferente para cada regsitro que se requiera.

Se puede ahorrar tiempo de acceso planeando la distribución de los registros en los bloques para que se puedan accesar varios de ellos en un mismo bloque, ademas, dado que el rendimiento de un sistema de Base de Datos está íntimamente relacionado con el tiempo de acceso a disco, el manejo adecuado de los bloques puede mejorar significativamente el rendimiento.

[3] TOMADO DE KORTH Y SILBERCHARTZ. FUNDAMENTOS DE BASES DE DATOS.

Hasta cierto punto, es fácil la inclusión de registros del mismo tipo en un bloque, digamos, como en el caso de meter cuentas de una misma sucursal en un mismo bloque, sin embargo, esto sólo se cunple cuando la Base de Datos no cambia, de otra manera, si se quisiera insertar un registro, es seguro que el lugar que le corresponde ya está ocupado, por lo que es necesario mover muchos registros, o abandonar la idea de implementarlos de ésta manera.

Como alternativa, existe una estructura un poco mayor, llamada *Cubeta*. Una cubeta consiste en tantos bloques como sean necesarios para representar datos, pero no puede compartirse un bloque entre dos cubetas.

El primer registro de la cubeta contiene la información repetitiva, mientras que los restantes contienen la información particular de cada registro, digamos, en el caso del ejemplo bancario, el primer registro contendrá el valor de *sucursal* mientras que los restantes tendrán el valor de *registro-cuenta*[4].

No es necesario encadenarlos, pues se sabe que son tratados como un solo registro de longitud variable. (Aunque es conveniente para la recuperación de espacio en registros borrados). Una cubeta puede requerir de varios bloques, los cuales están encadenados con métodos similares a los de los registros, es decir, existe un *encabezado de bloque* en el que se almacenan apuntadores a la cadena de bloques de la cadena.

En la práctica, cuando las supresiones son muchas, pueden quedar bloques vacíos, por lo que es conveniente mantener una cadena de bloques vacíos, además, conviene tener los bloques en un mismo cilindro del disco, puesto que ahorra tiempo de acceso.

[4] REFERENTES A LA FIG 7.2 DE ESTE CAPITULO.

En muchos casos, es tal el movimiento de la base de datos, que los cilindros adyacentes de un mismo disco no soportan tener todos los bloques relacionados de una cubeta, además se crean fragmentaciones muy alejadas, por lo que es conveniente utilizar procedimientos de reorganización de la Base de Datos. Estos procedimientos consiten en vaciar todos los datos de la Base en cintas, y después volver a cargarlos, previo un reacomodo de los bloques y de las cubetas.

Por último, en éste punto señalaremos que se puede lograr eficiencia combinando tipos de archivos, ya sean de métodos de recuperación de acceso contiguo (CAM) o de acceso directo (DAM) según sean las necesidades de la aplicación.

7.2 ORGANIZACION DE DATOS.

7.2.1. PILA O STACK.

Ya hemos hablado de archivos y sus formas de organización, de registros y sus diferentes formatos, sin embargo, es necesario hacer una retrospectiva de las formas de organizar datos.

Existen principalmente dos métodos para organizar datos: listas y árboles.

Dentro de las estructuras de datos tipo listas, encontramos a las pilas. Las pilas son estructuras en las cuales las inserciones y supresiones de datos se hacen siempre por el último elemento ingresado en la lista. A ésta filosofía se le denomina LIFO (last input first output ó ultimo en entrar primero en salir). Este tipo de estructuras se utilizan para almacenamiento secuencial, sin embargo, existe su correspondiente estructura para almacenamiento de acceso directo : la *pila ligada.*

En general, las listas ligadas, tienen la ventaja de que si se necesita insertar o suprimir elementos, basta con mover los apuntadores, pero tienen la aparente desventaja de que se utiliza un espacio extra para las ligas, sin embargo se puede ganar en espacio puesto que se puede usar todo el espacio disponible ya que los apuntadores hacen que no existan fragmentaciones difíciles de accesar, con lo que también se gana en tiempo de respuesta.

Las pilas son una estructura muy usada en Bases de Datos de tipo relacional, puesto que agiliza el acceso a registros recién ingresados, sin embargo, para bases de datos muy grandes se vuelve ineficiente, ya que implica utilizar algoritmos de búsqueda lineales, lo cual retarda el tiempo de acceso.

En otros tipos de Base de Datos, como la jerárquica, se mantiene un espacio disponible de dónde tomar registros vacíos, a los cuales se les insertarán datos, éste espacio no es más que una pila ligada.

7.2.2. *HEAP O MONTON.*

Ya hablamos de que las estructuras de datos también pueden ser jerárquicas, como en el caso de los árboles. En este tipo de estructuras, a cada uno de los escalones se les llama nivel, y a cada uno de los elementos se les llama nodos.

Para que una estructura de datos pueda ser considerada como un árbol debe cumplir con las siguientes características :

- Que exista un nodo especial llamado raiz, desde el cual poder accesar cualquier registro de información.

- Que existan subconjuntos de elementos llamados, t_0, t_1,...,t_n, tales que cada uno de ellos sean ha su vez una raiz de la cual accesar todos los nodos en los niveles posteriores. Cada una de éstas ramificaciones es llamada *sub rbol*.

De aquí se desprende que los procesos en las estructuras de árbol pueden ser recursivos, es decir denominarse subconjuntos para proceso, la cual es una característica importante para la rapidez de proceso.

Dentro de las estructuras de árbol, se denomina árbol binario a un árbol que tiene dos ramas como máximo, (es decir, en el siguiente nivel de cada nodo, solo tiene uno o dos hijos) y su aplicación es muy numerosa, dado que es más fácil de manipular.

Algunas aplicaciones de éstos árboles se usan en el diseño de compiladores, para transformar operaciones algebraicas jerárquicas en cadenas de operaciones con notación polaca.

Son muy usados en algoritmos de acceso a Base de Datos, sobre todo las que incluyen algún aspecto del enfoque jerárquico y sus combinaciones, puesto que el ordenamiento a través de estas estructuras es muy rápido.

Un caso especial de estos árboles binarios es el denominado *Heap o Monton*.

Un monton es un árbol tal, que dado el valor I para el nodo padre, el hijo izquierdo tendrá como valor 2*i , mientras que el hijo derecho tendrá como valor (2*i) + 1.

Por otra parte, las asignaciones se harán de tal forma que los valores asignados a la descendencia, serán siempre más pequeños que los asignados a los padres. Por lo tanto, el valor de cualquier trayectoria desciende a medida que se desciende en el árbol.

NORTE	12345-1	HUGO	900000	12345-2	MARTHA	1230000	⊥				
SUR	12346-1	SERGIO	900430	⊥							
ORIENTE	12347-1	CONY	123400	12347-2	TONY	450000	12347-3	MARCE	1200000	⊥	
NORTE	12348-1	MIKE	956902	⊥							

FIG. 7.3 REGISROS CON MARCA DE FIN DE REGISTRO.

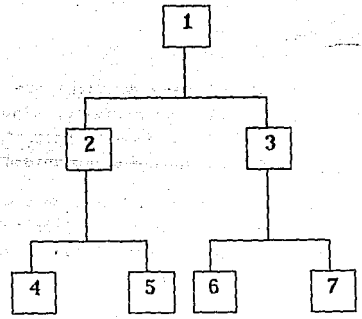


FIG. 7.4 ARBOL BINARO (MONTON).

CAPITULO 8.- SOLUCION DE PROBLEMAS REALES MEDIANTE USO DE BASES DE DATOS.

8.1.- BASES DE DATOS DISTRIBUIDAS.

Una base de datos distribuida es una base de datos que no está almacenada en su totalidad en un solo lugar físico, sino que se distribuye a lo largo de una red de computadores geográficamente separados que se conectan por medio de enlaces de comunicación.

Los sistemas distribuidos interconectan dispositivos de computadora en diferentes localidades (o nodos) para permitir el procesamiento local de los datos; la transmisión de éstos lleva informes de resumen a otras localidades, como las centrales de las empresas.

Un ejemplo de base de datos distribuida es un sistema bancario donde la base de datos de las cuentas de los clientes se distribuye en todas las sucursales del banco. Los datos se almacenan en el sitio donde se usan con más frecuencia, pero también están disponibles por medio de la red de comunicaciones para los usuarios de otros lugares.

El objetivo básico de un sistema distribuido es que el usuario lo perciba como un sistema centralizado. Esto quiere decir, que el usuario no necesita saber en donde se encuentran físicamente almacenados los datos, ya que las aplicaciones son independientes de la manera como se distribuyen los datos, lo que permite cambiar la distribución sin afectar esas aplicaciones(figura 8.1).

Existe cierto numero de situaciones en que el empleo de un procesador central no es la mejor solución. Las razones para el procesamiento distribuido son :

Funcionales.- Un sistema realiza cierto número de funciones diferentes, y es conveniente desarrollar y manejar estas funciones por separado.

Geográficas.- Si existen grupos de usuarios en diferentes lugares, puede resultar eficiente realizar una parte importante del procesamiento local en un procesador local.

Autonomía.- Si en una gran organización se ha delegado la responsabilidad para áreas de negocios o científicas; esto puede resolverse mediante la distribución de los datos, de manera que los privilegios de acceso, las responsabilidades de control de calidad, la confiabilidad de comunicaciones y la conveniencia de acceso correspondan al patrón de responsabilidad.

Confiabilidad.- Si puede aislarse parte de un sistema de datos, mucho del sistema puede continuar funcionando mientras un procesador o sus dispositivos de almacenamiento están descompuestos.

Crecimiento.- Si aumentan las demandas para un sistema de datos, agregar un nuevo nodo puede proporcionar capacidad adicional sin reemplazar todo el sistema anterior.

Redes.- A fin de permitir que múltiples procesadores se integren en un sistema, es necesario que exista una red de comunicacion entre los nodos.

Cualquier forma de compartir recursos implica un gran esfuerzo de operación y manejo. Los sistemas con capacidad local y distribuida resultan de gran interés para los diseñadores de sistemas.

Un sistema distribuido tendrá datos y tareas de procesamiento que son principalmente de importancia local y que se ejecutan en una computadora local. Los requerimientos de datos y procesamientos que no puedan cubrirse localmente generarán demandas a computadoras remotas.

Una computadora lógicamente remota puede ser otra computadora local o un dispositivo especializado unido a una computadora central.

Los beneficios de tener múltiples procesadores, desde el punto de vista de la base de datos, son mayor capacidad de cómputo y de respaldo en caso de que falle el procesador.

El determinar hasta qué punto es mas conveniente la distribución que la centralización depende del costo de manejo de operaciones, comunicaciones y procesamiento. Para reducir el costo operativo y de comunicaciones de bases distribuidas de datos es posible que el equipo real de computación sea compartido. Una base distribuida de datos no implica necesariamente distribución física, sino más bien una distribución de responsabilidades a múltiples bases de datos.

Cada base de datos en el conjunto distribuido tendrá sus conexiones internas y algunas con otros sitios. Las relaciones y conexiones disponibles en un sitio pueden describirse mediante un submodelo de base de datos.

Si una base de datos que opera en un sitio tiene derecho de acceso a datos provenientes de bases ubicadas en otros puntos, puede ser conveniente tener disponible una copia en cada sitio del modelo global de base de datos.

La comunicación de datos voluminosos entre bases de datos distintas es mas difícil que dentro de una base de datos, por lo que existen dos características relacionadas con la implantación : mensajes que se utilizan para establecer comunicación entre los subsistemas y la realización de *réplicas* de relaciones.

La mayor parte del trabajo referente a la integridad de bases distribuidas de datos se basa en la realización de réplicas de relaciones. Las operaciones de recuperación referentes a datos para los que hayan realizado réplicas puede hacerse localmente permitiendo un rápido procesamiento de consultas.

Si es posible, una base de datos deberá designarse como base primaria., es decir, todas las actualizaciones se realizan ahí y, posteriormente, se envían mensajes de actualización a las bases que contengan conexiones de identidad.

En caso de que lo anterior no sea posible, el tráfico de mensajes necesario para mantener la integridad aumenta muchísimo. En algunos casos es posible el intercambio periódico de la responsabilidad primaria.

Un ejemplo de esto se presenta en los bancos, donde durante las horas de negocios, las bases de datos primarias se encuentran en las sucursales. Después del cierre diario, la cosrrespondencia de los datos locales con los datos de la oficina central se verifica y se da la responsabilidad primaria al sitio central. Durante la noche, la base de datos central puede actualizarse rápidamente con transacciones que llegan de otros bancos a la oficina central. Los mensajes de actualización se comunican a las sucursales. En la mañana la responsabilidad pasa a las sucurslaes, después de una verificación de integridad.

Para optimizar las transacciones en las bases de datos distribuidas es necesario considerar los siguientes puntos :

Fuentes.- ¿Qué sitios contienen los elementos datos solicitados?

Tamaño de los segmentos de datos.- ¿Cuál es el resultado parcial esperado o el tamaño del segmento de datos para la subtransacción?

Capacidad de recuperación.- ¿Cuáles son la velocidad y el costo de recuperación en esos sitios?

Capacidad de comunicación.- ¿Cuál es la capacidad de los diferentes sitios para realizar cualquier procesamiento solicitado?

Sitio del resultado.- ¿La salida final se requiere en el sitio de la consulta, en otro sitio o en cualquier parte de la red?

Los analistas utilizan cuatro tipos diferentes de diseño de red para sistemas distribuidos:

1.-Punto a Punto.-

En los ambientes distribuidos, los sistemas punto a punto se unen entre sí como una red; varias localidades interconectadas tienen la capacidad de comunicarse con las demás. En la localidad, cada nodo tiene equipo que transmite o recibe datos. Algunas pueden almacenar y procesar datos y otras no. Solo una localidad puede enviar datos al mismo tiempo, de manera que las otras deben esperar hasta que el envío se termine, sin embargo, todas las localidades pueden recibir los datos en forma sinultánea.

Muchos nodos pueden compartir una sola línea de comunicación, dependiendo de la cantidad de uso de la línea se pueden interconectar 20 o 30 localidades si es necesario.

2.-Redes Jerárquicas.-

Estos sistemas están configurados para tener múltiples niveles de sistemas interconectados entre sí. Un gran sistema se interconecta con otro sistemas o terminales adicionales y éstos a su vez, están conectados con otros sistemas.

En una red jerárquica las comunicaciones *no* se llevan a cabo entre sistemas del mismo nivel; la transmisión de datos ocurre entre sistemas a diferentes niveles.

Este tipo de sistema distribuído es el más común en los negocios; se utiliza con frecuencia en empresas donde las oficinas principales se comunican con las regionales, éstas, a su vez, interactúan con las del distrito, que también se comunican a través de las terminales en las oficinas de campo.

3.-Redes Tipo Estrella.-

Cada terminal, computadora pequeña o grande, puede comunicarse sólo con el sitio central y no con otros nodos dentro de la red. Para transmitir información de un nodo a otro, primero se envían los detalles al nodo central y éste a su vez los enviará al destino final.

Un ejemplo de este tipo es el sistema telefónico : Cuando se marca un número se envían datos a la central telefónica local, que a su vez, envía el mensaje a otra central y a la persona a la que se desea llamar. La comunicación directa sin pasar a través de la central telefónica es imposible.

4.-Redes Tipo Anillo (Tipo Ciclo).-

Permite la comunicación directa entre los nodos y la computadora central. Si el sistema telefónico utiliza una red tipo anillo la comunicación, ya sea a través de la central telefónica o directamente a la persona a la que se desee hablar, sería posible.

8.1.1. RECUPERACION EN SISTEMAS DISTRIBUIDOS

En los sistemas distribuidos es necesario considerar las fallas de comunicación.-

Operación en todo el sistema.- Es posible que haya fallado una parte del sistema distribuido. En algunos nodos pueden seguirse presentando transacciones y es posible que algunas puedan concluírse, en tanto que otras no.

Conocimiento de falla.- Puede transmitirse un bloque a través de una linea de comunicación para escribirse o para un archivo remoto. La transacción que envía puede no saber durante un largo tiempo si el bloque se ha almacenado en forma correcta y segura.

La mayor actividad en paralelo aumenta los riesgos de falla en las bases distribuidas de datos.

Un motivo para la distribución es la autonomía de operación de los diferentes nodos. tener réplicas de los datos permite que se continúe con ciertas operaciones cuando un nodo haya fallado. La independencia de las operaciones de lectura se aumenta al tener datos con réplicas, pero las operaciones de actualización a datos con réplicas en nodos inhabilitados están bloqueadas.

Cualesquiera subtransacciones para actualizar nodos remotos inaccesibles pueden conservarse en una bitácora local de transacciones. Cuando de nuevo estén disponibles los sitios remotos, la transacción puede concluírse.

Si en alguna parte del sistema distribuido se conservan réplicas de todos los fragmentos de datos, una nueva base de datos local puede recuperarse a partir de esas copias.

Una vez que se restaura el nodo dañado, las transacciones y subtransacciones que no pudieran concluírse en su totalidad debido al daño de la red deberán volverse a iniciar para lograr consistencia a través de la base de datos.

Aún cuando la restauración de una base de datos distribuida parece ser compleja, una asignación cuidadosa de los privilegios de actualización puede reducir mucho el grado de interdependencia de los nodos, de manera que sólo puedan ocurrir unos cuantos o tal vez ningún conflicto de actualización.

Las líneas de comunicación de datos es un método común de transmisión de datos y utilizan ya sea líneas telefónicas conmutadas o rentadas que se conectan al equipo de cómputo a través de modems.

Los dos métodos de transmisión de datos son el *síncrono*, que se controla por un reloj, y el *asíncrono*, en el que las señales de inicio y final se envían junto con los datos.

Otros métodos de transmisión de datos utilizan equipos de satélites o de microondas. Algunas compañías están desarrollando sus propias redes, conocidas como **redes de área local** y las están utilizando para ligar diferentes sistemas de cómputo entre sí.

Una red de área local es una conexión de datos interconectados a través de un cable y diseñada para enviar y recibir datos transmitidos a través de distancias cortas. Una distinción entre las redes de área local y las grandes redes es la distancia.

Se espera que la combinación de un mayor costo de comunicaciones y la caída de los precios del hardware aumentarán el uso de sistemas en línea distribuidos en las compañías de todos tipos.

8.2.-APLICACIONES DE BASES DE DATOS PARA MICROCOMPUTADORAS.

Los primeros sistemas de información que las empresas desarrollan son sistemas de procesamiento de transacciones independientes. Sin embrago, conforme más sistemas se desarrollan y su utilidad para la gerencia se incrementa, a menudo se necesita que algunos estén integrados para permitir compartir la información para más de una aplicación.

Existen muchas aplicaciones para microcomputadoras, generalmente se refieren a nóminas, inventarios, ventas, ordenes de pago, registros de estudiantes, personas o clientes, etc.

Las bases de datos de uso cotidiano siempre se mantienen en algún orden determinado : alfabético, por fecha o quizá por código postal, etc. El aprender a definir las partes significativas de la información constituye un aspecto muy importante en el manejo de una base de datos.

Una vez estructurada la base de datos, se necesitará manejarla o administrarla proporcionándole a la computadora instrucciones precisas.

Administrar o manejar una base de datos involucra las siguientes tareas :

1.-Agregrar nuva información a la base de datos.

2.-Clasificar la base de datos en algún orden significativo.

3.-Buscar algún dato en la base de datos, de acuerdo a algún criterio de información que nos interese.

4.-Imprimir la información deseada de nuestra base de datos para incluírla en informes preformateados.

5.-Modificar la información que contiene la base de datos.

6.-Borrar *data* de la base de datos.

A continuación se mostrará un ejemplo muy completo de un sistema operado por menú. El término "operado por menú" significa que la persona que utilice el sistema necesita ejecutar un sólo programa para tener acceso a todo el sistema. Luego, para realizar las diversas tareas, el usuario simplemente selecciona del menú la opción que desea.

Sistema de Correspondencia.

El objetivo de este sistema es llevar un seguimiento de asuntos y el control de la correspondencia del Departamento de Nóminas en Ferrocarriles Nacionales de México. Las bases de datos fueron diseñadas de tal manera que se tiene acceso de muy distintas maneras a una misma información.(figura 8.2).

El sistema consta de cinco programas encadenados por las opciones del Menú Principal. El sexto programa contiene al Menú Principal.

El Menú Principal será el cerebro de todo el sistema de correspondencia: será lo primero y lo último que verá el usuario en la pantalla. El sistema está realizado en DBASE, el cual es un manejador de bases de datos que resulta muy útil cuando se sabe usar para almacenar, organizar, analizar y recobrar información con una microcomputadora.

El programa que contiene el Menú Principal se llama ACCESYS.PRG, el cual después de pedir una clave de acceso al sistema y validar que sea la correcta, muestra las opciones del Menú Principal (ver Anexo "A").

Como se muestra en los anexos se tienen 6 opciones dentro del Menú Principal, las cinco primeras opciones corresponden a los cinco programas restantes :

1.-Menú de Actualización de Documentos	-	CORRESPO.PRG
2.-Menú de Actualización de Funcionarios	-	CATFUNC.PRG
3.-Menú de Actualización de Dependencias	-	CATDEP.PRG
4.-Menú de Actualización de Instrucciones	-	CATINT.PRG
5.-Menú de Consultas	-	MENUCONS.PRG
6.-Salida del Menú	-	QUIT

La primera opción de este Menú Principal (Menú de Actualización de Documentos) es la más importante ya que es aquí donde se realizan las funciones más comunes en el manejo de bases de datos : (La pantalla del submenú correspondiente de ésta opción se muestra en el Anexo "A") altas, bajas, modificaciones, consultas y listados.

Las tres opciones siguientes se refieren a la actualización de los catálogos de Funcionarios, Dependencias e Instrucciones. Los tres programas correspondientes son muy similares ya que los tres realizan funciones muy parecidas, (según se muestra en el Anexo "A" de pantallas) la única diferencia es que accesan diferentes bases de datos. Las bases de datos utilizadas en el sistema y sus estructuras se muestran en el Anexo "B".

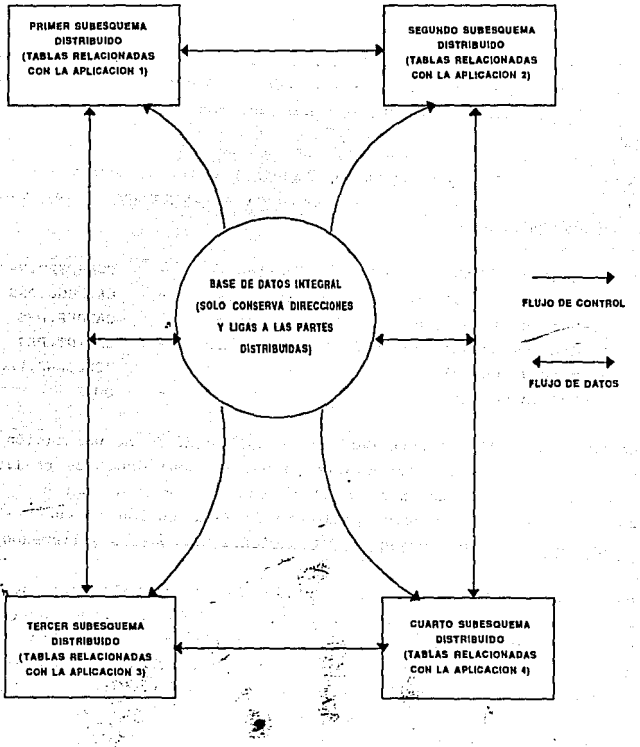


Fig. 8.1 Estructura Distribuida de Bases de Datos.

La quinta opción se refiere a las consultas de la documentación, las cuales pueden ser por dependencia, asunto y fecha límite, además de dar opción para imprimir. (Ver Anexo "A").

La última opción sale del sistema. En el Anexo "C" se encuentran los programas completos del sistema.

Para tener un panorama más general del sistema, en la figura 8.3 se muestra un diagrama que indica la conexión de los programas antes mencionados además de los procedimientos incluídos en cada uno de éstos.

A continuación se mencionan algunas aplicaciones fuera del procesamiento de datos que pueden sacar provecho de la utilización de un sistema de base de datos :

1.-Bases de Datos de Diseño.-

En los sistemas de diseño asistido por computadora (CAD) es preciso almacenar una gran cantidad de datos para representar al objeto que se está diseñando.

2.-Bases de Datos de Conocimientos.-

En los sistemas expertos y de inteligencia artificial, la información se representa como hechos que se expresan en forma lógica. Este conjunto de hechos puede considerarse como una base de datos que contiene conocimientos, es decir, como una base de conocimientos.

3.-Bases de Datos de Multimedios.-

Los datos de naturaleza gráfica pueden almacenarse en una base de datos, y su acceso dependerá de la estructura de un elemento de información gráfica. Los lenguajes de base de datos diseñados para las aplicaciones de procesamiento de datos no son adecuados para tales consultas. Se presentan problemas similares en el caso de datos de audio, diseño y otros tipos de datos que cuentan con una subestructura compleja.

8.2.1. BASES DE DATOS DE DISEÑO

En el diseño asistido por computadora (CAD, Computer Aided Design) se utilizan sistemas de bases de datos para almacenar y manipular información completa referente a un objeto de diseño. El objeto puede ser una pastilla de computadora, un automóvil, un avión, etc.

En general un objeto puede ser cualquier dispositivo o sistema que se diseña utilizando un sistema CAD.

El acceso a la base de datos puede efectuarse a través de lenguajes de consulta (muchas veces basados en lenguajes como el SQL), o en forma indirecta a través de herramientas de diseño de CAD.

A menudo los diseñadores interactuan con la base de datos utilizando una terminal gráfica por medio de la cual mueven o insertan componentes. Los lenguajes basados en texto (es decir, no gráficos) como el SQL no son apropiados para el usuario de la base de datos de diseño.

Uno de los retos al construir un sistema de base de datos de diseño es encontrar un modelo de datos adecuado para representar los datos de diseño, y un lenguaje que permita al diseñador expresar sus consultas y modificaciones de la base de datos en términos de objeto de diseño.

Muchos de los sistemas actuales son extensiones de sistemas de bases de datos comerciales o de investigación ya existentes. Los primeros sistemas de CAD utilizaban sistemas de aplicación especial en los que se representaban los objetos de diseño por medio de un conjunto de archivos. Los inconvenientes principales de este enfoque son la falta de independencia de los datos, lo complejo del manejo de la base de datos y la falta de sistemas de recuperación y de concurrencia que sean verdaderamente generales. Estas desventajas son idénticas a las que se enfrentaron las aplicaciones de procesamiento de datos antes de que se generalizara el uso de sistemas de bases de datos.

Los objetivos tienen una estructura jerárquica (árbol de diseño) basada en niveles de detalle. De manera más general, los objetos pueden tener una estructura de gráfica acíclica dirigida., por ejemplo, un diseñador de computadoras podría operar en los siguientes niveles de detalle :

- Tarjetas
- Pastillas (Chips)
- Diseño interno de las pastillas: celdas, rectángulos, etc.

En algunas aplicaciones de diseño es necesario trabajar sobre un objeto completo. Como ejemplos de este tipo de aplicaciones pueden mencionarse las pruebas, las estimaciones de costos, la simulación y verificación de regla de diseño. Otras aplicaciones pueden involucrar varios objetos en un nivel más bajo de diseño, ejemplos de ello son la modificación de un componente compartido y el diseño de tarjetas nuevas a partir de un conjunto dado de pastillas.

8.2.2. BASES DE CONOCIMIENTO

En la actualidad se están ampliando también los sistemas de base de datos opara que permitan la representación de hechos (reglas) que se expresan de manera lógica, y que pueden utilizarse para responder a consultas que no pueden hacerse en los lenguajes de consulta de base de datos estandar.

Las consultas que se hacen a las bases de conocimientos permiten obtener *metadatos*, es decir, datos acerca de los datos. Esta capacidad se aprovecha en los sistemas expertos, que utilizan tanto los hechos simples que se almacenan como información en la base de datos y las reglas de la base de conocimientos.

Las reglas son proposiciones lógicas y se expresan por lo común en forma de un predicado "Si-entonces". La parte del sistema experto que procesa las reglas interactua con un sistema de base de datos estandar. Este enfoque permite aprovechar la eficiencia de las técnicas de manejo de memoria y procesamiento de consultas de base de datos para manejar la base de conocimientos.

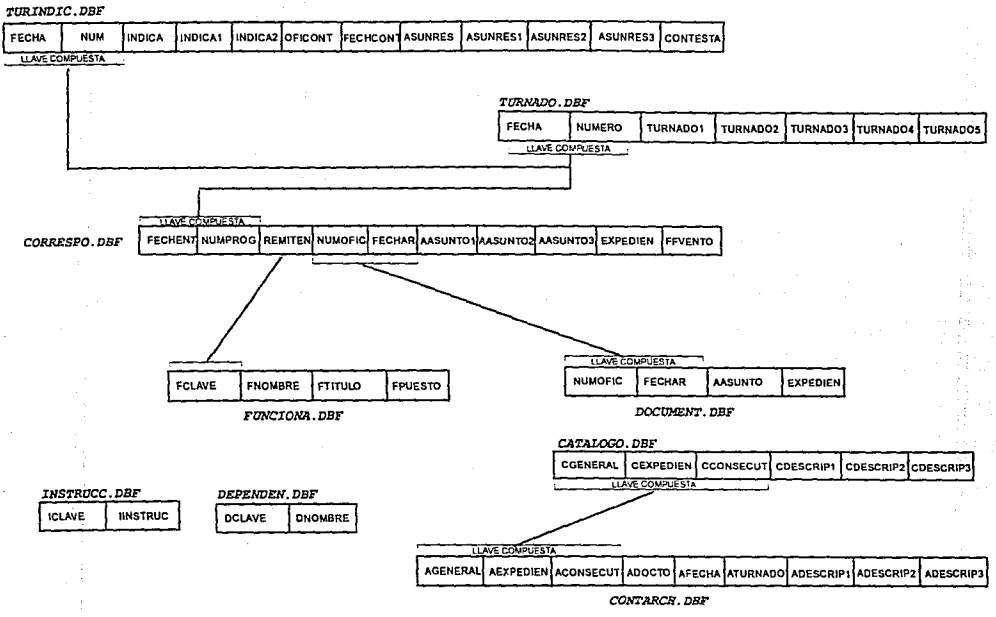


FIG. 8.2.- DISEÑO DE LAS BASES DE DATOS DEL SISTEMA DE CORRESPONDENCIA

CONCLUSIONES

Durante los 70s, existieron tres modelos de organización de bases de datos en la batalla : jerárquicos, de red y relacionales. Pero a finales de los 80s, el modelo relacional había ganado, eliminando completamente a los otros dos.

Ahora, a principios de los 90s, se está detectando que las bases de datos relacionales no proporcionan un modelo suficientemente rico del mundo real y que ellas probablemente también sean reemplazadas en los próximos diez años. El modelo que posiblemente haga a un lado al modelo relacional será uno similar al viejo modelo de red.

Es evidente el gran desarrollo de la microinformática tanto en hardware como en software; se tienen procesadores más rápidos y poderosos, dispositivos de almacenamiento que llegan a los Gigabytes, monitores con altísimas resoluciones, nuevos sistemas operativos, interfaces gráficas (Windows) y muchas más nuevas aplicaciones bajo este ambiente. Sin embargo, existen algunos aspectos de la informática que no se han desarrollado como correspondería.

El enfoque relacional es el modelo sobre el cual se fundamenta el desarrollo de aplicaciones sobre bases de datos. Este enfoque venció a los enfoques jerárquicos y de red convirtiéndose en el estándar utilizado por la gran mayoría de las aplicaciones de manejadores de bases de datos hasta estos días.

Actualmente, las aplicaciones actuales y los usuarios demandan mayor flexibilidad y poder en las estructuras de datos utilizadas. Es necesario un módelo de datos suficientemente poderoso, flexible y sencillo para que todos los usuarios finales puedan definir sus estructuras de datos ya sean simples o complejas : con columnas de longitud variable o columnas multivaluadas, sin tener que preocuparse por conceptos como el establecimiento de relaciones o la normalización de las tablas.

Es obvio, que ésto no es tan fácil pero empieza a ser ya una necesidad de los usuarios de PC.

En un futuro se espera poder explotar bases de datos distribuidas que residan en diferente tipos de ambientes, es decir, una parte podría residir en una PC, otra en un equipo intermedio y lo demás en un *mainframe* y, aunque las tendencias de éstos sistemas están perfectamente trazadas, aún es necesario esperar a que la tecnología permita alcanzar éstos objetivos.

Consideramos que es sumamente importante que los estudiantes de Informática estén relacionados desde los inicios de sus estudios con los conceptos y metodologías del diseño de bases de datos, ya que en cualquier compañía grande o pequeña, las bases de datos son un elemento primordial y común con el que habrán de trabajar.

Por esta razón, nuestra decisión de realizar éste trabajo, en el que se presentó material introductorio sobre conceptos de bases de datos, después se describieron los tres modelos de bases de datos, así como la metodología de diseño y sus aplicaciones.

B I B L I O G R A F I A

KORTH, HENRY
FUNDAMENTOS DE BASES DE DATOS
MEXICO, ED. Mc GRAW HILL, 1988

DATE, C.J.
INTRODUCCION A LOS SISTEMAS DE BASES DE DATOS
MEXICO, ED. ADDISON-WESLEY IBEROAMERICANA, 1989

GIO, WIEDERHOLD
DISEÑO DE BASES DE DATOS
MEXICO, ED. Mc GRAW HILL, 1985

MARTIN, JAMES
ORGANIZACION DE LAS BASES DE DATOS
MEXICO, ED. PRENTICE HALL, 1977

JAMES A. SENN
SISTEMAS DE INFORMACION
MEXICO, ED. Mc GRAW HILL, 1986

TREAMBLAY, JEAN PAUL
ESTRUCTURAS DE DATOS
MEXICO, 1971.

GALINDO SORIA, FERNANDO
APUNTES DE BASES DE DATOS 1 Y 2
MEXICO, UPICCSA

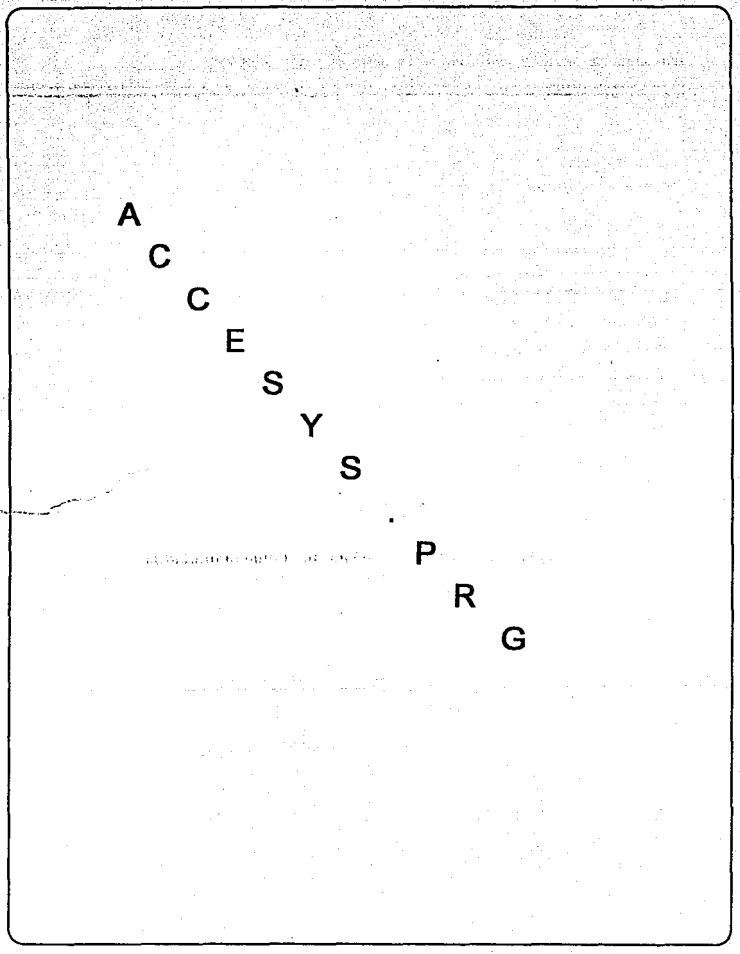
LAURIE, PETER
DATABASES: HOW TO MANAGE INFORMATION
IN YOUR MICRO.
PROVO. UTHA, 1987.

DATE, C.J.
BASES DE DATOS. UNA GUIA PRACTICA
MEXICO, ED. ADDISON-WESLEY IBEROAMERICANA, 1986

BLANCO B., ROBERTO
APUNTES DE PROGRAMACION AVANZADA
MEXICO, ENEP ARAGON 1989.

A N E X O " C "

PROGRAMAS PARA EL SISTEMA DE CORRESPONDENCIA


A
C
C
E
S
Y
S
P
R
G

A C C E S Y S . P R G

```
SET TALK OFF
SET SCOREBOARD OFF
SET STATUS OFF
SET DATE BRITISH
SET DELETE ON
SET INTENSITY OFF
CLEAR
@ 00,01 TO 24,79 DOUBLE
@ 01,22 SAY "FERROCARRILES NACIONALES DE MEXICO"
@ 02,26 SAY "DEPARTAMENTO DE NOMINAS"
@ 04,27 SAY "SISTEMA DE CORRESPONDENCIA"
DO WHILE .T.
   ACCESO = SPACE(04)
   SELECCION =SPACE(01)
   @ 00,01 TO 24,79 DOUBLE
   @ 00,00 CLEAR TO 24,00
   @ 05,02 CLEAR TO 23,78
   @ 10,10 TO 14,70
   @ 11,18 SAY "TECLEE CLAVE DE ACCESO PARA ENTRAR AL SISTEMA"
   @ 01,70 SAY DATE()
   SET COLOR TO N/N
   @ 13,38 GET ACCESO PICTURE "!!!!"
   READ
   SET COLOR TO
   IF ACCESO = "MAIL" THEN
      @ 05,02 CLEAR TO 23,78
      @ 06,30 SAY "MENU PRINCIPAL"
      @ 09,10 SAY "1.- MENU DE ACTUALIZACION DE DOCUMENTOS"
      @ 11,10 SAY "2.- MENU DE ACTUALIZACION DE FUNCIONARIOS"
      @ 13,10 SAY "3.- MENU DE ACTUALIZACION DE DEPENDENCIAS"
      @ 15,10 SAY "4.- MENU DE ACTUALIZACION DE INSTRUCCIONES"
      @ 17,10 SAY "5.- MENU DE CONSULTAS"
      @ 19,10 SAY "6.- SALIDA DEL MENU"
      @ 23,25 SAY "TECLEE LA OPCION DESEADA ["
      @ 23,54 SAY "]"
      @ 01,70 SAY DATE()
      DO WHILE .NOT. SELECCION$("123456")
         @ 23,52 GET SELECCION PICTURE "!"
         READ
      ENDDO
   ELSE
      QUIT
   ENDIF

   DO CASE
            CASE SELECCION = "1"
            CLEAR ALL
            DO CORRESPO
            CASE SELECCION = "2"
            DO CATFUNC
            CASE SELECCION = "3"
            DO CATDEP
            CASE SELECCION = "4"
            DO CATINT
            CASE SELECCION = "5"
            DO MENUCONS
            CASE SELECCION = "6"
```

```
            CLEAR ALL
            CLEAR
            QUIT
            RETURN
ENDCASE
ENDDO
```

```
        DO BAJAS
        CASE RESPUESTA = "03"
        DO CAMBIOS
        CASE RESPUESTA = "04"
        DO CONSUL
        CASE RESPUESTA = "05"
        DO CONSUL2
        CASE RESPUESTA = "06"
        DO FORMCONT
        CASE RESPUESTA = "07"
        DO MODCONT
        CASE RESPUESTA = "08"
        DO CONSCONT
        CASE RESPUESTA = "09"
        DO LISFOR
        CASE RESPUESTA = "10"
        DO CONTES
        CASE RESPUESTA = "11"
        DO NOCONT
        CASE RESPUESTA = "12"
        DO LISRESP
        CASE RESPUESTA = "13"
        DO LISNOR
        CASE RESPUESTA = "14"
        *DO BACKSEM
        CASE RESPUESTA = "15"
        CLEAR ALL
        USE
        RETURN
   ENDCASE
ENDDO


PROCEDURE ALTAS
ENTRADA = DATE()
SELECT A
SET ORDER TO 1
DO WHILE .T.
@ 05,02 CLEAR TO 23,78
@ 05,35 SAY "ALTAS"
     @ 09,10 SAY "FECHA DE ENTRADA        :"
     @ 23,03 SAY "PARA TERMINAR DEJE EN BLANCO EL NUM. PROGRESIVO"
     @ 09,30 GET ENTRADA
             READ
     NUMERO = "  "
     @ 10,10 SAY "NUM. PROGRESIVO"
     @ 10,30 GET NUMERO PICTURE "!!"
             READ
     @ 23,03 SAY "
     IF NUMERO = "  "
        EXIT
     ELSE
             SEEK DTOC(ENTRADA) + NUMERO
     ENDIF
     IF .NOT. EOF()
       @ 23,03 SAY "ESTE NUMERO YA FUE DADO DE ALTA,"
       @ 23,36 SAY XMENSAJE
       SET CONSOLE OFF
       WAIT
       SET CONSOLE ON
```

```
      @ 23,03 SAY XMEN
      LOOP
    ELSE
      DO PANTA1
    ENDIF
ENDDO
RETURN


PROCEDURE PANTA1
REMITE = SPACE(4)
NOFICIO= SPACE(20)
FECHREM = DATE()
FVENTO  = DATE()
TURNA = SPACE(4)
CLDOC = SPACE(2)
INDIC = SPACE(2)
INDIC1 = SPACE(2)
INDIC2 = SPACE(2)
EXPED = SPACE(20)
ASUNTO = SPACE (50)
ASUNTO1 = SPACE(50)
ASUNTO2 = SPACE(50)
ASUNTO3 = SPACE(50)
CONTE = SPACE(1)
DO WHILE .T.
X = 23
Y = 21
Z = 24
W = 27
A = 0
@ 06,02 CLEAR TO 23,78
   @ 09,03 SAY "CLAVE "
   @ 09,25 SAY "["
   @ 09,26 SAY ENTRADA
   @ 09,34 SAY "-"
   @ 09,35 SAY NUMERO
   @ 09,37 SAY "]"
   @ 10,03 SAY "REMITENTE"
   @ 23,03 SAY " TECLEE ESPACIOS EN EL REMITENTE PARA SALIR"
   @ 10,25 GET REMITE
   READ
   @ 23,03 SAY "                                                    "
   IF REMITE = "    "
      EXIT
   ENDIF
   SELECT B
      SEEK REMITE
      IF REMITE = DCLAVE
         @ 10,30 SAY DNOMBRE
         SELECT A
         SET ORDER TO 1
         @ 11,03 SAY "NO. DE OFICIO"
         @ 11,25 SAY "["
         @ 11,46 SAY "]"
         @ 11,26 GET NOFICIO
                 READ
         IF NOFICIO = "                    "
            LOOP
```

```
ENDDO
RETURN
*****************

PROCEDURE RUTINA
@ 23,03 SAY XMEN
@ 23,03 SAY "NO EXISTE CLAVE DE INDICACION,,PRESIONE CUALQUIER TECLA PARA CONTIN
SET CONSOLE OFF
WAIT
SET CONSOLE ON
@ 23,03 SAY XMEN
RETURN

*****************
PROCEDURE BAJAS
SELECT A
SET ORDER TO 1
DO WHILE .T.
@ 05,02  CLEAR TO 23,78
@ 05,35 SAY "BAJAS"
     @ 09,10 SAY "FECHA DE ENTRADA       :"
     @ 23,03 SAY "PARA TERMINAR DEJE EN BLANCO EL NUM. PROGRESIVO"
     @ 09,30 GET ENTRADA
             READ
     NUMERO = "  "
     @ 10,10 SAY "NUM. PROGRESIVO"
     @ 10,30 GET NUMERO PICTURE "!!"
             READ
     @ 23,03 SAY "                                                    "
     IF NUMERO = "  "
        EXIT
     ELSE
        SEEK DTOC(ENTRADA) + NUMERO
     ENDIF
        IF .NOT. EOF()
          IF DELETED()
            @ 23,10 SAY "                                                "
            @ 23,03 SAY "YA SE DIO DE BAJA ESTA CLAVE, "
            @ 23,33 SAY XMENSAJE
            SET CONSOLE OFF
            WAIT
            SET CONSOLE ON
            @ 23,03 SAY XMEN
          ELSE
            DO PANTA2
          ENDIF
        ELSE
          @ 23,10 SAY "                                                  "
          @ 23,03 SAY "ESTE REGISTRO NO SE ENCUENTRA, "
          @ 23,33 SAY XMENSAJE
          SET CONSOLE OFF
          WAIT
          SET CONSOLE ON
          @ 23,10 SAY "                                                  "
          @ 23,03 SAY XMEN
          LOOP
        ENDIF
ENDDO
RETURN
```

```
                    REPLACE FECHAR WITH FECHREM, AASUNTO WITH ASUNTO
                    REPLACE AASUNTO1 WITH ASUNTO1, AASUNTO2 WITH ASUNTO2
                    REPLACE AASUNTO3 WITH ASUNTO3, FFVENTO WITH FVENTO
                    REPLACE EXPEDIEN WITH EXPED
                    reindex
                    SELECT F
                    CONTE = "N"
                    APPEND BLANK
                    REPLACE FECHA WITH ENTRADA, NUM WITH NUMERO, TUR WITH
                    REPLACE INDICA WITH INDIC, INDICA1 WITH INDIC1, INDICA
                    REPLACE CONTESTA WITH CONTE
                    reindex
                    @ 23,03 SAY "                                          "
                    @ 23,03 SAY "SE DIO DE ALTA EL REGISTRO, "
                    @ 23,30 SAY XMENSAJE
                    SET CONSOLE OFF
                    WAIT
                    SET CONSOLE ON
                    RESP = " "
                    @ 23,03 SAY XMEN
                    RETURN
                 ELSE
                    SELECT F
                    GO TOP
                    SEEK DTOC(ENTRADA) + NUMERO
                    DO WHILE dtoc(FECHA) = DTOC(ENTRADA) .AND. NUM = NUMER
                       DELETE
                       SKIP
                    ENDDO
                    @ 23,03 SAY "
                    @ 23,03 SAY "NO SE DIO DE ALTA EL REGISTRO, "
                    @ 23,33 SAY XMENSAJE
                    SET CONSOLE OFF
                    WAIT
                    SET CONSOLE ON
                    @ 23,03 SAY XMEN
                    RETURN
                 ENDIF
              ELSE
                 DO RUTINA
                 LOOP
              ENDIF *INDIC2*
            ELSE
              DO RUTINA
              LOOP
            ENDIF *INDIC1*
          ELSE
            DO RUTINA
            LOOP
          ENDIF *INDIC*
        ENDDO *DO WHILE DEL TURNADO*
      ELSE
        @ 23,03 SAY "
        @ 23,03 SAY "NO SE ENCONTRO ESE REMITENTE, "
        @ 23,34 SAY XMENSAJE
        SET CONSOLE OFF
        WAIT
        SET CONSOLE ON
        @ 23,03 SAY XMEN
      ENDIF *REMITE*
```

```
   ENDIF
   @ 12,03 SAY "FECHA DE REMISION"
   @ 12,25 SAY "["
   @ 12,34 SAY "]"
   @ 12,26 GET FECHREM
           READ
   @ 13,03 SAY "ASUNTO"
   @ 13,25 GET ASUNTO
           READ
   @ 14,25 GET ASUNTO1
           READ
   @ 15,25 GET ASUNTO2
           READ
   @ 16,25 GET ASUNTO3
           READ
   IF ASUNTO = "                              "
      LOOP
   ENDIF
   @ 18,03 SAY "TURNADO A"
DO WHILE .T.
   TURNA =SPACE(4)
   INDIC =SPACE(2)
   INDIC1=SPACE(2)
   INDIC2=SPACE(2)
   SELECT C
   @ 18,X GET TURNA
   READ
   IF TURNA = "     "
      EXIT
   ENDIF
   SEEK TURNA
   IF TURNA # FCLAVE
      @ 23,03 SAY "NO SE ENCONTRO ESE TURNADO, PRESIONE CUALQUIER TECLA PA
      SET CONSOLE OFF
      WAIT
      SET CONSOLE ON
      @ 23,03 SAY XMEN
      LOOP
   ENDIF
   A = A + 1
   DO WHILE .T.
      SELECT E
      @ 20,03 SAY "INDICACION"
      @ 20,Y GET INDIC
      READ
      SEEK INDIC
      IF INDIC = "  "
         LOOP
      ENDIF
      IF INDIC # ICLAVE
         DO RUTINA
         LOOP
      ENDIF
         @ 20,Z GET INDIC1
         READ
         IF INDIC1 = "  "
            EXIT
         ENDIF
         SEEK INDIC1
         IF INDIC1 # ICLAVE
```

```
         DO RUTINA
         LOOP
      ENDIF
         @ 20,W GET INDIC2
         READ
         IF INDIC2 = "  "
            EXIT
         ENDIF
         SEEK INDIC2
         IF INDIC2 # ICLAVE
            DO RUTINA
            LOOP
         ENDIF
            EXIT
ENDDO
          IF A # 5
            @ 23,03 SAY "DESEA PONER MAS TURNADOS? (S/N) "
            @ 23,35 GET RESP PICTURE "!"
            READ
            @ 23,03 SAY XMEN
            IF UPPER(RESP) = "S"
               CONTE = "N"
               SELECT F
               APPEND BLANK
               REPLACE FECHA WITH ENTRADA, NUM WITH NUMERO,TUR WITH TUR
               REPLACE INDICA WITH INDIC, INDICA1 WITH INDIC1, INDICA2
               REPLACE CONTESTA WITH CONTE
               reindex
               @ 23,03 SAY XMEN
               X = X + 10
               Y = Y + 10
               Z = Z + 10
               W = W + 10
               LOOP
             ELSE
             ENDIF
            ELSE
            ENDIF
               @ 21,03 SAY "EXPEDIENTE"
               @ 21,25 GET EXPED
               READ
               IF EXPED = "                    "
                  LOOP
               ENDIF
               @ 22,03 SAY "FECHA VENCIMIENTO"
               @ 22,25 SAY "["
               @ 22,34 SAY "]"
               @ 22,26 GET FVENTO
               READ
               @ 23,03 SAY "
               @ 23,10 SAY "ESTAN CORRECTOS LOS DATOS ? (S/N)"
               @ 23,47 GET RESP PICTURE "!"
               READ
               @ 23,03 SAY XMEN
               IF UPPER(RESP) = "S"
                  SELECT A
                  SET ORDER TO 1
                  APPEND BLANK
                  REPLACE FECHENT WITH ENTRADA, NUMPROG WITH NUMERO
                  REPLACE REMITEN WITH REMITE, NUMOFIC WITH NOFICIO
```

```
                    @ 22,34 SAY "]"
                    @ 22,26 SAY FFVENTO
                    @ 23,10 SAY "
                    @ 23,10 SAY "DESEA BORRAR ESTE REGISTRO ? (S/N)"
                    @ 23,46 GET RESP PICTURE "!"
                            READ
                    @ 23,03 SAY XMEN
                    ENDIF
                       IF UPPER(RESP) = "S"
                          DELETE
                          PACK
                          SELECT F
                          SEEK  DTOC(ENTRADA) + NUMERO
                          DO WHILE DTOC(FECHA) = DTOC(ENTRADA) .AND. NUM = NUMERO
                             DELETE
                             SKIP
                          ENDDO
                          PACK
                          @ 23,03 SAY "                                                  "
                          @ 23,03 SAY "SE DIO DE BAJA EL REGISTRO,"
                          @ 23,31 SAY XMENSAJE
                          SELECT A
                          SET CONSOLE OFF
                          WAIT
                          SET CONSOLE ON
                          @ 23,03 SAY XMEN
                          RESP = " "
                          EXIT
                        ELSE
                          @ 23,03 SAY "                                                  "
                          @ 23,03 SAY "NO SE DIO DE BAJA EL REGISTRO,"
                          @ 23,34 SAY XMENSAJE
                          SET CONSOLE OFF
                          WAIT
                          SET CONSOLE ON
                          @ 23,03 SAY XMEN
                          EXIT
                        ENDIF
ENDDO
RETURN

PROCEDURE CAMBIOS
SELECT A
SET ORDER TO 1
DO WHILE .T.
@ 05,02 CLEAR TO 23,78
@ 05,35 SAY "CAMBIOS"
NUMERO = "  "
     @ 09,10 SAY "FECHA DE ENTRADA        :"
     @ 23,03 SAY "PARA TERMINAR DEJE EN BLANCO EL NUM. PROGRESIVO"
     @ 09,30 GET ENTRADA
             READ
     @ 10,10 SAY "NUM. PROGRESIVO"
     @ 10,30 GET NUMERO PICTURE "!!"
             READ
     @ 23,03 SAY "                                                  "
     IF NUMERO = "  "
        EXIT
     ELSE
        SEEK DTOC(ENTRADA) + NUMERO
```

```
SET ORDER TO 1
@ 11,03 SAY "NO. DE OFICIO"
@ 11,25 SAY "["
@ 11,26 SAY NUMOFIC
@ 11,47 SAY "]"
@ 12,03 SAY "FECHA DE REMISION"
@ 12,25 SAY "["
@ 12,26 SAY FECHAR
@ 12,34 SAY "]"
@ 13,03 SAY "ASUNTO"
@ 13,25 SAY AASUNTO
@ 14,25 SAY AASUNTO1
@ 15,25 SAY AASUNTO2
@ 16,25 SAY AASUNTO3
@ 18,03 SAY "TURNADO A"
@ 20,03 SAY "INDICACION"
SELECT F
LOCATE FOR FECHA = ENTRADA .AND. NUM = NUMERO
DO WHILE FECHA = ENTRADA .AND. NUM = NUMERO
   @ 18,X SAY TUR
   @ 20,Y SAY INDICA
   @ 20,Z SAY INDICA1
   @ 20,W SAY INDICA2
   CONTINUE
   X = X + 10
   Y = Y + 10
   Z = Z + 10
   W = W + 10
   LOOP
ENDDO
SELECT A
SET ORDER TO 1
@ 21,03 SAY "EXPEDIENTE"
@ 21,25 SAY EXPEDIEN
@ 22,03 SAY "FECHA VENCIMIENTO"
@ 22,25 SAY "["
@ 22,34 SAY "]"
@ 22,26 SAY FFVENTO
@ 23,10 SAY "                                                                    "
@ 23,10 SAY "DESEA CAMBIAR ESTE REGISTRO ? (S/N)"
@ 23,48 GET RESP PICTURE "!"
READ
@ 23,03 SAY XMEN
IF UPPER(RESP) = "S"
   DO WHILE .T.
      A = 0
      X = 23
      Y = 21
      Z = 24
      W = 27
      @ 10,25 SAY REMITEN
      @ 00,01 TO 24,79 DOUBLE
      @ 10,25 GET REMITE
      READ
      SELECT B
      SEEK REMITE
      IF REMITE # DCLAVE
        @ 23,03 SAY XMEN
        @ 23,03 SAY "NO SE ENCONTRO ESE REMITENTE,PRESIONE CUALQUIER TECLA PARA
        SET CONSOLE OFF
```

```
                ELSE
                   @ 23,03 SAY XMEN
                   EXIT
                ENDIF
    ENDDO
IF A # 5
 DO WHILE .T.
    IF A = 5
       EXIT
    ENDIF
    @ 23,03 SAY XMEN
    @ 23,03 SAY "DESEA PONER MAS TURNADOS? (S/N) "
    @ 23,36 GET RESP PICTURE "!"
    READ
    @ 23,03 SAY XMEN
    IF UPPER(RESP) = "S"
       X = X + 10
       Y = Y + 10
       Z = Z + 10
       W = W + 10
      DO WHILE .T.
       SELECT C
       TURNA = "    "
       @ 18,X GET TURNA
       READ
       IF TURNA = "    "
          EXIT
       ENDIF
       SEEK TURNA
       IF TURNA # FCLAVE
          @ 23,03 SAY "NO SE ENCONTRO ESE TURNADO, PRESIONE CUALQUIER TECLA
          SET CONSOLE OFF
          WAIT
          SET CONSOLE ON
          @ 23,03 SAY XMEN
          LOOP
       ELSE
       ENDIF
       A = A + 1
       DO WHILE .T.
          SELECT E
          INDIC = "  "
          @ 20,Y GET INDIC
          READ
          IF INDIC = "  "
             LOOP
          ENDIF
          SEEK INDIC
          IF INDIC # ICLAVE
             DO RUTINA
             LOOP
          ENDIF
          EXIT
       ENDDO
       DO WHILE .T.
          SELECT E
          INDIC1 = "  "
          @ 20,Z GET INDIC1
          READ
          IF INDIC1 = "  "
```

```
            EXIT
         ENDIF
         SEEK INDIC1
         IF INDIC1 # ICLAVE
            DO RUTINA
            LOOP
         ENDIF
         EXIT
      ENDDO
      DO WHILE .T.
         SELECT E
         INDIC2 = "   "
         @ 20,W GET INDIC2
         READ
         IF INDIC2 = "   "
            EXIT
         ENDIF
         SEEK INDIC2
         IF INDIC2 # ICLAVE
            DO RUTINA
            LOOP
         ENDIF
         EXIT
      ENDDO
         SELECT F
         APPEND BLANK
         REPLACE FECHA WITH ENTRADA, NUM WITH NUMERO,TUR WITH TURNA
         REPLACE INDICA WITH INDIC, INDICA1 WITH INDIC1, INDICA2 WITH INDI
         @ 23,03 SAY XMEN
         EXIT
    ENDDO
  ELSE
    EXIT
  ENDIF
  LOOP
ENDDO
ELSE
ENDIF
 SELECT A
 EXPED = EXPEDIEN
 @ 21,25 SAY EXPEDIEN
 @ 21,25 GET EXPED
 READ
 FVENTO = FFVENTO
 @ 22,26 SAY FFVENTO
 @ 22,26 GET FVENTO
 READ
 @ 23,10 SAY "                                                    "
 @ 23,10 SAY "ESTAN CORRECTOS LOS DATOS  ? (S/N)"
 @ 23,47 GET RESP PICTURE "!"
 READ
 @ 23,03 SAY XMEN
 IF UPPER(RESP) = "S"
    SELECT A
    REPLACE FECHENT WITH ENTRADA, NUMPROG WITH NUMERO
    REPLACE REMITEN WITH REMITE, NUMOFIC WITH NOFICIO
    REPLACE FECHAR WITH FECHREM, AASUNTO WITH ASUNTO
    REPLACE AASUNTO1 WITH ASUNTO1, AASUNTO2 WITH ASUNTO2
    REPLACE AASUNTO3 WITH ASUNTO3, FFVENTO  WITH FVENTO
    REPLACE EXPEDIEN WITH EXPED
```

```
         @ 23,03 SAY "SE REALIZARON LOS CAMBIOS "
         @ 23,30 SAY XMENSAJE
         SET CONSOLE OFF
         WAIT
         SET CONSOLE ON
         RESP = " "
         @ 23,03 SAY XMEN
         EXIT
      ELSE
         SELECT F
         GO TOP
         SEEK DTOC(ENTRADA) + NUMERO
         DO WHILE FECHA = ENTRADA .AND. NUM = NUMERO
            DELETE
            SKIP
         ENDDO
         @ 23,03 SAY "NO SE REALIZO NINGUN CAMBIO, "
         @ 23,32 SAY XMENSAJE
         SET CONSOLE OFF
         WAIT
         SET CONSOLE ON
         @ 23,03 SAY XMEN
         EXIT
      ENDIF
 ENDDO
ELSE
   @ 23,01 SAY "                                                                    "
   @ 23,03 SAY "NO SE REALIZO NINGUN CAMBIO, "
   @ 23,34 SAY XMENSAJE
   SET CONSOLE OFF
   WAIT
   SET CONSOLE ON
   @ 23,03 SAY XMEN
ENDIF
RETURN


PROCEDURE CONSUL
SELECT A
SET ORDER TO 1
RESP = "S"
DO WHILE .T.
IF UPPER(RESP) =  "S"
@ 05,02 CLEAR TO 23,78
@ 05,35 SAY "CONSULTAS"
     @ 09,10 SAY "FECHA DE ENTRADA       :"
     @ 23,03 SAY "PARA TERMINAR DEJE EN BLANCO EL NUM. PROGRESIVO"
     @ 09,30 GET ENTRADA
             READ
     NUMERO = "  "
     @ 10,10 SAY "NUM. PROGRESIVO"
     @ 10,30 GET NUMERO PICTURE "!!"
             READ
     @ 23,03 SAY "
     IF NUMERO = "  "
        EXIT
     ELSE
        SEEK DTOC(ENTRADA) + NUMERO
     ENDIF
        IF .NOT. EOF()
```

```
        IF DELETED()
          @ 23,03 SAY "                                                    "
          @ 23,03 SAY "YA SE DIO DE BAJA ESTA CLAVE, "
          @ 23,33 SAY XMENSAJE
          SET CONSOLE OFF
          WAIT
          SET CONSOLE ON
          @ 23,03 SAY XMEN
        ELSE
          DO PAN4
        ENDIF
      ELSE
        @ 23,10 SAY "                                                    "
        @ 23,03 SAY "ESTE REGISTRO NO SE ENCUENTRA, "
        @ 23,33 SAY XMENSAJE
        SET CONSOLE OFF
        WAIT
        SET CONSOLE ON
        @ 23,10 SAY XMEN
        @ 23,03 SAY XMEN
        LOOP
      ENDIF
 ELSE
  EXIT
 ENDIF
NDDO
ETURN

ROCEDURE PAN4
EMITE = SPACE(4)
URNA = SPACE(4)
 = 23
 = 21
 = 24
 = 27
O WHILE .T.
 06,02 CLEAR TO 23,78
  @ 09,03 SAY "CLAVE "
  @ 09,25 SAY "["
  @ 09,26 SAY ENTRADA
  @ 09,34 SAY "-"
  @ 09,35 SAY NUMERO
  @ 09,37 SAY "]"
  @ 10,03 SAY "REMITENTE"
  @ 10,25 SAY REMITEN
  REMITE = REMITEN
  SELECT B
     SEEK REMITE
     IF REMITE = DCLAVE
        @ 10,30 SAY DNOMBRE
        SELECT A
        SET ORDER TO 1
        @ 11,03 SAY "NO. DE OFICIO"
        @ 11,25 SAY "["
        @ 11,26 SAY NUMOFIC
        @ 11,47 SAY "]"
        @ 12,03 SAY "FECHA DE REMISION"
        @ 12,25 SAY "["
        @ 12,26 SAY FECHAR
        @ 12,47 SAY "]"
```

```
          @ 13,03 SAY "ASUNTO"
          @ 13,25 SAY AASUNTO
          @ 14,25 SAY AASUNTO1
          @ 15,25 SAY AASUNTO2
          @ 16,25 SAY AASUNTO3
          @ 18,03 SAY "TURNADO A"
          @ 20,03 SAY "INDICACION"
          SELECT F
          LOCATE FOR FECHA = ENTRADA .AND. NUM = NUMERO
          DO WHILE .NOT. EOF()
               @ 18,X SAY TUR
               @ 20,Y SAY INDICA
               @ 20,Z SAY INDICA1
               @ 20,W SAY INDICA2
               CONTINUE
               X = X + 10
               Y = Y + 10
               Z = Z + 10
               W = W + 10
               LOOP
          ENDDO
                  SELECT A
                  SET ORDER TO 1
                  @ 21,03 SAY "EXPEDIENTE"
                  @ 21,25 SAY EXPEDIEN
                  @ 22,03 SAY "FECHA VENCIMIENTO"
                  @ 22,25 SAY FFVENTO
                  @ 23,10 SAY "
                  @ 23,10 SAY "DESEA REALIZAR MAS CONSULTAS ? (S/N)"
                  @ 23,47 GET RESP PICTURE "!"
                          READ
                  @ 23,03 SAY XMEN
                     IF UPPER(RESP) = "S"
                       EXIT
                     ELSE
                       RETURN
                     ENDIF
            ELSE
              @ 23,25 SAY "
              @ 23,25 SAY "NO EXISTE CLAVE DE TURNADO"
            ENDIF
       ELSE
         @ 23,25 SAY "                                          "
         @ 23,25 SAY "NO SE ENCONTRO ESE REMITENTE"
       ENDIF
ENDDO
RETURN

PROCEDURE CONSUL2
N = SPACE(2)
SELECT A
GO TOP
RESP = "N"
DO WHILE .T.
X = 11
@ 05,02 CLEAR TO 23,78
@ 05,35 SAY "CONSULTAS2"
@ 07,10 SAY "FECHA DE ENTRADA       :"
@ 23,03 SAY "TECLEE UNOS EN LA FECHA PARA SALIR"
@ 07,30 GET ENTRADA
```

```
    READ
IF ENTRADA = CTOD("11/11/11")
   EXIT
ELSE
 LOCATE FOR FECHENT = ENTRADA .OR. FECHENT > ENTRADA
 @ 09,02 SAY "REMITE"
 @ 09,11 SAY "NUM. DE OFICIO"
 @ 09,27 SAY "REMISION"
 @ 09,54 SAY "NO."
 @ 09,36 SAY "TURNADO"
 @ 09,63 SAY "EXPTE."
 @ 09,75 SAY "CONT"
 @ 09,44 SAY "ENTRADA"
      IF EOF()
         @ 20,03 SAY "NO EXISTE NINGUNA CON ESA FECHA"
         @ 23,03 SAY "DESEA SEGUIR CONSULTANDO ?(S/N)"
         @ 23,45 GET RESP
         READ
         IF UPPER(RESP) = "S"
            LOOP
          ELSE
            EXIT
         ENDIF
      ENDIF
     DO WHILE .NOT. EOF()
        @ X,09 SAY NUMOFIC
        @ X,27 SAY FECHAR
        @ X,03 SAY REMITEN
        @ X,54 SAY NUMPROG
        @ X,59 SAY EXPEDIEN
        @ X,44 SAY FECHENT
        E = FECHENT
        N = NUMPROG
        SELECT F
        SEEK DTOC(E) + N
        DO WHILE E = FECHA .AND. NUM = N
           @ X,38 SAY TUR
           @ X,76 SAY CONTESTA
           X = X + 1
           SKIP
           IF NUM # N
              EXIT
           ENDIF
        ENDDO
        SELECT A
        CONTINUE
        X = X + 1
        IF X > 20
           @ 23,03 SAY "
           @ 23,20 SAY "DESEA MAS INFORMACION (S/N) ? "
           @ 23,55 GET RESP
           READ
           IF UPPER(RESP) = "S"
             @ 11,02 CLEAR TO 23,78
             X = 11
             LOOP
           ELSE
             EXIT
           ENDIF
         ELSE
```

```
            LOOP
            ENDIF
         ENDDO
            @ 23,03 SAY "
            @ 23,03 SAY "
            @ 23,03 SAY "DESEA SEGUIR CONSULTANDO ?(S/N)"
            @ 23,45 GET RESP
            READ
            @ 23,03 SAY "
            @ 23,03 SAY "
            IF UPPER(RESP) = "S"
              LOOP
            ELSE
              EXIT
            ENDIF
    ENDIF
ENDDO
RETURN

PROCEDURE FORMCONT
DO WHILE .T.
NOFICON = SPACE(20)
FEREPCON = DATE()
ASUNCON = SPACE(50)
ASUNCON1 = SPACE(50)
ASUNCON2 = SPACE(50)
ASUNCON3 = SPACE(50)
TURNA = SPACE(4)
CONTE = " "
NUMERO = SPACE(2)
@ 05,02 CLEAR TO 23,78
@ 05,25 SAY "REGISTRO DE LA CONTESTACION"
@ 07,03 SAY "FECHA DE ENTRADA :"
@ 23,03 SAY "PARA TERMINAR DEJE EN BLANCO EL NUM. PROGRESIVO"
@ 07,22 GET ENTRADA
        READ
@ 08,03 SAY "NUM. PROGRESIVO  :"
@ 08,22 GET NUMERO PICTURE "!!"
        READ
  IF NUMERO = "  "
     EXIT
  ELSE
     @ 09,03 SAY "CVE. FUNCIONARIO :"
     @ 09,22 GET TURNA PICTURE "!!!!"
             READ
     IF TURNA = "    "
        EXIT
     ELSE
        SELECT C
        SEEK TURNA
        IF EOF()
           FFNOMBRE = "SIN NOMBRE"
        ELSE
           FFNOMBRE = FNOMBRE
        ENDIF
     ENDIF
     SELECT F
     SET ORDER TO 3
     SEEK DTOC(ENTRADA) + NUMERO + TURNA
     IF .NOT. EOF()
```

```
                @ 23,03 SAY "SE DIO DE ALTA EL REGISTRO, "
                @ 23,32 SAY XMENSAJE
                SET CONSOLE OFF
                WAIT
                SET CONSOLE ON
                @ 23,03 SAY XMEN
                RESP = " "
                LOOP
             ELSE
                @ 23,03 SAY "                                                    "
                @ 23,03 SAY "NO SE DIO DE ALTA EL REGISTRO, "
                @ 23,34 SAY XMENSAJE
                SET CONSOLE OFF
                WAIT
                SET CONSOLE ON
                @ 23,03 SAY XMEN
                LOOP
                ENDIF
      ELSE
        @ 23,03  SAY "                                                    "
        @ 23,03  SAY "YA SE CONTESTO ESTE REGISTRO, "
        @ 23,34  SAY XMENSAJE
        SET CONSOLE OFF
        WAIT
        SET CONSOLE ON
        @ 23,03 SAY XMEN
        LOOP
      ENDIF
     ELSE
        @ 22,10 SAY "                                                    "
        @ 23,03 SAY "NO SE ENCONTRO DICHO REGISTRO, "
        @ 23,34 SAY XMENSAJE
        SET CONSOLE OFF
        WAIT
        SET CONSOLE ON
        @ 23,03 SAY XMEN
        LOOP
       ENDIF
    ENDIF
ENDDO


PROCEDURE MODCONT
NOFICON = SPACE(20)
FEREPCON = DATE()
ASUNCON = SPACE(50)
ASUNCON1 = SPACE(50)
ASUNCON2 = SPACE(50)
ASUNCON3 = SPACE(50)
TURNA = SPACE(04)
CONTE = " "
DO WHILE .T.
@ 05,02 CLEAR TO 23,78
NUMERO = SPACE(2)
@ 05,25 SAY "CAMBIO  DE  LA CONTESTACION"
@ 07,03 SAY "FECHA DE ENTRADA :"
@ 23,03 SAY "PARA TERMINAR DEJE EN BLANCO EL NUM. PROGRESIVO"
@ 07,22 GET ENTRADA
        READ
@ 08,03 SAY "NUM. PROGRESIVO  :"
```

```
IF CONTESTA # "S"
   SELECT A
   SEEK DTOC(ENTRADA) + NUMERO
   IF EOF()
      @ 23,03 SAY XMEN
      @ 23,03 SAY "ERROR EN ARCHIVO DE CORRESPO, "
      @ 23,34 SAY XMENSAJE
      SET CONSOLE OFF
      WAIT
      SET CONSOLE ON
      EXIT
   ELSE
   ENDIF
@ 09,02 CLEAR TO 23,78
@ 07,42 SAY "NO. OFICIO:"
@ 07,54 SAY NUMOFIC
@ 08,42 SAY "REMITENTE : "
@ 08,54 SAY REMITEN
@ 09,03 SAY "ASUNTO                :"
@ 09,25 SAY AASUNTO
@ 10,25 SAY AASUNTO1
@ 11,25 SAY AASUNTO2
@ 12,25 SAY AASUNTO3
@ 13,03 SAY "FUNCIONARIO :"
@ 13,25 SAY FFNOMBRE
SELECT F
@ 14,03 SAY "INDICACIONES:"
@ 14,25 SAY INDICA
@ 14,28 SAY INDICA1
@ 14,31 SAY INDICA2
@ 15,03 SAY "------------------------------------------------------------
@ 16,03 SAY "NO. DE OFICIO (RESP):"
@ 16,25 GET NOFICON
        READ
@ 17,03 SAY "FECHA DE RESPUESTA  :"
@ 17,25 GET FEREPCON
        READ
@ 18,03 SAY "RESPUESTA             :"
DO WHILE ASUNCON = "                "
   @ 18,25 GET ASUNCON
           READ
ENDDO
@ 19,25 GET ASUNCON1
        READ
@ 20,25 GET ASUNCON2
        READ
@ 21,25 GET ASUNCON3
        READ
@ 23,03 SAY "
@ 23,10 SAY "ESTAN CORRECTOS LOS DATOS ? (S/N)"
@ 23,47 GET RESP PICTURE "!"
        READ
@ 23,03 SAY XMEN
        IF UPPER(RESP) = "S"
           CONTE = "S"
           REPLACE OFICONT WITH NOFICON, FECHCONT WITH FEREPCON
           REPLACE ASUNRES WITH ASUNCON, ASUNRES1 WITH ASUNCON1
           REPLACE ASUNRES2 WITH ASUNCON2, ASUNRES3 WITH ASUNCON3
           REPLACE CONTESTA WITH CONTE
           @ 23,03 SAY "                                          "
```

```
@ 08,22 GET NUMERO PICTURE "!!"
      READ
  IF NUMERO = "  "
     EXIT
  ELSE
     @ 09,03 SAY "CVE. FUNCIONARIO:"
     @ 09,22 GET TURNA
           READ
     IF TURNA = "    "
        EXIT
     ELSE
        SELECT C
        SEEK TURNA
        IF EOF()
           FFNOMBRE = "SIN NOMBRE"
        ELSE
           FFNOMBRE = FNOMBRE
        ENDIF
        SELECT F
        SET ORDER TO 3
     SEEK DTOC(ENTRADA) + NUMERO + TURNA
     IF .NOT. EOF()
      IF CONTESTA = "S"
         SELECT A
         SEEK DTOC(ENTRADA) + NUMERO
         IF EOF()
            @ 23,03 SAY XMEN
            @ 23,03 SAY "ERROR EN ARCHIVO DE CORRESPO, "
            @ 23,34 SAY XMENSAJE
            SET CONSOLE OFF
            WAIT
            SET CONSOLE ON
            EXIT
         ELSE
         ENDIF
       @ 09,02 CLEAR TO 23,78
       @ 07,42 SAY "NO. OFICIO:"
       @ 07,54 SAY NUMOFIC
       @ 08,42 SAY "REMITENTE : "
       @ 08,54 SAY REMITEN
       @ 09,03 SAY "ASUNTO                :"
       @ 09,25 SAY AASUNTO
       @ 10,25 SAY AASUNTO1
       @ 11,25 SAY AASUNTO2
       @ 12,25 SAY AASUNTO3
       @ 13,03 SAY "FUNCIONARIO :"
       @ 13,25 SAY FFNOMBRE
       SELECT F
       @ 14,03 SAY "INDICACIONES:"
       @ 14,25 SAY INDICA
       @ 14,28 SAY INDICA1
       @ 14,31 SAY INDICA2
       @ 15,03 SAY "-------------------------------------------------------------------
       @ 16,03 SAY "NO. DE OFICIO (RESP):"
       @ 16,25 SAY OFICONT
       @ 17,03 SAY "FECHA DE RESPUESTA  :"
       @ 17,25 SAY FECHCONT
       @ 18,03 SAY "RESPUESTA             :"
       @ 18,25 SAY ASUNRES
       @ 19,25 SAY ASUNRES1
```

```
@ 20,25 SAY ASUNRES2
@ 21,25 SAY ASUNRES3
@ 23,03 SAY "                                                                "
@ 23,10 SAY "DESEA CAMBIAR  LOS  DATOS ? (S/N)"
@ 23,47 GET RESP PICTURE "!"
        READ
IF RESP = "S"
 NOFICON = OFICONT
 @ 16,25 SAY OFICONT
 @ 16,25 GET NOFICON
         READ
 FEREPCON = FECHCONT
 @ 17,25 SAY FECHCONT
 @ 17,25 GET FEREPCON
         READ
 @ 18,03 SAY "RESPUESTA           :"
 DO WHILE ASUNCON = "          "
    ASUNCON = ASUNRES
    @ 18,25 SAY ASUNRES
    @ 18,25 GET ASUNCON
            READ
 ENDDO
 ASUNCON1 = ASUNRES1
 @ 19,25 SAY ASUNRES1
 @ 19,25 GET ASUNCON1
         READ
 ASUNCON2 = ASUNRES2
 @ 20,25 SAY ASUNRES2
 @ 20,25 GET ASUNCON2
         READ
 ASUNCON3 = ASUNRES3
 @ 21,25 SAY ASUNRES3
 @ 21,25 GET ASUNCON3
         READ
 @ 23,03 SAY "                                                                "
 @ 23,10 SAY "ESTAN CORRECTOS LOS DATOS ? (S/N)"
 @ 23,47 GET RESP PICTURE "!"
         READ
 @ 23,03 SAY XMEN
        IF UPPER(RESP) = "S"
           CONTE = "S"
           REPLACE OFICONT WITH NOFICON, FECHCONT WITH FEREPCON
           REPLACE ASUNRES WITH ASUNCON, ASUNRES1 WITH ASUNCON1
           REPLACE ASUNRES2 WITH ASUNCON2, ASUNRES3 WITH ASUNCON3
           REPLACE CONTESTA WITH CONTE
           @ 23,03 SAY "                                                    "
           @ 23,03 SAY "SE MODIFICO  EL  REGISTRO, "
           @ 23,32 SAY XMENSAJE
           SET CONSOLE OFF
           WAIT
           SET CONSOLE ON
           @ 23,03 SAY XMEN
           RESP = " "
           LOOP
        ELSE
           @ 23,03 SAY "                                                    "
           @ 23,03 SAY "NO SE DIO DE ALTA EL REGISTRO, "
           @ 23,34 SAY XMENSAJE
           SET CONSOLE OFF
           WAIT
```

```
                SET CONSOLE ON
                @ 23,03 SAY XMEN
                LOOP
                ENDIF
        ELSE
           @ 23,03 SAY "                                                    "
           @ 23,03 SAY "NO SE MODIFICO EL REGISTRO, "
           @ 23,34 SAY XMENSAJE
           SET CONSOLE OFF
           WAIT
           SET CONSOLE ON
           @ 23,03 SAY XMEN
           LOOP
        ENDIF
       ELSE
         @ 23,03  SAY "                                                    "
         @ 23,03  SAY "NO SE HA DADO CONTESTACION , "
         @ 23,34  SAY XMENSAJE
         SET CONSOLE OFF
         WAIT
         SET CONSOLE ON
         @ 23,03 SAY XMEN
         LOOP
       ENDIF
      ELSE                              :
         @ 22,10 SAY "                                                    "
         @ 23,03 SAY "NO SE ENCONTRO DICHO REGISTRO, "
         @ 23,34 SAY XMENSAJE
         SET CONSOLE OFF
         WAIT
         SET CONSOLE ON
         @ 23,03 SAY XMEN
         LOOP
        ENDIF
     ENDIF
ENDDO

PROCEDURE CONSCONT
NOFICON = SPACE(20)
FEREPCON = DATE()
ASUNCON = SPACE(50)
ASUNCON1 = SPACE(50)
ASUNCON2 = SPACE(50)
ASUNCON3 = SPACE(50)
TURNA = SPACE(04)
CONTE = " "
DO WHILE .T.
 @ 05,02 CLEAR TO 23,78
 NUMERO = SPACE(2)
 @ 05,25 SAY "CONSULTA DE  LA CONTESTACION"
 @ 07,03 SAY "FECHA DE ENTRADA :"
 @ 23,03 SAY "PARA TERMINAR DEJE EN BLANCO EL NUM. PROGRESIVO"
 @ 07,22 GET ENTRADA
         READ
 @ 08,03 SAY "NUM. PROGRESIVO  :"
 @ 08,22 GET NUMERO PICTURE "!!"
         READ
  IF NUMERO = "  "
     EXIT
  ENDIF
```

```
 @ 09,03 SAY "CVE. FUNCIONARIO:"
 @ 09,22 GET TURNA
         READ
IF TURNA = "    "
   EXIT
ELSE
    SELECT C
    SEEK TURNA
    IF EOF()
       FFNOMBRE = "SIN NOMBRE"
    ELSE
       FFNOMBRE = FNOMBRE
    ENDIF
 ENDIF
 SELECT F
 SET ORDER TO 3
 SEEK DTOC(ENTRADA) + NUMERO + TURNA
 IF .NOT. EOF()
  IF CONTESTA = "S"
     SELECT A
     SEEK DTOC(ENTRADA) + NUMERO
     IF EOF()
        @ 23,03 SAY XMEN
        @ 23,03 SAY "ERROR EN ARCHIVO DE CORRESPO, "
        @ 23,34 SAY XMENSAJE
        SET CONSOLE OFF
        WAIT
        SET CONSOLE ON
        EXIT
     ELSE
     ENDIF
   @ 09,02 CLEAR TO 23,78
   @ 07,42 SAY "NO. OFICIO:"
   @ 07,54 SAY NUMOFIC
   @ 08,42 SAY "REMITENTE : "
   @ 08,54 SAY REMITEN
   @ 09,03 SAY "ASUNTO               :"
   @ 09,25 SAY AASUNTO
   @ 10,25 SAY AASUNTO1
   @ 11,25 SAY AASUNTO2
   @ 12,25 SAY AASUNTO3
   @ 13,03 SAY "FUNCIONARIO :"
   @ 13,25 SAY FFNOMBRE
   SELECT F
   @ 14,03 SAY "INDICACIONES:"
   @ 14,25 SAY INDICA
   @ 14,28 SAY INDICA1
   @ 14,31 SAY INDICA2
   @ 15,03 SAY "-------------------------------------------------------------------------------"
   @ 16,03 SAY "NO. DE OFICIO (RESP):"
   @ 16,25 SAY OFICONT
   @ 17,03 SAY "FECHA DE RESPUESTA  :"
   @ 17,25 SAY FECHCONT
   @ 18,03 SAY "RESPUESTA           :"
   @ 18,25 SAY ASUNRES
   @ 19,25 SAY ASUNRES1
   @ 20,25 SAY ASUNRES2
   @ 21,25 SAY ASUNRES3
   @ 23,03 SAY "
   @ 23,10 SAY "DESEA SEGUIR CONSULTANDO ? (S/N)"
```

```
      @ 23,47 GET RESP PICTURE "!"
              READ
      IF RESP = "S"
         LOOP
      ELSE
         EXIT
      ENDIF
     ELSE
        @ 23,03 SAY "                                          "
        @ 23,03 SAY "NO SE LE HA DADO RESPUESTA, "
        @ 23,33 SAY XMENSAJE
        SET CONSOLE OFF
        WAIT
        SET CONSOLE ON
        @ 23,03 SAY XMEN
        RESP = " "
        LOOP
      ENDIF
    ELSE
       @ 23,03 SAY XMEN
       @ 23,03 SAY "NO SE ENCONTRO EL REGISTRO, "
       @ 23,34 SAY XMENSAJE
       SET CONSOLE OFF
       WAIT
       SET CONSOLE ON
       @ 23,03 SAY XMEN
       LOOP
    ENDIF
ENDDO

PROCEDURE CONTES
RESP = "N"
F = DATE()
N = SPACE(2)
CONTE = SPACE(1)
DO WHILE .T.
   X = 13
   TURNA = SPACE(4)
   @ 05,03 CLEAR TO 23,78
   @ 05,30 SAY "CON RESPUESTA"
   @ 09,10 SAY "TECLEE LA CLAVE DEL FUNCIONARIO"
   @ 23,03 SAY "PARA TERMINAR DEJE EN BLANCO LA CLAVE DEL FUNCIONARIO"
   @ 09,43 GET TURNA
       READ
       IF TURNA = "    "
          EXIT
       ENDIF
          SELECT F
          SET ORDER TO 2
          SEEK TURNA + "S"
          IF EOF()
             @ 23,10 SAY XMEN
             @ 23,03 SAY "NO EXISTE CLAVE DE FUNCIONARIO, "
             @ 23,34 SAY XMENSAJE
             SET CONSOLE OFF
             WAIT
             SET CONSOLE ON
             @ 23,03 SAY XMEN
             LOOP
          ENDIF
```