UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ACATLAN

ESTUDIO, OPERACION Y ANALISIS DE SISTEMAS MANEJADORES DE BASES DE DATOS COMPARTIDAS EN RED LOCAL (L.A.N.)

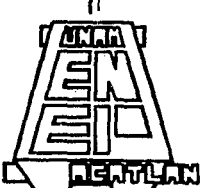
T E S I S

QUE PARA OBTENER EL TITULO DE

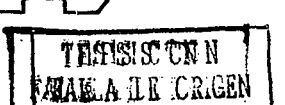
LICENCIADO EN MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A :

JORGE LOPEZ GUERRERO



Acatlán, Edo. de México 1993



UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE.

INTRODUCCION.

El presente trabajo da a conocer los diferentes aspectos teóricos y practicos con respecto a dos Sistemas Manejadores de Base de Datos SQL-SERVER y FORCE que se ejercen en el campo de la informática, y que a su vez pueden ser aplicados a todo tipo de industria o institución de diferente giro que cuente con una Red de Area Local, de aquí que es necesario el conocimiento de conceptos como Redes de Area Local, Bases de Datos y Sistemas Manejadores de base de datos trabajando en Redes. Que son los puntos principales de los tres primeros capítulos.

También se dara un panorama general de los Sistemas SQL-SERVER y FORCE sobre los cuales se habla, proporcionando sus antecedentes, así como características, estructura y la forma en como se constituyen en una Red de Area Local, todo para que el lector comprenda mejor la filosofia de trabajo del tema. Estos puntos y otros de importancia son tratados en el capítulo cuatro y cinco del presente trabajo.

Podemos añadir que una vez fijado el conocimiento de los primeros capítulos aunados con las pruebas y estudios de los capítulos cuatro y cinco se podrá establecer un estudio comparativo de SQL-SERVER y FORCE en sus diferentes características que se llevarán al capítulo seis.

Se debe aclarar que el contenido de este escrito quiere ser totalmente objetivo y no tener preferencia hacia ninguno de los dos Sistemas Manejadores de Base de Datos estudiados. Más bien se busca que el propio lector en base a las alternativas observadas establezca un propio criterio y escoja el sistema que pueda ser más competente para la obtención de información veraz y oportuna para la toma de decisiones adecuadas y encaminadas al logro de los objetivos de la industria o institución.

Finalmente se espera que el presente trabajo sea de utilidad para aquellos que lo consulten, así como una base de consulta para compañeros de nuevas generaciones y sobre todo para que ayude a la difusión del tema investigado.

1. INTRODUCCION A LAS REDES LOCALES

Hacia la mitad de la década de los 70's nacen las llamadas microcomputadoras, las cuales vienen a descongestionar en cierta forma a las viejas máquinas centrales y de mayor tamaño.

A principios de la década de los 80's las microcomputadoras habían revolucionado por completo el concepto de la computación electrónica así como sus aplicaciones y mercado. Sin embargo los gerentes de informática fueron perdiendo el control de la información, ya que el proceso de ésta no se encontraba totalmente centralizado, a esta época se le denominaba la era del floppy disk, ya que los vendedores de microcomputadoras proclamaban que "En una microcomputadora se podía almacenar en 30 diskettes la información de todo un archivo". Sin embargo, de alguna manera, se había retrocedido en la forma de procesar información por que nuevamente había que acarrear la información almacenada en los diskettes de una micro a otra y la relativa o poca capacidad de los diskettes hacía difícil el manejo de grandes cantidades de información.

Con la llegada de una nueva tecnología, se lograron dispositivos que permitieron almacenar grandes cantidades de información. Sin embargo una desventaja era el alto costo que significaba la adquisición de un disco duro, además los usuarios tenían la necesidad de compartir información y programas en forma simultánea, compartir ciertos dispositivos como impresoras y se contaba con poca seguridad.

Estas razones, principalmente, aunadas a otras como el poder compartir recursos de relativa baja utilización y alto costo, llevó a la diversidad de fabricantes a la idea de las redes locales, y de desarrollar un sistema operativo que pudiera coordinar la intercomunicación entre microcomputadoras.

En un principio, las redes de microcomputadoras se formaban por simples conexiones que permitían a un usuario accesar recursos que se encontraban residentes en otra microcomputadora tales como otros discos duros, impresoras, etc. Estos equipos permitían a cada usuario el mismo acceso a todas las partes de un disco causando obvios problemas de seguridad y de integridad en los datos.

Hacia 1983, la compañía NOVELL, INC. fue la primera en introducir el concepto de File Server (servidor de archivos), en el que todos los usuarios pueden tener acceso a la misma información, compartiendo archivos y contando con niveles de seguridad.

Podemos decir que el mercado de redes locales ha llegado a ser el segmento de más rápido crecimiento en la industria de la computación, este crecimiento explosivo está ocurriendo conforme los usuarios se extienden hacia aplicaciones desde las más simples compartiendo recursos hasta sofisticadas aplicaciones interactivas distribuidas, tales como programación de recursos, correo electrónico, documentos compartidos y desarrollo de software.

Ahora que muchos usuarios corporativos han llegado a familiarizarse y a estar acostumbrados a las redes, se están extendiendo redes más grandes de múltiples edificios conectadas a minicomputadoras y macrocomputadoras.

Finalmente se espera para la década de los 90's un continuo crecimiento de la industria de redes locales así como el surgimiento de mas tecnologías de conectividad que conlleven a una mejor forma de administrar y compartir la información.

LA RED TIPICA.

La red típica consta de ocho o más microcomputadoras que se enlazan entre si, utilizando algún tipo de cable de manera que alguna de ellas puedan compartir sus recursos(datos, programas, impresoras, etc.) hacia las demás en distancias relativamente cercanas, digamos dentro de un mismo edificio y tienen acceso a funciones especializadas por medio de comunicaciones transparentes, aclarando que una red puede ser la unión de dos microcomputadoras de cualquier marca aunque no entran conceptos de seguridad e integridad.Las redes se instalan y se administran para que las personas que trabajan las microcomputadoras puedan compartir impresoras, conexiones a mainframes, archivos de programas y datos, modems, tarjetas y otros dispositivos.

Una de estas computadoras es configura como un dispositivo de almacenamiento de información compartida comunmente llamado Servidor de Archivos, existiendo también servidores de impresión y de comunicación. En un servidor de archivos, un usuario no puede accesar a discos que se encuentren en otras microcomputadoras indistintamente, es también tarea de una microcomputadora configurada como administrador de los recursos comunes. Al hacer esto, se logra una verdadera eficiencia en el uso de estos recursos así como una total integridad de los datos. Los archivos y programas pueden ser accesados en modos multiusuario guardando el orden de actualización por el procedimiento de bloqueo de registros. Es decir cuando algún usuario se encuentra actualizando un registro, este se bloquea para evitar que algún otro usuario lo extraiga o intente actualizar.

Una red local o LAN (Local Area Network), es un sistema de comunicación que enlaza un conjunto generalmente de microcomputadoras, con la finalidad de compartir recursos tanto de hardware como de software, transmitir, recibir y compartir información, es decir es una necesidad que puede soportar diferentes dispositivos y transmitir los diferentes tipos de información.

Los elementos clave que determinan el costo y alcance de una red local son sus topologías, su medio de transmisión, y sus protocolos de control de acceso al medio.

Las topologías mas comunes son la estrella, la bus o árbol y la ring o anillo.

El medio de transmisión incluye cable de par torcido, cable coaxial de banda ancha y banda base y cables de fibra óptica.

Los protocolos de control de acceso al medio mas comunes son el CSMA/CD, token passing con token bus y token ring.
(estos elementos se analizarán con más detalle en temas siguientes).

Otro elemento que se pueden incluir en la definición es que una red local es normalmente privada más que pública o comercial.

Finalmente podemos decir que una LAN es probablemente la mejor selección cuando una variedad de dispositivos y una mezcla de tipos de tráfico se involucran.

1.1 CARACTERISTICAS

Algunas de las características indispensables, o deseables para un mejor rendimiento de las redes locales son:

- Servidores, de archivos, impresión, comunicación, base de datos, etc.) y estaciones de trabajo.

- Alta velocidad de transmisión (0.1-100 Megabytes por segundo, Mbps)

- Cortas distancias de comunicación entre los dispositivos (0.1-50 kilómetros)

- Baja tasa de errores (10^{-8} - 10^{-11}). 10^{-8} por cada 10^{8} bits, un bit se espera que tenga error.

- Tarjetas de red, estas tarjetas tienen como tarea el intercambiar información de un nodo a otro, de un nodo al server o viceversa, utilizando un enlace físico (típicamente cable coaxial o par torcido).

- Sistema de cableado, son los medios de comunicación (que más adelante se estudiarán), de conexión, conectores especiales, etc.

- Software de aplicación, Es la forma en la cual el usuario de cada estación de trabajo, puede utilizar sus programas y archivos específicos. Este software puede ser tan amplio como se necesite: procesadores de palabra, paquetes integrados, sistemas administrativos y de contabilidad y áreas afines, entre otros.

- Sistema operativo LAN, Este sistema se engloba en dos componentes básicos. El sistema operativo de red mismo del servidor y el sistema operativo de la estación de trabajo. El sistema operativo del servidor de red se ejecuta dentro de éste y procesa y controla todos los servicios.

El sistema operativo del servidor se puede dividir en cinco subsistemas básicos:

1. Nucleo de control o Kernel, distribuye la actividad del usuario tan uniformemente como sea posible a través de los servicios de disco, y de cualquier dispositivo de entrada/salida, mantiene la información de estado de muchos procesos, es decir coordina los diferentes procesos de los otros subsistemas.

2. Interfaces de red, apoyan las tecnologías que son la implementación real del medio de la red.

3. Sistemas de archivos, mecanismos mediante los cuales los datos son organizados, almacenados y recuperados, a partir de discos duros o RAM entre otros.

4. Extensiones de sistema, por lo general son manejadores de protocolo /1 de alto nivel que efectúan operaciones tales como el traslado entre protocolos de acceso de archivos requeridos por los diferentes sistemas operativos de usuarios o estaciones.

5. Servicios del sistema, cubren todos los servicios que no se ajustan fácilmente en cualquiera de las otras categorías del modelo. Estos pueden ser servicios para almacenar y dirigir al nivel del sistema, tales como enfilar protocolos o subsistemas de contabilidad.

- Proceso distribuído, en el procesamiento distribuído, los diferentes sistemas operativos funcionando concurrentemente deben estar conectados de una manera sencilla y funcional. El procesamiento distribuído es una tecnología de "Sistema Abierto" que ofrece a los usuarios la libertad de elegir el hardware y el software que mejor trabaje para ellos.

- Manejo de seguridades, por medio de esta característica es posible evitar que usuarios no calificados puedan leer información confidencial e incluso puedan dañarla o adicionar datos que de alguna manera afecten al sistema administrativo de una empresa esto asegura que la información no pueda ser accesada por usuarios que no la requieran corriendo el riesgo de alteraciones o pérdidas indeseables, las seguridades se pueden manejar en cuatro niveles, por grupo de usuarios, por usuarios individuales, por subdirectorio y por archivo, la prioridad de las seguridades es más a nivel archivo que a nivel grupo.

- Manejo de passwords, para tener una mayor protección de la información se cuenta con el manejo de palabras claves para evitar el acceso a la red a personas indeseables o no autorizadas. Este sistema es una buena solución para empresas que no contemplen un crecimiento de equipo conectado a la red que exceda cuatro nodos en un periodo por lo menos de un año.

/1 DISPOSITIVO DE COMUNICACION

- Confiabilidad, conforme aumenta el nivel de aplicación de una red en una organización o una empresa, también aumenta la dependencia hacia ella, por lo que el grado de confiabilidad en la mísma, debe ser máximo. Esto es una característica indispensable que debe tener toda red.

- Conexiones remotas o entrelazamiento de redes, es posible conectar una red o estación remota por medio de productos de entrelazamiento. Existen cuatro productos para entrelazar redes:

1. Repetidores, solo pueden enlazar las redes con formatos de protocolo similar, son severamente limitados por la distancia.

2. Puentes, interconectan a dos redes de tal manera que los usuarios piensen que están conectados a una sola red, son mas eficientes que los repetidores en esta tarea, debido a que los primeros envían solamente datos que son necesarios para la otra red.

3. Ruteadores, interconectan a dos redes diferentes, sus capacidades van mucho más allá de las de un puente, ya que los ruteadores no son transparentes, los usuarios que desean tener acceso a los recursos o a otra red, deben dar una dirección de destino.

4. Gateway, interconectan a redes de diferentes tipos. Realizan la interconexión de la manera más directa posible. Convierten la salida de los dispositivos de una red en el protocolo que entienden los dispositivos de la otra red. A diferencia de los ruteadores, los gateway de hecho traducen un protocolo a otro.

- Integridad de datos, las redes son instaladas y crecen a partir de la necesidad de compartir información. Independientemente del tamaño de la red, los datos representan la vida y el éxito de la organización, una pérdida de datos puede ser devastadora es por eso que la capacidad para recuperar y establecer es fundamental para cualquier red. Los usuarios y administradores de la red necesitan tener la seguridad de que los datos se encuentran protegidos de daño y corrupción, y de que pueden ser recuperados y referenciados fácilmente. Las amenazas a los datos de la red surgen desde muchas fuentes.- Error del usuario, caída del sistema, desastre natural y otros. Cuando los datos se transmiten de un dispositivo a otro en la red se incrementan los riesgos. Debido a que la recuperación es el primer paso para restablecer la red, la integridad de los datos se extiende hasta la administración de los medios fuera de línea.

En algunas redes existen sistemas de monitoreo para recuperar rápidamente la información perdida desde los medios almacenados y para proteger el contenido de los medios de un acceso no autorizado.

- Tolerancia a fallas, la tolerancia a fallas del sistema está diseñada para que el sistema nunca se detenga, o bien para que cuando se detenga, lo haga el menor tiempo posible. Fallas de todo tipo producen interrupciones no deseadas en proceso de información, más importante aún que la interrupción es el daño resultante de pérdida o corrupción de la información. Archivos de datos, archivos indexados, estructura de árbol en subdirectorios y otros componentes del sistema, pueden todos estar corruptos o dañados por fallas originadas en cualquier parte de la red, acciones inconclusas causadas por estas fallas que ocurrieron en la estación de trabajo de la red en el medio de comunicación, en el servidor de la red o en los discos duros del servidor, pueden dar como resultado una gran variedad de situaciones desastrosas.

La tolerancia a fallas permite el manejo de discos en espejo y duplicados, con lo que se logra un funcionamiento ininterrumpido por fallas en el disco, suponiendo que por fallas en el sistema eléctrico el server es apagado incorrectamente y esto provoca que el disco se dañe, en este caso pese a tener un respaldo en cinta, el tiempo de restauración será casi siempre transparente ya que la información está duplicada en línea en ambos discos del server, con lo que sólo bastará deshabilitar el disco dañado, para que el server esté funcionando nuevamente, esto no podrá llevarse más allá de unos cuantos minutos.

Adicionalmente los discos en espejo consisten en almacenar la información en dos discos duros idénticos en el servidor. Si uno de ellos tiene problemas, el servidor de la red notifica al administrador de la situación y continúa operando con el disco en buen estado.

1.2 TOPOLOGIAS

La topología es la estructura que consta de las rutas (arreglos físicos), que proveen las comunicaciones integrando dispositivos o nodos de la red y la interconexión entre ellos.

La topología se puede interconectar en base a como se desea asignar el flujo de información a través de la red. El control puede ser centralizado o distribuído. El control centralizado de acceso a la red (¿Cuáles nodos pueden enviar mensajes y cuándo?) y la distribución de canal (¿Qué tanto del canal físico un nodo puede usar y por cuánto tiempo?) son controlados por un nodo. Cuando el control es distribuído, todos los nodos tienen igual privilegio para transmitir la información y no tiene que esperar a que un nodo central les conceda el permiso para transmitir.

La estructura topológica o arquitectura de la red. tiene relación con el arreglo físico y la conectividad de los elementos de la red, el tipo de datos que se podrán transmitir, la velocidad, la capacidad, la eficiencia y la clase de aplicaciones que la red puede soportar.

Para poder definir con detalles las topologías se explicarán brevemente algunos conceptos de comunicación de datos y medios de transmisión.

- Línea punto a punto: Esta línea es usada para describir un canal que se establece entre dos estaciones. Estas estaciones pueden ser un CPU y una terminal o dos CPUs, la longitud de la línea es muy importante, puede ir desde metros hasta kilómetros.

  Este tipo de línea puede operar en modo simplex, half duplex o full duplex.

Simplex .- Permite que los datos se transmitan sólo en una dirección.

Half duplex .- Permite que los datos se transmitan en ambas direcciones pero no al mismo tiempo.

Full duplex .- Realiza la transmisión de datos en ambas direcciones simultáneamente, no hay retardo.

- Línea Multipunto : En una línea multipunto pueden conectarse más de dos terminales en la línea, la conexión de las diferentes terminales se efectúa por agrupamiento ó por derivaciones múltiples.

Por agrupamiento se dice que varias terminales se conectan al mismo punto y es lo que conocemos como enlace multipunto. Se dice que la línea trabaja con derivaciones múltiples cuando se conectan las terminales a la línea en puntos diferentes o por una combinación de ambas.

- Medios de transmisión: Es la ruta física entre transmisores y receptores dentro de la red.

Existen dos tipos de transmisión que son:

- Transmisión en serie: Consiste en enviar una serie de datos por una línea de comunicación, bit por bit.

- Transmisión en paralelo : Consiste en enviar una serie de datos por una línea de comunicación, todos los bits al mismo tiempo.

Los tres principales tipos de cables en una LAN son el coaxial, el telefónico UTP/STP y el de fibra óptica.

Cable Coaxial, está formado por un alambre conductor básico abierto por una placa metálica que actúa como tierra. El alambre conductor y la tierra se encuentran separados por un aislante plástico y, finalmente, todo el conjunto es protegido por una cubierta exterior, estos cables pueden transportar una señal eléctrica a mayor distancia entre mas grueso es el conductor sus ventajas ofrecidas son: Transmisión de voz, video y datos, fácil instalación, con ancho de banda de 10 Mbps, distancias hasta 600 metros sin necesidad de repetidores y buena tolerancia a interferencias.

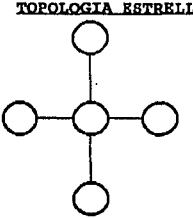
Cable telefónico, está formado por dos alambres de cobre que se encuentran aislados por una cubierta plástica y torcidos uno contra el otro. Es ésta la característica que los distingue con el nombre de par torcido. El par torcido a su vez se encuentra protegido por una cubierta aislante y protectora en la capa exterior. Los llamados UTP son baratos y flexibles permitiendo manipular una señal a una distancia máxima de 110 metros sin el uso de amplificadores, los STP son más caros y menos flexibles, pero permiten un rango de operación de hasta 500 metros, ambos tienen facilidad de instalación, ancho de banda de 10 Mbps y buena tolerancia a interferencias.

Cable de fibra óptica, se utiliza para aquellos casos en donde las grandes distancias son un factor determinante para la implantación de una red, se requiere una alta capacidad de aplicaciones de comunicación y es inmune a interferencias. La fibra óptica dividida en fibra óptica exterior e interior se encuentra protegida por una placa aislante y protectora en la parte mas exterior para darle mayor integridad al cable.

Es sin embargo, muy flexible y sus distancias obtenidas para LAN son de 2000 metros de nodo a nodo sin amplificadores, transmiten voz, video y datos por el mismo canal y tiene un ancho de banda de 200 Mbps entre otras ventajas.

Las topologías más comunes para redes de área local son: La estrella, anillo o ring y bus o árbol (la bus es un caso especial de la árbol, con solo un tronco y sin ramificaciones; se usa el término bus/árbol cuando la distinción no es importante).

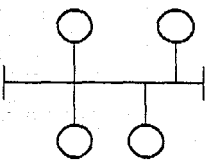
TOPOLOGIA ESTRELLA

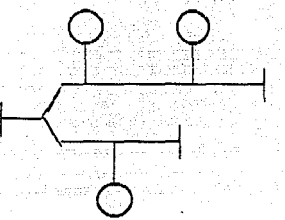


Las redes locales de este tipo de topología constan generalmente del nodo central, el cual puede ser la computadora misma, o bien un controlador dedicado al manejo de terminales y periféricos(server). El controlador interrroga a cada dispositivo si tiene datos para enviar. Y solamente cuando el controlador otorga su permiso, los dispositivos pueden enviar sus mensajes o datos. Una característica principal de la red estrella es que la inteligencia necesaria para controlar la red reside en un solo lugar y está compartida por todos los usuarios del sistema. Pero, debido a lo mismo si el punto principal falla, toda la red falla. Por lo tanto, la confiabilidad y la necesidad de redundancia son consideraciones importantes en la topología estrella. Esta topología no permite cursar grandes flujos de tráfico y no es adecuada para redes con gran dispersión geográfica.

Dos ventajas importantes son que el nodo aisla a una estación de trabajo con otra, resultando una red fiable frente a averías, en la estación de trabajo (la avería de una estación no afecta el funcionamiento de la red) y la flexibilidad-complejidad es buena permitiendo incrementar o disminuir con sencillez el número de estaciones, ya que las modificaciones son sencillas y están todas localizadas en el nodo central.

TOPOLOGIA BUS/ARBOL





BUS ARBOL

El tipo más usado en LAN es un bus o árbol usando cable coaxial, y la ring que usa cable de par torcido, cable coaxial o fibra óptica.

En esta topología, un canal de comunicaciones se comparte para todos los nodos o estaciones sobre la red, la información baja en ambos sentidos y consiste de una sección de cable con dos extremos llamado bus, la información se transmite sobre el bus procedente de un nodo y las señales se propagan a lo largo del canal y donde todos los nodos conectados al bus pueden escuchar cada transmisión realizada.

Pero, dado que varios elementos comparten el mismo canal o trayectoria de datos, sólo un elemento puede transmitir a la vez, usualmente en forma de paquete de datos, conteniendo éste la dirección del destino del paquete, que se propaga por el medio, todos los otros elementos lo reciben, pero sólo lo copia el elemento direccionado.

La topología bus/árbol es una configuración multipunto; ésto permite que más de dos dispositivos a la vez puedan ser conectados y capaces de transmitir en el medio, esto es opuesto a la topología que ha sido discutida anteriormente, la cual es una configuración punto a punto.

Hay dos técnicas de transmisión en uso para la bus/árbol la banda base y la banda ancha.

La banda base usa señales digitales y puede ser empleada en cables de par torcido o cables coaxiales. La banda ancha usa señales analógicas en el rango de frecuencia de radio (RF) y es empleada sólo en cable coaxial.

Hay una variante que conocemos como "solo canal de banda ancha", la cual tiene señales características de banda ancha pero con algunas de las restricciones de banda base, esta técnica es explicada más adelante.

El multipunto natural de la bus/árbol ocasiona severos problemas comunes de sistemas de banda base y banda ancha(que más adelante se verán). Primero, está el problema de determinar qué estación de el medio puede transmitir a cualquier punto a la vez. Segundo la configuración multipunto es hecha con señales balanceadas.

Cuando dos dispositivos cambian datos sobre una unión, el poder de la señal de el transmisor puede ser ajustada dentro de ciertos límites.

La señal puede ser bastante, así que , después de la atenuación a través del medio, éste encuentra el requerimiento fuerte para recibir una mínima señal, y así bastante fuerte para mantener una adecuada señal para propagar la proporción. Si éste es muy fuerte, la señal puede sobrecargar los circuitos para el transmisor, creando otras señales superiores. Sin embargo éste es fácil para una unión punto a punto, la señal balanceada para una configuración multipunto es compleja. Si cualquier dispositivo puede transmitir a cualquier otro, entonces la señal balanceada puede ser alcanzada por todas las permutaciones de estaciones tomando dos a la vez.

Sistema banda base

El sistema banda base se define como aquél que tiene un canal simple y usa señales digitales. Este sistema tiene principalmente las siguientes características:

La transmisión es a base de paquetes en un ancho de banda, no se acepta multicanalización por división de frecuencias (FDM, de sus siglas en ingles Frecuency-Division Multiplexing), es bidireccional porque la transmisión es en ambas direcciones, como muestra la 1, además el sistema puede extenderse hasta una distancia de 1 km.

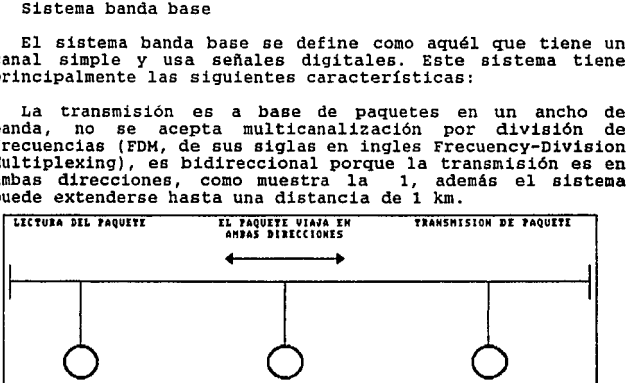


FIGURA 1. TECNICA DE TRANSMISION BANDA BASE

El término banda ancha se refiere a algún canal que tiene una amplitud mayor que un canal de voz (0.3-3.4 Kilohertz). Por lo regular, cuando se habla de un sistema banda ancha se refiere a la transmisión de señales analógicas. A continuación para tratar estos sistemas se dividen en dos: banda ancha que permite FDM y aquellos que transportan solo una señal analógica simple denominados banda ancha de canal simple.

Sistema de Banda Ancha FDM

Los sistemas de banda ancha FDM usan comunmente señales analógicas y permiten tener múltiples canales. En el espectro de frecuencia de cables se divide en canales o secciones de anchos de banda. Los canales separados pueden soportar tráfico de datos, tv, y señales de radio, aquí cabe aclarar que algunos canales de datos pueden transmitir señales digitales.

Estos sistemas tienen dos rutas una de transmisión y otra de recepción. La ruta de transmisión va dirigida hacia el "headend" (elemento que se coloca al final de uno de los extremos del bus), en el canal todas las estaciones transmiten.

La ruta de recepción va del "headend" hacia fuera, las señales son propagadas por el "headend" en esta ruta y todas las estaciones reciben en la misma.

Físicamente, dos configuraciones diferentes son usadas para implantar las rutas de transmisión y recepción (ver figura 2), en la configuración banda ancha "Mid-Split" se transmite a una frecuencia f1, luego al llegar al "headend" (en este caso un convertidor de frecuencias), los retransmite a una frecuencia f2, en la cual van a ser recibidas las señales por las estaciones (ver figura 2.a). La otra configuración es llamada banda ancha cable-doble se transmiten y reciben las señales a la misma frecuencia f1 y el "headend" es tan sólo un elemento pasivo (ver figura 2.b)

Algunos de los principales componentes de estos sistemas son: el cable, acopladores direccionales y "splitters". Los cables usados son de tres tipos: "trunk", de distribución y "drop"; su diámetro está en un rango de 0.635 cm a 2.54 cm y su atenuación se encuentra en un rango de 0.7 a 6 decibeles (db), por 30.48 cm.

Con respecto a los acopladores direccionales o "taps" son el medio para dividir una entrada en dos salidas y la combinación de dos entradas en una salida. "Splitters" son usados para ramificar el cable, dan una atenuación igual a lo largo de las ramificaciones (ver figura 3).

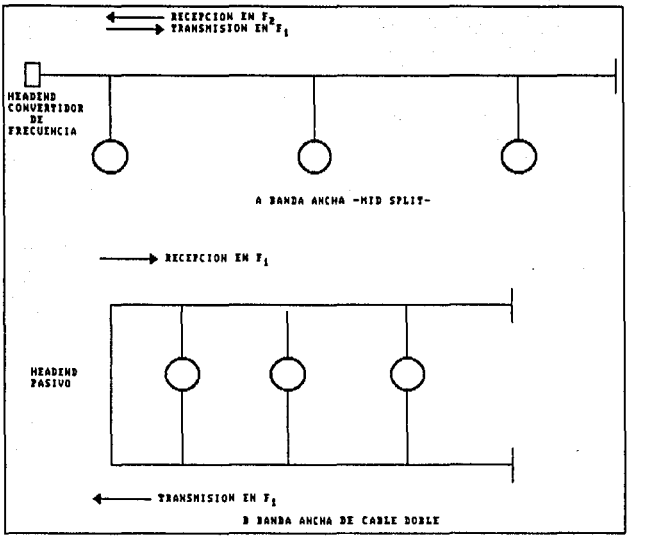


FIGURA 2. TECNICAS DE TRANSMISION BANDA ANCHA

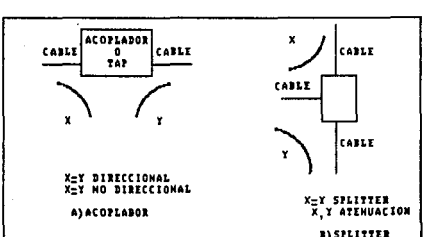


FIGURA 3 ACOPLADORES DIRECCIONALES Y SPLITTER

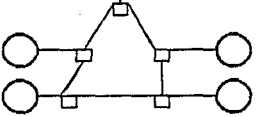
Tres clases de transferencia de datos digitales son posibles en un cable de banda ancha; dedicado, "switched" y de acceso múltiple. En el dedicado se tiene un enlace punto a punto, transmitiendo y recibiendo a la misma frecuencia. En el "switched", los dispositivos se conectan a un modem y este al medio de transmisión para una transmisión y una recepción se les asigna una frecuencia a los dispositivos, entre todas las frecuencias que se tengan (fo..fn). En el acceso múltiple todos están a la misma frecuencia y sólo uno puede transmitir a la vez.

Sistema de Banda Ancha Canal Simple

Un sistema banda ancha canal simple tiene las siguientes características: transmisión bidireccional (la técnica de transmisión es igual a la de banda base (ver figura de sistema banda base), usa topología bus, puede no usar amplificadores y no hay necesidad de un "headend". Se transmite de acuerdo a alguna forma de manipulación por corrimiento de frecuencia (FSK Frecuency Shift Keying), a bajas frecuencias ya que en éstas se tiene menos atenuación.

Finalmente La topología bus/árbol es caracterizada por el uso de un solo cable de múltiple acceso. Los dispositivos comparten el medio y sólo un dispositivo puede transmitir a la vez, presenta simplicidad en su instalación, el retardo de propagación de información es reducido, tiene un costo reducido y gran flexibilidad para variar el número de estaciones de trabajo, pero una avería en cualquier punto del bus implica una transtorno en toda la red y las averías son difíciles de localizar.

TOPOLOGIA ANILLO



Consiste de un número de repetidores, cada uno conectado a otros por enlace de transmisión unidireccional para formar una sola trayectoria cerrada (ver figura 4), los datos se transmiten secuencialmente bit a bit, y cada repetidor los regenera y los retransmite.

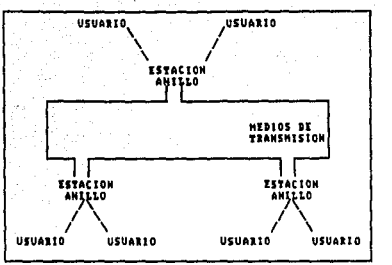


FIGURA 4. SISTEMA ANILLO

El objeto de emplear la topología anillo en una red local es eliminar la dependencia de un elemento central, como en el caso de estrella, al mismo tiempo que proporciona canales de comunicación entre todos los dispositivos de la red para transmisiones a altas velocidades. Las redes de este tipo transmiten siempre en una sola dirección.

En las redes locales anillo, los datos fluyen secuencialmente alrededor de la trayectoria del anillo, con cada nodo conectado por un repetidor. Las redes tipo anillo se diseñan con la intención de distribuir el control de la red local a cada una de las entidades a comunicar.

El control distribuído quiere decir que cada interfaz en el anillo tiene suficiente lógica para permitirle determinar cuándo puede enviar un mensaje que va de nodo a nodo sobre una dirección. En general esto significa que el inicio y el término de mensajes deben ser especialmente marcado permitiendo a una estación cualquiera insertar su propio tráfico entre mensajes entrantes. EL receptor de un mensaje puede marcarlo como "reconocido" antes de continuar con la transmisión del mismo.

En este tipo de red se dan los siguientes puntos:

- Los mensajes circulan alrededor del anillo en serie, en enlaces punto a punto entre repetidores. Una estación que desea transmitir, espera su turno enviando los mensajes sobre el anillo en forma de paquete, que contiene el origen y el destino en los campos de dirección, así como los datos.

- Conforme el paquete circula, el nodo destino copia los datos en su memoria (buffer local) y el paquete continúa circulando hasta volver al nodo origen, permitiendo así una forma de reconocimiento.

- Ningún nodo toma decisiones de enrutamiento, su único requerimiento es que sea capaz de reconocer el origen del paquete de mensaje y la dirección de éste.

Para que el sistema se comporte como una red de comunicaciones, requiere tres funciones: inserción del mensaje, recepción del mensaje y el borrado de mensaje. Para la inserción del medio en el canal de comunicaciones y transmitirlo, se encarga el repetidor. En la recepción del mensaje, como los mensajes van en paquetes y en el un campo de dirección del destinatario, el repetidor verifica si le corresponde, si es afirmativo lo copia de lo contrario lo retransmite. Con respecto al borrado del mensaje se tienen dos criterios: uno es que el destinatario después de copiarlo se encarge de borrarlo y el segundo consiste en que el transmisor después de dar una vuelta lo borre, de esta manera se puede hacer que varias estaciones puedan copiar el mensaje.

El más grande beneficio de LAN's con topología anillo es que son a base de enlaces de comunicación punto a punto, también representa un beneficio que cada repetidor regenere y retransmite cada bit ya que se logra un mayor grado de seguridad de transmitir correctamente y en ciertas condiciones se alcanzan tasas altas de datos procesados efectivos.

Otras topologías son la de estrella distribuída que es una extensión de la arquitectura en estrella por interconexión de varias de éstas, la de malla donde cada estación esta conectada con todas o varias estaciones de trabajo formando una estructura simétrica o irregular y la múltiple, que consiste en varias redes, en lugar de una, conectadas a través de un puente o dispositivo de entrelazamiento sin que sea necesario que todas las redes tengan la misma topología.

1.3 PROTOCOLOS DE CONTROL DE ACCESO AL MEDIO

El protocolo de control de acceso al medio es el método para determinar cuál dispositivo de comunicación tiene acceso al medio de transmisión en cualquier tiempo.

A continuación se describen los protocolos de acceso al medio, existiendo básicamente dos.

1.- Carrier Sense Multiple Access With Collision protection (CSMA/CD).

2.- Token-Passing.

Además, se dan en cada caso los sistemas de transmisión que soportan (banda base o banda ancha) y sus correspondientes características físicas del medio de transmisión.

1.- CSMA/CD, Carrier Sense significa que cada nodo está "Sensando" al bus para encontrar el momento en que se encuentra desocupado. Multiple Access significa que todos los nodos tienen el mismo derecho de enviar sus paquetes y Collision Detection significa que en el remoto caso de que dos nodos envíen información al mismo tiempo causando una colisión, CSMA/CD la detecta y le indica a ambos que la envíen nuevamente.

Esto es, CSMA/CD está basado en contensión, es decir las estaciones de usuario compiten por igual para el acceso a la red, donde cada estación debe escuchar al bus de la red para ver si está libre y así poder transmitir un mensaje, si la red está ocupada, debe esperar un cierto período de tiempo aleatorio antes de tratar de transmitir otra vez. La característica "detección de choques" de CSMA/CD significa que mientras una estación está transmitiendo, debe escuchar a la red para asegurarse de que ninguna otra estación está transmitiendo causando colisión. Si ésta detecta una colisión, "refuerza" a la red transmitiendo una señal detectable, luego "detiene" la transmisión y solicita el mensaje.

Mientras que la estación siga detectando una colisión, es forzada a esperar antes de que pueda continuar transmitiendo. Debido a que todas las estaciones deben oir simultáneamente todas las transmisiones, las redes CSMA/CD utilizan una topología bus.

En la figura 5 se muestra como trabaja la técnica CSMA/CD.

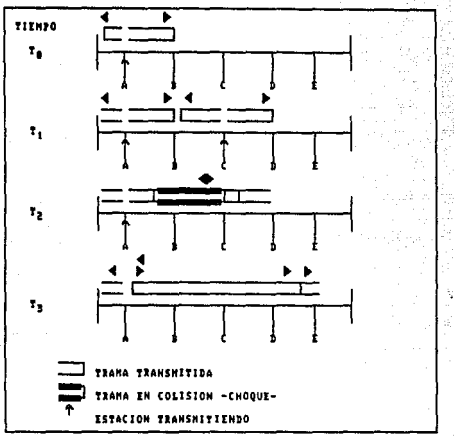


FIGURA 5. CSMA.

En t0, A inicia la transmisión de una trama dirigida a D. En t1, B y C, están listos para transmitir B sensa o detecta una transmisión y desiste. C, sin embargo, no esta al tanto de la transmisión de A e inicia su transmisión (t2). Cuando la transmisión de A llega a C, C detecta la colisión y cesa la transmisión.

El estándar IEEE /2802.3 especifica los detalles de CSMA/CD, así como el medio físico de transmisión a ser usado.

El nivel físico se especifica para un cable coaxial 50 ohms, banda base. En este contexto, banda base se refiere al uso de la señalización analógica, las señales digitales se transmiten en un promedio de 10 Mbps.

Para mantener una calidad adecuada de la señal a 10 Mbps, la longitud máxima de un segmento de cable es de 600 metros. Para extender la longitud de la red, se usa un repetidor.

/2 DE SUS SIGLAS EN INGLES, ASOCIACION DE INGENIEROS ELECTRONICOS Y ELECTRICOS. ASOCIACION QUE DESARROLLA ESTANDARES DE COMUNICACION, CON LA FINALIDAD DE ESTABLECER PROCEDIMIENTOS PARA LOGRAR LA COMUNICACION ENTRE LOS NODOS

CSMA/CD es una técnica de uso atractivo, debido a su simplicidad. Es fácil de implementar, y para ello existe un gran número de tarjetas en el mercado.

Además, su funcionamiento es bueno a diversas cargas. Sin embargo tiene dos inconvenientes:

1.- Bajo cargas severas, el funcionamiento del sistema se degrada severamente, hay mas colisiones, y la red se satura con intentos de retransmisión de tramas, que previamente han sufrido una colisión.

2.- La cantidad de retardo experimentado para transmitir una trama es impredecible, ya que cualquier transmisión puede colisionarse. Esta es una característica no deseable para aplicaciones en tiempo real tales como una red de control de procesos,un sistema de adquisición de datos en subestaciones, etc.

La figura 6, muestra una red local Ethernet que se caracteriza por contar con el protocolo CSMA/CD y topología BUS.

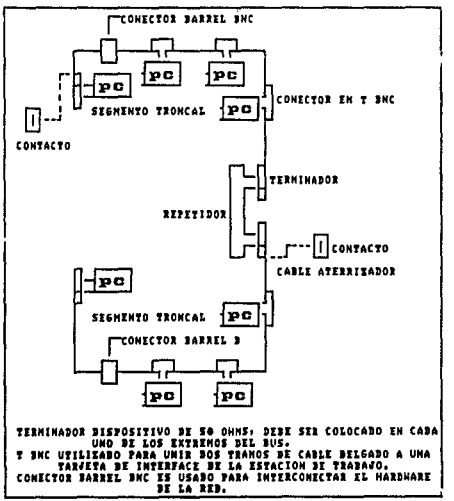


FIGURA 6. RED ETHERNET CON CABLE COAXIAL

2.- **TOKEN PASSING.**La segunda técnica de acceso al medio es Token-Passing, la cual define reglas para accesar a la red. Una trama de datos especial, llamada token, es pasada de estación a estación en una secuencia ordenada.

Unicamente la estación que tiene el token, tiene el derecho de transmitir. La estación pasa el token a su sucesor cuando ya no tiene más datos para transmitir o cuando el tiempo de captura del token (t-holding) ha terminado. Token-Passing puede ser implementado usando topología anillo o bus. En las redes Token Ring /3 , el token es pasado al sucesor físico de la estación. Si una estación no está iniciando una transmisión, debe repetir el dato recibido y enviarlo a la próxima estación física. Apropiado para que una trama de datos establezca contacto con todas las estaciones en el anillo, debe ser recibido y difundido alrededor del anillo por cada estación en turno. Esta rotación total de trama de datos es una parte integral del protocolo Token-Ring, que se verá más adelante.

En redes Bus, los tokens no son pasados al vecino más próximo físico, pero sí alrededor del anillo lógico, definido por la dirección lógica de las estaciones que vienen antes y después en el anillo lógico, de modo que se pueda desarrollar la recuperación inmediata en caso de que el token sea extraviado.

Además de las tramas token, otros mensajes de protocolo son enviados entre estaciones para determinar el orden del Token-Passing, para admitir nuevos miembros y para remover estaciones del anillo lógico que falla al responder.

Una de las ventajas del método Token-Passing sobre CSMA/CD es la habilidad para poner prioridades para la transmisión de datos. En una red CSMA/CD es difícil poner prioridades dado que todas las estaciones tienen igual acceso a la red. Además las redes Token-Passing son determinísticas ya que el acceso a la red sigue un conjunto de reglas secuenciales y ordenadas y contrario a las redes CSMA/CD, las redes Token-Passing no tiene límites rigurosos en la longitud de la red, la cual puede ser extendida arriba de 25 kilómetros.

A continuación se da una descripción más detallada de cada una de las técnicas de acceso Token-Passing.

TOKEN-BUS. Para superar las dos desventajas de CSMA/CD , se desarrolló la técnica de Token-Bus, en el cual las estaciones sobre el bus forman un anillo lógico. Esto es, a las estaciones se les asigna posiciones lógicas en una

---

/3 RED LOCAL DE LA EMPRESA IBM

secuencia ordenada, con el último miembro seguido del anterior (ver figura 7).

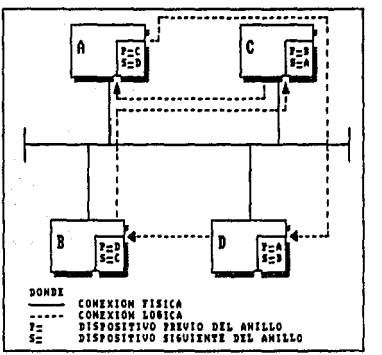


FIGURA 7. TOKEN BUS

Cada estación "sabe" la identidad de las estaciones precedentes y siguientes. El ordenamiento físico de las estaciones sobre el bus es irrevelante e independiente del ordenamiento lógico.

Una trama de control llamado token regula el derecho de acceso y contiene una dirección destino, la estación que recibe al token obtiene el control del medio durante un cierto tiempo, ésta puede transmitir una o más tramas, sondear estaciones y recibir respuestas; si el tiempo ha terminado pasa el token a la siguiente estación en una secuencia lógica, esta estación tiene ahora el permiso para transmitir. Así, la operación de estado-uniforme (steady-state), consiste en alternar la transferencia de datos y la transferencia del token, además se permiten estaciones en el bus aún cuando no usen token. Estas estaciones pueden responder sólo a sondeos o solicitudes de reconocimiento.

El Token-Bus requiere mucho mantenimiento, y cuando menos se deberán realizar las siguientes funciones por una o más estaciones sobre el bus.

a) Inicialización del anillo.- Al configurar o establecer la red, o después de que un anillo lógico haya fallado, el anillo deberá reiniciarse, para ello se requiere algún algoritmo descentralizado para determinar quien va primero, quien va segundo y así sucesivamente.

b) Adición del anillo.- Periódicamente, a las estaciones no participantes se les debe garantizar la oportunidad de ingresar por sí mismas al anillo.

c) Eliminación del anillo.- Una estación puede eliminarse voluntariamente del anillo añadiendo su predecesor y su sucesor.

d) Recuperación.- Puede ocurrir un número de errores incluyendo direcciones duplicadas (dos estaciones creen que es su turno) y falla del anillo (no hay estaciones que piensen que es su turno).

El Token-Bus, a diferencia del CSMA/CD tiene un buen desempeño bajo cargas considerables. Además, los retardos son limitados, ya que cada dispositivo tiene que esperar, al menos una vuelta completa del token. La única desventaja de esta técnica es su complejidad, lo cual ha reducido la introducción de chips.

El estándar IEEE 802.4 Token-Bus especifica opciones físicas. Las tres usan cable coaxial de 75 ohms CATV, y señalización analógica RF. Dos de las especificaciones del medio emplean señalización banda ancha en 1 canal.

Esto significa que la señalización analógica se usa, pero los modems para transmisión no tienen un gran ancho de banda y tampoco se puede usar múltiplexaje por división de frecuencias.

La opción más simple y más económica del nivel físico opera a 1 Mbps. Esta pretende proporcionar una red local de bajo costo que puede ser instalada con cable coaxial flexible o semirígido.

Una vez que el sistema está instalado, los cambios futuros se limitarán a agregar más entradas al cable existente y añadiendo extensiones al final del cable, pero sin incrementar significativamente el tamaño de la red.

La segunda opción es también de un canal, banda ancha, a cinco o diez Mbps. Este sistema es más costoso y tiene una tasa más elevada de datos que la prímera opción, pero es más económica que un sistema total de banda ancha.

Cuando se implementa con cable coaxial semirígido este sistema puede convertirse a banda ancha, realizando algunos cambios en hardware.

La opción final es un sistema total de banda ancha que puede manejar en forma simultánea múltiples canales de datos, así como canales de video.

Se tienen tres promedios de datos :

1 Mbps que ocupa un canal de 1.5 Mhz.
5 Mbps que ocupa un canal de 6 Mhz.
10 Mbps que ocupa un canal de 12 Mhz.

Resumiendo en el protocolo Token-Bus las estaciones que están conectadas al bus llevan un orden lógico que no necesariamente es de acuerdo a su posición física, formando una ruta cerrada. El acceso al canal es regulado por una señal de control llamado "token". se tienen 3 especificaciones y todas ocupan el cable coaxial y señales analógicas. Dos de las especificaciones son en un sistema banda ancha canal simple. La tercera especificación es en un sistema de banda ancha donde se pueden manejar canales de video simultáneamente, a tasas de datos de 1 a 10 Mbps.

TOKEN RING. La topología anillo consiste de ciclos cerrados de repetidores con dispositivos conectados a los repetidores.

Los datos circulan alrededor del anillo en series de enlace punto a punto.

El Token-Ring es el único protocolo de control de acceso al medio para topología anillo, especificado por el IEEE 802. La técnica Token-Ring se basa en el uso de un token que circula alrededor del anillo cuando todas las estaciones están desocupadas. Una estación que desea transmitir debe esperar hasta que detecte un Token-Passing. Luego cambio el token de token libre a token ocupado, este puede hacerse cambiando el último bit del token, por ejemplo de 01111111 a 01111110. Luego transmite una trama inmediatamente después del token ocupado. Si no hay token libre en el anillo, las otras estaciones que desean transmitir deben esperar.

El token en el anillo efectuará un recorrido y será depurado por la estación transmisora, la cual insertará un nuevo token libre en el anillo, cuando se hayan dado las siguientes condiciones: La estación ha completado la transmisión de su trama, y el token ocupado ha regresado a la estación.

Si la longitud del bit del anillo es menor que la longitud de la trama, la primera condición implica la segunda; si no, una estación podría liberar un token libre después de haber finalizado su transmisión, pero antes de que reciba su propio token ocupado. En cualquier caso, el uso de un token garantiza que sólo una estación en un tiempo dado puede transmitir.

Cuando una estación libera un nuevo token, la estación anterior que tiene datos para enviar será capaz de capturar el token y transmitir.

Para superar varias situaciones de error tales como el que no haya token en circulación, así como un persistente token ocupado, una estación se designa como monitor del token activo el cual detecta la condición de pérdida del token, usando más tiempo que el requerido por la trama de mayor longitud, para completar la travesía del anillo.

Para recuperación, el monitor depura el anillo de cualquier dato residual y envía un token libre. Para detectar un token ocupado en circulación, el monitor pone el bit del monitor en 1 para cualquier token ocupado que pasa, si detecta un token ocupado con un bit ya establecido, sabe que la estación falló al depurar su trama. Así el monitor cambia el token ocupado a token libre.

Otras estaciones en el anillo tienen el papel de monitor pasivo. Su función primaria consiste en detectar fallas al monitor activo y asumir dicho papel.

Resumiendo, el protocolo Token-Ring está orientado a una topología anillo, en el medio de transmisión circula una señal de control llamada token, cuando está libre una estación puede transmitir poniéndole el token ocupado después cuando vuelve a regresar le coloca libre y otra estación puede atrapar el token y transmitir. A nivel físico son utilizados cables blindados conteniendo dos pares trenzados balanceados.

La figura 8, muestra una red local Arcnet con protocolo token-passing y topología en forma de anillo pero con una pequeña variante que hace ver a la red con una forma de tipo estrella.

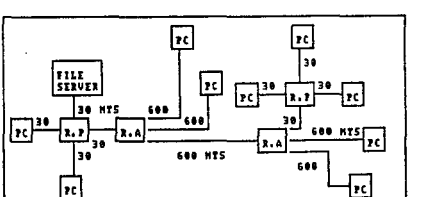


FIGURA 8. RED ARCNET CON CABLE COAXIAL

1.4 VENTAJAS DE LAS REDES LOCALES

A través de la lectura de este capítulo, hemos podido darnos cuenta de la gran capacidad y ventajas de las redes locales.

En este punto se explicarán algunas de las ventajas que ofrece operar una red de área local.

- **Facilidad** de compartir recursos, se comparten losrecursos caros: las grandes memorias, las impresoras, los plotter, etc; así como la información tal es el caso de: datos de computadora, voz, información de audio, video, y facsimile.

- **Diversidad** de equipo, el equipo que se conecta puedeser de diferentes fabricantes, es decir una red local provee el potencial para conectar dispositivos de múltiples vendedores, los cuales pueden dar al cliente gran flexibilidad y poder de negocio.

- **Aprovechamiento de los recursos**, se adquiere un mejoraprovechamiento de los recursos de sus computadoras compartiendo máquinas de altas y bajas capacidades con el disco duro, la interrelación de las minis con los mainframes, de sistemas operativos y otros paquetes de administración, la conexión PC's, minis o mainframes y con server que tienen la capacidad de ofrecer otros servicios como correo electrónico.

  El correo electrónico está apenas despegando: La mayor parte de las LAN se venden con correo electrónico como parte del paquete, y cada vez más y más negocios están incluyendo esta fácilidad.

  Al llevar el correo electrónico a la LAN y aprovechar lo que esencialmente es una capacidad no usada, las compañías liberan espacio y recursos en sus minis y macrocomputadoras; y en el proceso, se ahorran miles de pesos en costos de comunicación y cómputo. Los ahorros de costo en computación caen debido a que hay menos usuarios que tienen sesiones largas con una anfitriona central a través de un canal de telecomunicaciones.

- **Alta recuperación de información**, con las redes se tiene la posibilidad de recuperar todas las informaciones generadas en cualquier campo de trabajo aún cuando éste sea muy vasto.

- **Acceso por varios usuarios a archivos, base de datos a cualquier software de la red.**

- **Mandar impresiones o archivos de datos de un dispositivo a otro.**

- **Comunicación con otras redes y usuarios.**

- **El costo de los requerimientos para conectar una red de área local es menor a los costos de los dispositivos de comunicación entre sí.**

- **Brinda alta seguridad**, con la existencia de poderosos y seguros sistemas operativos para red.

- **Ayuda a la estandarización del software**, ya que los programas de aplicación residen en un server, y todos los usuarios deben accesar sus programas desde él, entonces podemos reglamentar mejor uso del software

- **Proceso distribuído**, esta característica marca una de las ventajas de una red sobre un multiusuario, esto es debido a que el proceso real se ejecuta individualmente en cada máquina, a diferencia de que los procesos se desarrollen en la máquina central lo que causa una disminución notable de la eficiencia del sistema a medida de que éste crece.

  Esto significa que la ejecución de las tareas no está a cargo de la máquina central sino que cada estación de trabajo es lo suficientemente inteligente para realizar cualquier tarea sin necesidad de accesar al server a menos que requiera alguna lectura al disco o mandar una impresión, debido a esta ventaja, el tiempo de respuesta en la ejecución de un programa se puede incrementar variando la velocidad de las estaciones sin necesidad de cambiar el server.

- **Remplazar aplicaciones o sistemas**, evitando el "todo o nada", cualquier faceta de sus capacidades es que el viejo equipo puede quedarse en el sistema que corre una sóla aplicación, si el costo de movimiento que la aplicación da a una nueva máquina no es justificada.

1.5 NOVELL- NETWARE COMO SISTEMA OPERATIVO PARA RED.

Se puede decir que el Sistema Operativo de una red, es el conjunto de programas que regulan el funcionamiento de la misma, proporciona los elementos para la interface con el usuario, controla y define los niveles de seguridad, se encarga de la integridad y seguridad de la información, controla como se comparten los recursos, etc.

Se dice que que el sistema operativo de red está ahora reconocido como el componente más importante de una red.

La empresa NOVELL desarrolló originalmente el Netware como el sistema operativo para el equipo NOVELL-S NET. Una red que utiliza una topología de estrella y un servidor propietario basado en el microprocesador 68000, el cual no tenía ningún sistema operativo estandar, NOVELL decidió desarrollar el suyo partiendo de cero, y lo optimizó para redes; diseñando de paso todas sus características, alrededor de la funcionalidad de la red.

Cuando comenzó el éxito de las PCs, los autores de Netware, viendo que este software está escrito con C, podría fácilmente convertirse a la arquitectura de la familia Intel 8080 y que podría soportar virtualmente cualquier red en el mercado.

Debido a que el ROM BIOS /4 de la IBM PC XT, fue diseñado para un sistema operativo (DOS) de un solo usuario, con la deficiencia para ambientes multiusuario y ya que el Netware es particularmente multiusuario, los programadores de Netware decidieron ignorar el ROM BIOS y optaron por comunicarse directamente con el hardware, para eliminar efectivamente cualquier limitación. Lograron con ello permitir a Netware procesar requerimientos de otra estación de trabajo.

Novell maneja, en su sistema operativo de red Netware cuatro niveles que tienen entre si pocas variaciones.

NIVEL 1: **NETWARE ELS I (Entry Level System)**

ELS I contiene las siguientes características:

- Proceso Distribuído.
- Manejo de Seguridades.
- Manejo Optimizador de la Memoria Extendida.

/4 READ ONLY MEMORY BASIC INPUT-OUTPUT SYSTEM

Netware al ser un sistema operativo no basado en DOS si permite el uso de la memoria extendida, y ésta la aprovecha en el server para optimizar el uso del disco de la red, para este fin Netware contempla varios elementos:

. Cache del directorio.

Esto significa que tanto el directorio como el FAT (tablas de localización de archivos) permanecen en RAM todo el tiempo con lo cual se deja de perder tiempo en las lecturas para la ubicación de los archivos dentro del disco.

. Cache para archivos.

Por medio de esta opción se logra almacenar momentáneamente en RAM (Memoria de Acceso Aleatorio) aquellos archivos que estén en transito ya sea del disco hacia el usuario o del usuario hacia el disco, con ello se liberan más rápidamente las estaciones de trabajo cuando mandan escribir archivos al server, por otra parte cuando la memoria es suficiente algunos archivos que estadísticamente están ocupando más permanecen en RAM para que las próximas lecturas no requieran del disco para realizarlas.

- Manejo de Passwords.

NIVEL 2: NETWARE ELS II

Esta versión posee todas las características del nivel anterior aún cuando tiene aumentados los menús y los comandos de línea.

- Mayor Número de Usuarios.

Adicionalmente al aumento de menús y comandos, también se tienen en este nivel un número mayor de usuarios a controlar, en este caso son ocho los usuarios concurrentes que pueden trabajar.

- Conexiones Remotas.

Para este nivel ya es posible colocar una estación remota por medio de la declaración de un puente interno que realice la conversión de protocolos locales en protocolo asincrono que permita el control de la estación remota conectada a la red por medio de modems y líneas telefónicas ya sean públicas o privadas.

En el caso de que se requieran puentes internos locales o más de ocho estaciones de trabajo simultáneo, es necesario pensar en la utilización del siguiente nivel de Netware.

Este nivel de Netware es ya versión 2.1x por lo que goza de todas las ventajas de contabilización de recursos y limitación por cuenta del tiempo de acceso y recursos utilizados.

NIVEL 3: ADVANCED NETWARE 286

Esta versión posee también todas las características del nivel anterior aún cuando tiene aumentados los menús y los comandos de línea.

- Mayor Número de Usuarios.

Para este nivel ya se tiene un límite de estaciones concurrentes de cien por lo que ya no hay ningún problema de crecimiento a corto o mediano plazo, ya que esta versión soporta hasta cuatro puentes internos, y se puede contemplar la posibilidad de instalar más de un server configurado en redes mayores.

NIVEL 4: ADVANCED NETWARE 286 SFT.(System Fault Tolerant)

Esta versión posee como es de esperarse las características del nivel anterior, aún cuando tiene mejoras en el aspecto de la seguridad de la información.

- Tolerancia a Fallas.

Adicionalmente este nivel posee una protección por software que asegura la integridad de las bases de datos, es decir, suponiendo el caso de una falla eléctrica, o que el server sea apagado incorrectamente y se estaban alterando diferentes registros de una base de datos tal como una contabilidad al dar de alta una póliza de varios movimientos, qué pasará si cualquiera de estos errores sucediera a media actualización?, la recuperación de los registros alterados puede llevar varias horas, sin embargo con la opción TTS (Sistema de Registro de Transacciones) que tiene este nivel, la recuperación de los registros es instantánea y se realiza en el momento de prender el server, para que todo esto se lleve a cabo solamente es necesario indicarle al sistema operativo donde termina, si algo sucede entre el inicio de una transacción (actualizaciones de archivos) y el final de ésta tal que no se puede terminar, entonces todos aquéllos registros alterados regresan a su valor anterior en el momento en que se detecta que la transacción no llegó a un final exitoso.

NIVEL 5: NETWARE 386.

Se había dicho que únicamente existían cuatro niveles de Novell sin embargo nace netware 386 para los procesadores 386 y 486, es muy rápido, no se degrada bajo pesadas cargas de procesamiento y provee enormes cantidades de almacenamiento, puede manipular varias peticiones de entrada-salida de discos simultáneamente y acomodar archivos de 4 Gb (gigabytes) permitiendo que los volúmenes se extiendan a múltiples discos, permite a cada servidor 100,000 archivos abiertos y hasta 250 usuarios concurrentemente.

Otros fabricantes pueden añadir funcionalidad a Netware 386 mediante programas llamados Netware Loadable Modules(NLMs) que pueden residir en el servidor. Los NLMs incluyen muchos tipos de aplicaciones que varían desde servidores de base de datos a servicios de monitores y administración.

Hoy existe la versión 3.1 extendiendo la evolución al incrementar la eficiencia y al proveer las herramientas necesarias para crear poderosas aplicaciones basadas en el servidor y preparadas para la red.

NETWARE 3.1 incluye carga automática de los NLMs, permitiendo a un servidor retornar a su estado operacional sin intervención del usuario después de un fallo eléctrico. El software en las estaciones de trabajo también apoya la memoria expandida y extendida para dejar más espacios en la estación para las aplicaciones de DOS añade una opción de servidor preferido que permite al administrador de la red seleccionar el servidor donde los usuarios arrancan, a diferencia de las versiones anteriores que automáticamente seleccionan el primer servidor que corresponda, añade dos mejoras en la seguridad: la auditoría de seguridad y los resguardos cifrados. La primera característica mantiene un rastro de auditoría no modificable de todos los cambios de seguridad que ocurren en el servidor. Y a medida que Netware hace copias de los archivos en la red, la información se envía y se guarda en forma cifrada y solamente descifra cuando regresa al servidor después de recuperar archivos, en fin la empresa NOVELL se ha consagrado como el número uno en sus sistemas operativos para red.

2. INTRODUCCION A LAS BASES DE DATOS

La expresión base de datos comenzó a popularizarse al principio de 1960, trayendo a partir de ese momento una serie de desarrollos que han venido a evolucionar el procedimiento de la información en todas las esferas de la vida, en todas las áreas de la industria y el comercio. Y han ampliado en gran medida las posibilidades de acciones abiertas al hombre.

Algunos términos importantes para adentrar en el conocimiento de las bases de datos son :

Archivo. Un archivo es una colección de datos en un disco al que se tiene acceso por un nombre singular. Suele contener una secuencia de registros de formato idéntico, cada uno conteniendo una serie de campos. Podría contener un diccionario de datos, un índice, una disposición de pantalla o una combinación de todos ellos.

Registro. Un registro es un grupo de campos afines de información considerados como una unidad. Cada fila en la tabla 1 es un registro.

Campo. Un campo identifica una posición en un registro en donde un dato elemental es almacenado. Este campo tiene ciertas características, tales como largo y tipo (una cadena de caracteres, tipo numérico, etc.) y se representan mediante una columna en la tabla 1.

	CAMPO	CAMPO	CAMPO	CAMPO	CAMPO
REGISTRO1	NOMBRE	CALLE	CIUDAD	ESTADO	CLAVE
REGISTRO2	NOMBRE	CALLE	CIUDAD	ESTADO	CLAVE
REGISTRO3	NOMBRE	CALLE	CIUDAD	ESTADO	CLAVE

TABLA 1

Esquema. Es el conjunto de información que describe a la base de datos cuando está organizado de manera formal.

Subesquema. En una gran base de datos, el esquema mismo 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. Estas incluyen:

- 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.

Diccionario de datos. El diccionario de datos es una descripción completa de los campos en una base de datos. Este diccionario describe las relaciones entre los diversos campos en una base de datos. También describe cada campo por nombre, membretes de encabezamiento de informe, la extensión, tipo de datos, límites alto y bajo, etc.

Indice. Un índice es una tabla de números de registros. Estos números llamados punteros están dispuestos para ayudarle a encontrar rápidamente un registro particular mediante una clave. El índice permite la recuperación de registros en orden de secuencia y permite la inserción de nuevos registros.

Clave. Una clave es un identificador singular de un registro. Puede ser un campo único o un grupo de campos.

Transacción. Cálculo que se permite sea procesado tán rápidamente como el hardware disponible lo permita.

El Término computación. se emplea para denotar la sección de una aplicación que maneja la base de datos.

La mayoría de las computaciones que se emplean para manejar un conjunto de datos son conceptualmente simples. Reconocemos cuatro tipos de computaciones relacionadas con bases de datos:

1. La construcción del conjunto de datos.
2. Actualización de los campos en el conjunto de datos.
3. Recuperación de datos del conjunto de datos.
4. Reducción de grandes cantidades de datos a forma utilizable.

Una aplicación de base de datos utilizará las cuatro clases de computaciones, pero en ciertas aplicaciones lo harán más con un tipo que con los otros.

Se dice que la única computación que emplea los poderes de cálculo de la computadora es la reducción de datos para obtener la información.

La construcción de una base de datos incluye la obtención de los datos, su codificación y su captación. A menudo este tipo es la parte más costosa de una operación de bases de datos.

La actualización de una base de datos incluye colocar nuevos datos, modificar, cuando sea necesario valores de los ya almacenados y eliminar los que no son válidos.

La recuperación de los datos puede consistir en tomar un elemento específico para obtener un valor o hecho almacenado, o en la recolección de una serie de elementos relacionados para obtener datos referentes a alguna relación que se manifieste al unirlos.

Para tomar o traer un registro específico, se entrará a la base de datos mediante un argumento de búsqueda, el que se comparará con una llave de los registros. Algunas veces el argumento de una búsqueda se denomina llave de búsqueda.

La reducción de datos resulta necesaria cuando el volúmen de los datos importantes excede la capacidad de los solicitantes. Cuando la información deseada esta difundida en toda la base de datos, puede resultar necesario el acceso a la mayor parte de su contenido, para concentrar o abstraer los datos. Los resumenes estadísticos, las declaraciones anuales de operación de negocios o las presentaciones gráficas de datos, son ejemplos de técnicas de reducción de datos frecuentemente utilizadas. La reducción de datos exige mucho del desempeño de una base de datos.

Entrando a la definición de base de datos, hay quienes conciben a ésta, como una gran piscina en donde los ejecutivos pudieran ir de pesca de cualquier información que desearan. Esta gran piscina puede estar concentrada en una localidad determinada o distribuída en varias, todas ellas posiblemente interconectadas mediante un sistema de telecomunicación, y teniendo acceso a la base de datos y a programas de la más diversa índole.

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 o más aplicaciones, de la mejor manera posible; los datos se almacenan de modo que resulten independientes de los programas que los usan; se emplean métodos bien determinados para incluir datos nuevos y para modificar o extraer los almacenados.

Concretamente, la base de datos es la colección completa de datos, punteros, tablas, índices, diccionarios, etcétera.

En las organizaciones más sencillas, encontramos casi siempre una colección de registros organizados para una aplicación determinada. La idea básica en la implantación de una base de datos es de que los mismos deben ser aprovechados para tantas aplicaciones como sea posible.

Por eso, la base de datos se concibe a menudo como un repositorio donde se reúne la información necesaria para el ejercicio de las funciones propias de un organismo gubernamental, una empresa, una fábrica o cualquier otra organización.

La base de datos permite no sólo la lectura de los datos almacenados, sino la continua modificación de los que son necesarios para el control de las operaciones. Es posible inspeccionar la base de datos en busca de la respuesta a los interrogantes que se plantean o de información para fines de planeamiento sin tener en cuenta algunas fronteras administrativas.

La base de datos se utiliza en la programación de aplicaciones tales como inventarios, facturaciones, reservas de pasajes en líneas aéreas y contabilidad. Los beneficios principales de la utilización de la base de datos, son los ahorros de costos y de tiempo en la instalación de sistemas, para luego cambiarlos.

Un sistema de administración de base de datos puede organizar, procesar y presentar los datos seleccionados de una base de datos. Esta capacidad permite a quienes toman decisiones, rastrear, probar y consultar el contenido de la base de datos, para atraer las respuestas a las preguntas no recurrentes, y no previstas en los informes regulares de manera rápida y eficaz.

2.1 CARACTERISTICAS

Una de las características más importantes de la mayoría de las bases de datos es la de mantenerse en plena crisis de cambio y crecimiento. La base de datos debe prestarse a una fácil reestructuración siempre que haya que agregarle nuevos tipos de datos o utilizarla para nuevas aplicaciones. Esta restructuración no debe originar la necesidad de volver a escribir los programas de aplicación y, en general, no debe ser fuente de transtornos. La facilidad con que pueda modificarse la base de datos tendrá siempre un efecto directo sobre la capacidad para desarrollar nuevas aplicaciones del procesamiento de datos dentro del organismo que la explota.

Algunas características deseables se presentan a continuación:

- Independencia de datos. A menudo se habla de la independencia de datos como una de las características destacadas de las bases de datos. Existe la independencia física de los datos, asi como la independencia lógica. En la primera el hardware de almacenamiento y las técnicas físicas de almacenamiento son modificados sin obligar a la modificación de los programas de aplicación, en la segunda, se pueden agregar nuevos datos, o expandir la estructura lógica, sin que sea necesario reescribir los programas de aplicación existentes. Esta idea implica que los datos y los programas de aplicación que de ellos se sirven son mutuamente independientes, de manera que unos u otros puedan ser modificados sin afectar a los restantes. En particular, el programador de aplicaciones no debe ser afectado por los cambios que se introduzcan en los datos, en su organización, o en los dispositivos físicos donde se almacenan.

- Tiempo de respuesta adecuado. Las bases de datos diseñadas para ser usadas por operadores de terminal deben asegurar un tiempo de respuesta adecuado para el diálogo entre el hombre y la terminal. Además, el sistema de base de datos debe tener capacidad para manejar un adecuado caudal de transacciones.

  En los sistemas en que el volúmen de tráfico es reducido, el caudal de transacciones no tiene porqué imponer muchas restricciones al diseño de la base de datos. En cambio, en los sistemas de alto volúmen de tráfico, por ejemplo los de reserva de plazas en las aereolíneas, el caudal de transacciones tiene una gran influencia sobre la organización del almacenamiento físico.

- **Redundancia mínima.** En la mayoría de las bibliotecas de cintas o discos anteriores al advenimiento de la técnica de las bases de datos, hay una sorprendente cantidad de datos duplicados o redundantes. Muchos se hallan simultáneamente almacenados en varios volúmenes con distintas finalidades y también con diferentes fechas de actualización. En la base de datos se pretende eliminar esta redundancia. Ella ha sido definida como una colección no redundante de datos, pero, en realidad, en muchas de ellas se admite cierta redundancia con el objeto de reducir los tiempos de acceso o simplificar los métodos de direccionamiento. Algunos registros se duplican para facilitar la reconstrucción de la base en caso de daño accidental. Hay, pues, necesidad de armonizar el grado de redundancia con otras características deseables de la base, de modo que es preferible hablar de redundancia controlada o redundancia mínima en lugar de no redundancia como criterio de diseño. En otros términos, en una base de datos bien diseñada se evita la redundancia perjudicial o innecesaria.

  La redundancia no controlada acarrea varios inconvenientes. En primer lugar tenemos el costo adicional del almacenamiento de copias múltiples de los mismos datos. En segundo término, y esto es mucho más serio, el hecho de que para actualizar por lo menos una parte de las copias redundantes es preciso recurrir a múltiples operaciones de actualización.

  La redundancia, por lo tanto, es más onerosa en los archivos que exigen una actualización permanente, o peor, en los que es frecuente la inclusión de nuevos datos y la eliminación de los ya obsoletos. Por último, debido a que las distintas copias pueden hallarse en diferentes estados de actualización, el sistema tiende a proporcionar informaciones contradictorias.

- **Capacidad de búsqueda.** El usuario de una base de datos suele plantear muchos interrogantes acerca de los datos almacenados. En la mayoría de las aplicaciones comerciales actuales, se han anticipado los tipos de averiguación y los datos se han organizado físicamente de modo de poder satisfacerlas con adecuada prontitud. Hay una creciente demanda para sistemas que sean capaces de atender consultas o producir informes que no han sido previstos detalladamente en la época del diseño. El usuario quiere presentar pedidos espontáneos de información en una terminal. Las averiguaciones no anticipadas hacen necesario explorar algunas partes de la base de datos. Si en la terminal se pretende una respuesta inmediata la exploración tiene que ser rápida.

La capacidad para explorar una base de datos rápidamente y con diferentes criterios de búsqueda depende mucho de la organización física de los datos.

Con algunas organizaciones los tiempos de búsqueda son excesivamente prolongados para poder considerar las respuestas como de tiempo real. Además los datos estan disponibles para los usuarios casi todas las veces que los necesitan.

- Integridad. Cuando una base de datos incluye información utilizada por muchos usuarios, es importante que no puedan destruirse los datos almacenados ni las relaciones que existen entre los distintos datos. Ocasionalmente se producirían fallos de hardware y diversos tipos de accidente.

  El almacenamiento de los datos y los procedimientos de actualización e inserción deben asegurar que el sistema pueda recuperarse de estas contingencias sin daño para los datos. Toda instalación debe garantizar la integridad de la información que almacena.

  Además de proteger los datos contra posibles problemas sistémicos deben incluirse también procedimientos de chequeo que aseguren que los valores de los datos se ajusten a ciertas reglas prescritas de antemano. (en el punto 2.3 se estudiará a detalle esta característica)

- Seguridad de los datos. Los datos albergados en una base de datos deben ser conservados con seguridad y reserva. La información almacenada es a menudo de gran valor para la empresa. Cuando más vital es la información almacenada, tanto más importante es protegerla contra los fallos del hardware o el software, contra las catástrofes, y contra personas incompetentes que pretendan dar un uso ilegítimo.

Los siguientes siete requisitos son escenciales para la seguridad de la base de datos :

1. La base de datos debe estar protegida contra el fuego, el robo y otras formas de destrucción.

2. Los datos deben ser reconstruibles, porque por muchas precauciones que se tomen, siempre ocurren accidentes.

3. Los datos deben poder ser sometidos a procesos de auditoría.

4. El sistema debe diseñarse a prueba de intromisiones. Los programadores, por ingeniosos que sean, no deben poder pasar por alto los controles.

5. Ningún sistema puede evitar de manera absoluta las intromisiones mal intencionadas, pero es posible hacer que resulte muy difícil eludir los controles.

   Los usuarios de la base de datos deben ser sometidos a un proceso de identificación positiva antes de tener acceso a ella.

6. El sistema debe tener capacidad para verificar que sus acciones han sido autorizadas.

7. Las acciones de los usuarios deben ser supervisadas, de modo tal que pueda descubrirse cualquier acción indebida o errónea.

- **Afinación.** En muchos caso, el uso de la base de datos evoluciona contínuamente, a medida que más personas se van familiarizando con ella y se crean más programas de aplicación. El ajuste de la organización del almacén con el objeto de mejorar su desempeño se convierte así en un proceso contínuo, llamado afinación.

  La correcta afinación tiene dos requisitos. Primero, necesita la independencia física de los datos. Segundo, requiere medios para supervisar automáticamente el uso de base de datos con el fin de que puedan hacerse los ajustes necesarios.

- **Bajo Costo.** Bajo costo de almacenamiento y el uso de los datos y minimización del costo de los cambios.

- **Flexibilidad.** Los datos pueden ser utilizados o explorados de manera flexible, con diferentes caminos de acceso. (En el punto 2.3 se estudiará a detalle esta característica)

- **Recuperación de información.** La recuperación de información se realiza al aplicar consultas a bases de datos. Las aplicaciones de recuperación de información son aquellos sistemas que proporcionan servicio operativo regular, por ejemplo, sistemas que manejen cronológicamente actividades, manejan inventarios o preparan facturas o nóminas. Las aplicaciones de base de datos en áreas tales como reservaciones de líneas aéreas y registros de habilidades se acercan a las aplicaciones de recuperación de información.

- **Adecuada rápidez de acceso y de exploración.** Los mecanismos de acceso y los métodos de direccionamiento son lo suficientemente rápidos, de acuerdo a los usos previstos.

  La conveniencia y necesidad de la exploración espontánea se incrementa en la medida que se difunde el uso interactivo de los sistemas.

- **Claridad y facilidad de uso.** Los usuarios tienen fácil acceso a los datos que se encuentran a su disposición y los comprenden sin dificultad.

- **Interface de alto nivel con los programadores.** Los programadores de aplicaciones disponen de medios sencillos para pedir datos y estar aislados de las complejidades internas de organización y direccionamiento de los archivos.

2.2 TIPOS DE BASES DE DATOS.

En este punto se analizarán los diferentes tipos de base de datos que se definen a continuación :

.BASE DE DATOS JERARQUICA.
.BASE DE DATOS CON CAPACIDAD DE RED.
.BASE DE DATOS RELACIONAL.
.BASE DE DATOS ORIENTADA AL OBJETO.
.BASE DE DATOS DISTRIBUIDA.

BASE DE DATOS JERARQUICA.

Por definición, un árbol está compuesto por una jerarquía de elementos denominados nodos. El nivel más alto de la jerarquía tiene un solo nodo, el que se llama raíz.

Con excepción de la raíz, todo nodo está vinculado a otro nodo de nivel más alto llamado padre. Ningún elemento puede tener más de un padre. En cambio, todo elemento puede tener uno o más elementos relacionados, en un nivel más bajo, éstos son los hijos. Los elementos que se encuentran en las puntas de las ramas se llaman hojas, ver figura 1.

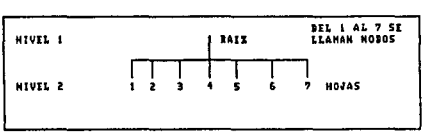


FIGURA 1. ARBOL

Todo árbol puede ser descrito como una jerarquía de nodos con relaciones binodales de tal modo que:

1. El más alto nivel de la jerarquía tiene un sólo nodo.

2. Los nodos restantes se reparten en m > 0 conjuntos disjuntos (es decir, no conectados) T1...Tm y cada uno de estos conjuntos constituye a su vez un árbol. Los árboles T1...Tm, se llaman subárboles de la raíz.

En un árbol es difícil definir operaciones relacionales, tales como reducir el número de atributos o componer una nueva relación que combina los campos provenientes de dos relaciones y rara vez se implantan en estos sistemas.

Sin embargo, en la mayoría de los casos es posible volver a plantear consultas de relación en forma de estrategias de búsqueda en árboles o transformar consultas jerárquicas no ambiguas en consultas equivalentes relacionales.

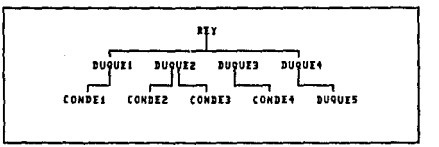


FIGURA 2.

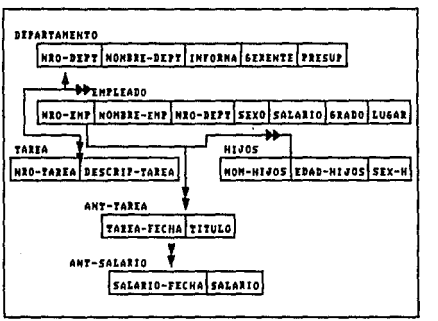


FIGURA 3. ESQUEMA PARA UN ARCHIVO JERARQUICO DE MULTIPLES NIVELES.

Un beneficio operacional de los árboles es que las llaves se almacenan sólo en un lugar, por lo que son más fáciles de actualizar.

Las figuras 2 y 3 ilustran ejemplos de estructuras ramificadas (árboles) cuyos nodos son agregados de datos, segmentos o registros.

Las estructuras de árbol implican por lo general que hay una correspondencia simple de hijo a padre de modo que la correspondencia inversa es compleja, como en la figura 2.

La figura 4 ilustra un esquema y una instancia del mismo esquema para una estructura en árbol simple, de sólo dos niveles.

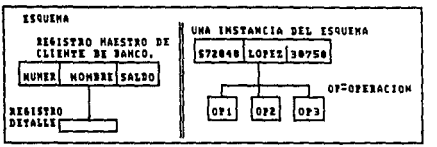


FIGURA 4. UN ARCHIVO JERARQUICO CON SOLO DOS TIPOS DE REGISTRO.

Ocasionalmente hay correspondencia simple en los dos sentidos, como ocurre en la figura 5, donde la estructura se refiere a los registros asociados con una misma entidad, los que se almacenan por separado.

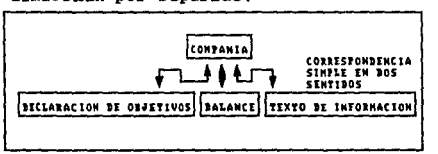


FIGURA 5.

La principal ventaja de este tipo de estructuras es la no redundancia del almacenamiento. El registro objetivo está asociado directamente con su llave. Los campos de llave aparecen sólo una vez y no hay réplicas de ellos en otros niveles, sin embargo, una desventaja del archivo con estructura de árbol es que la amplitud puede reducirse en forma significativa, de manera que pueden resultar necesarias muchas más localizaciones para recuperar un registro promedio de un gran archivo, agregando que el tiempo de recuperación en una estructura de árbol no está restringido por la consideración en números enteros de nivel.

El término archivo jerárquico se aplica a los archivos que exhiben relaciones de tipo árbol entre sus registros. La figura 4 muestra un archivo maestro de <<Detalles>>, un tipo de archivo jerárquico muy común, con dos tipos de registro. La figura 3 ejemplifica un archivo jerárquico de cuatro niveles.

Un archivo jerárquico está estructurado por los datos; el número de niveles y la amplitud se controlan en forma externa.

A fin de poder procesar en forma efectiva estructuras jerárquicas de datos, deben cumplirse ciertas condiciones estructurales. Una condición es que haya una amplitud razonable en cada nivel. El número de hijos en un registro no deberá ser tan pequeño que resulten necesarios demasiados niveles para encontrar los datos en un nivel. Ni deberá haber tantos hijos que la búsqueda de uno en especial resulte difícil. Otra condición es que un número igual de nivel implica un tipo idéntico de registros.

Para ejemplificar la definición de base de datos jerárquica, si se producen coches en una empresa, y sabemos que un coche esta compuesto por partes (motor,cuerpo,chasis), que a su vez se descomponen en subpartes (válvulas,cilindros,bujias) y luego en sub-subpartes, etc, nos damos cuenta que se forma una estructura jerárquica natural. Para almacenar estos datos, se desarrolló el modelo de datos jerárquico en el cual cada registro de la base de datos representa una pieza específica, teniendo estas relaciones padre/hijo, que liga cada pieza a su subpieza, y así sucesivamente.

Para acceder a los datos, un programa puede hallar una pieza en particular mediante un número o clave, descender al primer hijo, ascender hasta su padre y moverse de lado hasta el siguiente hijo. Para la recuperación de los datos, se requiere navegar a través de los registros, moviéndose hacia arriba, hacia abajo y hacia los lados un registro cada vez.

BASES DE DATOS CON CAPACIDAD DE RED

Una red se crea cuando se asignan estructuras más complejas que las jerarquías. Una jerarquía es la estructura más compleja que puede construirse dentro de un solo archivo. Si en una relación entre datos un hijo tiene más de un padre, la relación no puede ya ser descrita por medio de un árbol o estructura jerárquica. En una red cualquier componente puede vincularse con cualquier otro, es decir un hijo puede tener más de un padre.

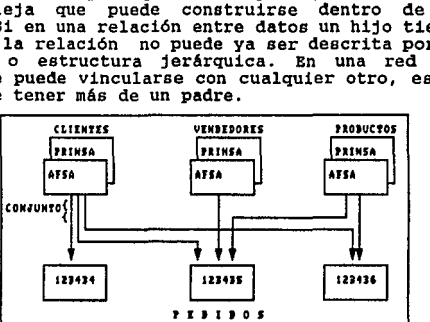


FIGURA 6. MULTIPLES RELACIONES PADRE/HIJO

Cada pedido puede participar en tres relaciones padre/hijo diferentes, ligando el pedido al cliente que lo remitió, al vendedor que lo aceptó y al producto ordenado. Estas relaciones se conocen como conjuntos en este modelo.

En la figura 7 se ilustra una red con cinco tipos de registros. Cada una de las relaciones representada se puede considerar como una relación de padre a hijo. El tipo de registro ORDEN DE COMPRA es un hijo del registro ARTICULO y un padre del tipo de registro RENGLON DE COMPRA.

Es conveniente distinguir entre una estructura en que la correspondencia de hijo a padre es simple o inexistente y aquella en que la correspondencia entre dos tipos de datos cualesquiera es compleja en los dos sentidos. En este último caso por lo menos una de las líneas del esquema tendrá doble punta de flecha en los dos sentidos.

Llamaremos red compleja a esta clase de estructuras. Por el contrario, llamaremos red simple a aquella que no tiene doble punta de flecha en los dos extremos de ninguna línea. El de la figura 7 es una caso de red simple, que se puede transformar en compleja si se utiliza la correspondencia entre ORDEN DE COMPRA y ARTICULO, porque una orden de compra corresponde por lo general a más de un artículo. Veáse figura 8.

Es posible que una red compleja tenga sólo dos tipos de registro, así lo apreciamos en la figura 9. El registo PROVEEDOR puede tener más de un hijo, porque está en condiciones de suministrar más de un tipo de artículo. A su vez, un registro ARTICULO puede tener más de un padre, porque son varios los proveedores que lo suministran.

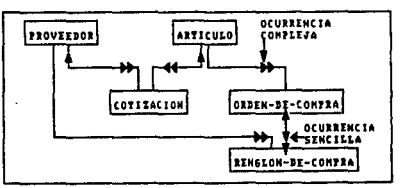


FIGURA 7. EN UNA APLICACION DE COMPRAS, SE USA UNA RED CON CINCO TIPOS DE REGISTRO.

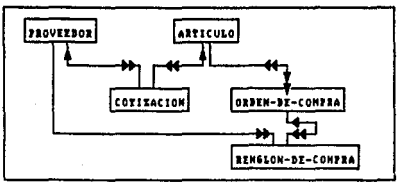


FIGURA 8. LA INTRODUCCION DE UNA CORRESPONDENCIA COMPLEJA ENTRE EL TIPO DE REGISTRO ORDEN-DE-COMPRA Y EL TIPO DE REGISTRO ARTICULO DA LUGAR A ESTA RED COMPLEJA.

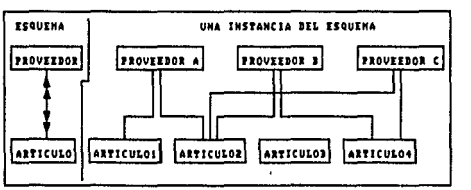


FIGURA 9. UNA RED COMPLEJA CON SOLO DOS TIPOS DE REGISTRO. LA UNICA RELACION TIENE CORRESPONDENCIA COMPLEJA EN LOS DOS SENTIDOS.

Para acceder a una base de datos de este tipo, un programa de aplicación puede hallar un registro padre específico mediante una clave por ejemplo un número de cliente, descender al primer hijo en un conjunto en particular, moverse lateralmente de un hijo al siguiente dentro del conjunto, ascender desde un hijo a su padre en otro conjunto.

Este tipo de bases ofrecen flexibilidad ya que permiten a una base de datos representar datos que no tengan estructura jerárquica sencilla, pero al igual que las bases jerárquicas resultan ser rígidas. Las relaciones de conjunto y la estructura de los registros tienen que ser especificados de antemano. Modificar la estructura de la base de datos requiere la reconstrucción de la base de datos.

El esquema CODASYL /1 permite la construcción de redes casi arbitrarias de relaciones conectadas. En la especificación del esquema DDL /2 se encuentran los dos componentes principales empleados para construir una base de datos CODASYL: REGISTROS y CONJUNTOS.

Es posible describir cualquier tipo de registro a la vez como miembro de un conjunto singular y como miembro de conjuntos más complejos. El tipo de registro CLIENTE de la figura 10, por ejemplo, podría ser declarado como miembro de un conjunto singular, tal como aparece en la figura 11. La secuencia definida para los registros de un conjunto singular no tiene por qué ser la misma que la que tienen los mismos registros como partes de otro conjunto, como veremos en breve.

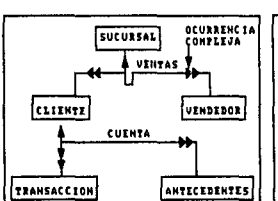


FIGURA 10. UN ARBOL DE TRES NIVELES SE DEFINE CON DOS CONJUNTOS. EL TIPO DE REGISTRO CLIENTE ES A LA VEZ PROPIETARIO DEL CONJUNTO CUENTA Y UNO DE LOS MIEMBROS DEL CONJUNTO DE VENTAS.

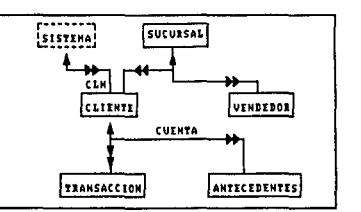


FIG.11. EL TIPO DE REGISTRO CLIENTE SE DECLARA AHORA COMO MIEMBRO DE UN CONJUNTO SINGULAR DENOMINADO CLN. ESTA DECLARACION PERMITE QUE LOS REGISTROS CLIENTE SE TRATEN COMO SI SE HALLASEN EN UN UNICO ARCHIVO SIMPLE.

Cada tipo de registro miembro tiene un padre, y solo uno, en el mismo conjunto: su tipo de registro propietario.

Pero el número de padres es arbitrario si cada uno de ellos pertenece a un conjunto distinto. A su vez, no hay limitación para el número de conjuntos de los cuales un tipo dado de registro es miembro. Por lo tanto, el lenguaje permite redes simples con una relación de uno a muchos entre padres e hijos. Si bien este lenguaje de descripción de datos permite describir sin dificultad las redes simples, no es posible describir con él redes complejas (con correspondencia compleja entre hijos y padres) sin convertir éstas, previamente, a formas más simples.

/1 LENGUAJE PARA LA DESCRIPCION DE DATOS, ADECUADO PARA DEFINIR ESQUEMAS.
/2 DATA DESCRIPCION LANGUAJE. DEFINIDO POR CODASYL.

Las redes son comunes en los problemas de "Factura de materiales". Es posible realizar el acceso a través de los enlaces a todas las partes necesarias para una Sección de Automóviles y si se presenta la falta de alguna parte, ir a través del conjunto de enlace a partes y obtener la lista de otro equipo que ocupe la misma parte. Entonces es posible consultar las otras Secciones de Automóviles de importancia en lo referente a las necesidades y a una posible demasía.

En un tipo red existen conexiones casi arbitrarias que asignan registros dato a registros sucesores y predecesores. El proceso de recuperación entra a la estructura empleando un argumento de localización y después navega a través de dicha estructura empleando operaciones de obtención del siguiente, obtención del propietario u obtención de miembro recolectando valores dato en el recorrido. Estos valores dato se entregan en forma incremental al procesamiento de consulta y se emplean para controlar el progreso de esta última. El resultado no se pone a la disposición como una unidad, ya que su estructura no es predecible.

BASES DE DATOS RELACIONALES

Es posible evitar el enmarañamiento a que dan lugar las estructuras ramificadas y de red recurriendo a una técnica que se llama normalización. La normalización es un proceso de paso a paso que permite reemplazar relaciones entre datos (del tipo jerárquico o red) con relaciones de la forma plana bidimensional, representando los datos en base a tablas las cuales deberán organizarse de forma tal que no se pierda ninguna de las relaciones existentes entre datos y las cuales regularmente son matrices rectangulares que pueden ser descritas matemáticamente, así, a una tabla como la de la figura 12 la llaman relación.

La base de datos construída por medio de relaciones y basada en matrices planas de datos es una base de datos relacional.

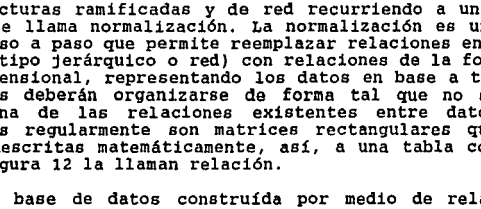


FIGURA 12.

La relación o tabla, es un conjunto de tuplas o registros. Si se trata de n-tuplas, esto es, si la tabla tiene n columnas, se dice que la relación es de grado n. Las relaciones de grado 2 se llaman binarias las de grado 3, ternarias, y las de grado n, enearias.
El conjunto de valores de un mismo tipo, esto es, cada columna de la relación, constituye por definición un dominio. La columna j-ésima es el dominio j-ésimo de la relación.

Los diferentes usuarios de la misma base de datos percibirán diferentes conjuntos de datos y diferentes relaciones entre ellos. Es por lo tanto necesario extraer de las columnas de las tablas los subconjuntos pedidos por algunos usuarios, creando así tablas de menor grado, o, por el contrario, a veces es preciso fundir dos o más tablas en una, creando una de mayor grado.

Estas operaciones se describen sin dificultad dado que algunas tablas tendrán muchas filas y columnas, es importante la capacidad para la extracción de subconjuntos. El separar y unir proporciona un grado de flexibilidad que no es posible lograr con la mayor parte de las estructuras de árbol o red.

Es posible que todos los archivos pueden ser representados como archivos planos. En primer término, un archivo sería <<plano>> si no existiera en él un determinado grupo repetitivo, se normaliza inmediatamente si se remueve ese grupo y se forma con él una tabla o archivo plano aparte, debidamente nominado, como se ve en la figura 13.

La normalización en este caso exige que algunos datos aparezcan en más de un registro con la finalidad de identificar éstos. Esta duplicidad no representa necesariamente un aumento de las necesidades de espacio de almacén, porque la normalización atañe a las estructuras lógicas -la vista de los datos propia del usuario- y no a la forma como los datos se representan físicamente.

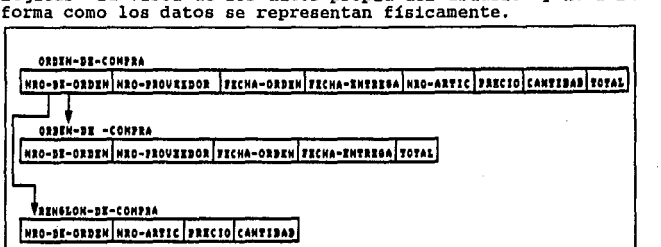


FIGURA 13. ELIMINACION DE UN GRUPO REPETITIVO POR PARTICION DEL ARCHIVO EN DOS RELACIONES.

Todo registro debe estar asociado con una clave que permita su identificación. A veces el registro se identifica mediante un único atributo. Por ejemplo, NRO-ORDEN en la figura 13 identifica los registros ORDEN-DE-COMPRA. En otras ocasiones, no obstante, hay que recurrir a más de un atributo para lograr la identificación inequívoca de un registro. Así, ningún atributo único es suficiente para identificar un registro RENGLON-DE-COMPRA en la figura 13, de modo que la clave deberá estar formada por la conjunción de dos atributos: NRO-ORDEN y NRO-ARTICULO y que se identifica subrayando el atributo.

La clave deberá cumplir dos requisitos:

1. Identificación unívoca: En cada registro de una relación, el valor de la clave debe identificar unívocamente ese registro.

2. No redundante: Ningún atributo de la clave podrá ser descartado sin destruir la propiedad de identificación unívoca.

En cada registro puede existir más de un conjunto de atributos capaz de satisfacer estos dos requisitos. Esos conjuntos se denominan claves candidatas y de ellas una debe ser designada como clave primaria usada para identificar el registro. Cuando hay opción, la clave primaria se elegirá, primero, de modo que ninguno de los atributos que la forman tenga un valor indefinido y, segundo, de modo que el número de atributos sea el mínimo.

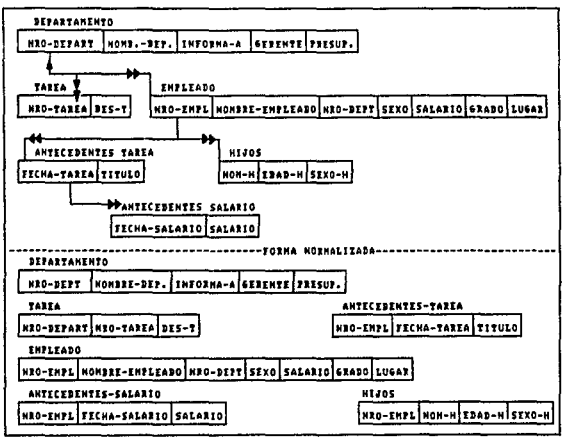


FIGURA 14.

También es posible normalizar un árbol y una red. La figura 14 ilustra un árbol de cuatro niveles y la manera de reemplazarlo por medio de seis relaciones. La clave de cada relación puede incorporar la de la relación precedente en el árbol. ANTECEDENTES-TAREA, por ejemplo, tiene como clave (NRO-EMPLEADO, FECHA-TRABAJO) en la que se incluye el dato precedente (NRO-EMPLEADO). La relación EMPLEADO, en cambio, tiene una clave simple (NRO-EMPLEADO) porque ésta es suficiente para identificar los registros de EMPLEADO y no es preciso por eso incorporar la clave de la relación precedente.

Existe una forma de representación por medio de líneas con puntas de flecha, por ejemplo, si las relaciones de grado 2 son líneas con puntas de flecha en la descripción lógica, esta transformación resulta difícil de representar. Cuando hay líneas con puntas de flecha que van de un bloque a otro y prosiguen a un tercero, el usuario tiende a seguir líneas y a presumir que ellas representan relaciones ternarias, resultando esto inválido. Consideremos, el caso 1 de la figura 15. Hay flechas que van de ARTICULO 4 a SUBCONJUNTO C y desde aquí a PRODUCTO 5. El usuario inadvertido podría pensar entonces que el PRODUCTO 5 contiene el ARTICULO 4 y estaría en lo cierto. Consideremos ahora el caso 2 de la misma figura.

El usuario puede creer que el PRODUCTO 5 tiene como uno de sus proveedores el 4, pero esta vez estaría equivocado. El SUBCONJUNTO C utilizado en el PRODUCTO 5 puede ser provisto por el PROVEEDOR 2 y no por el PROVEEDOR 4, mientras que el mismo subconjunto para el PRODUCTO 3 únicamente sería provisto por el PROVEEDOR 4.

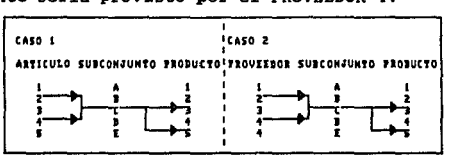


FIGURA 15. TRAMPA DE CONEXION.

Esta situación exige para su aclaración relaciones ternarias del tipo siguiente:

CASO 1			CASO 2		
ARTICULO	SUBCONJUNTO	PRODUCTO	ARTICULO	SUBCONJUNTO	PRODUCTO
2	C	3	2	C	5
2	C	5	4	C	3
4	C	3			
4	C	5			

Podemos decir, que la representación relacional es mucho mejor que las líneas con puntas de flecha, debido a, que es flexible, fácil de comprender y matemáticamente rigurosa si se la ejecuta debidamente.

El secreto de la flexibilidad de las bases de datos relacionales reside en la facilidad con que las relaciones se prestan a <<separarse o unirse>>, por medio de operadores como, π donde la relación EMPLEADO se proyecta mediante el enunciado EMP para formar una nueva relación llamada EMPL, representada en la figura 16.

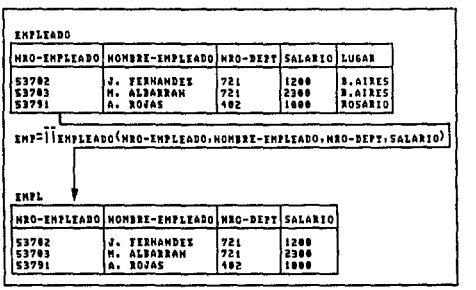


FIGURA 16.

La inversa de partir relaciones es la operación de conectar diferentes relaciones formando una unión natural. El símbolo * representa el operador UNION. La figura 17 muestra una operación de unión natural ejecutada sobre tres relaciones. La relación resultante puede ser mucho más pequeña que las relaciones que se unen.

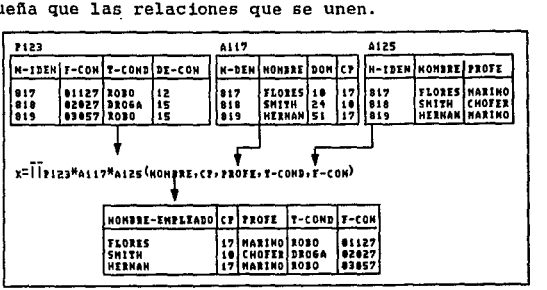


FIGURA 17. UNION NATURAL.

Al ejecutar una operación de unión hay que tomar algunas precauciones para asegurarse de que el resultado será válido. Es muy posible ejecutar operaciones de esta clase con relaciones que semánticamente no pueden ser unidas.

Las notaciones π y *, son propias del álgebra relacional. Una operación del álgebra operacional tiene como operandos una o más relaciones y las manipula de modo que forman una nueva relación. Una manera más automática de atacar el problema consiste en acudir a lo que se llama cálculo relacional. En este caso, el usuario se limita a definir el resultado que desea y deja al sistema el decidir qué operaciones se requieren para obtener ese resultado a partir de la base de datos. El cálculo relacional es una notación que sirve para definir una relación que ha de derivarse de las relaciones ya existentes en la base de datos. Los símbolos más comunes del cálculo relacional son:

x.v	El conjunto de los valores de datos en el dominio y de la relación x.
$A(X_1.Y_1.X_2.Y_2..)$	Una relación llamada A formada por los dominios que contienen los conjuntos de valores X1.Y1 .
:	<<De modo que>> La expresión escrita a la izquierda de los dos puntos indica qué es lo que debe obtenerse, mientras que la expresión escrita a la derecha es un calificador.
E	<<Existe>>
∟	<<Para todo>>
∩	<<Y>> Valen a la vez las condiciones unidas por este símbolo.
V	<<O>> Una cualquiera de las condiciones unidas por este símbolo.
¬	<<No>> No es aplicable la condición escrita a la derecha del símbolo.
=, <>, < >	Igual a, no igual a, menor que, mayor que.
`x`	El valor literal de x.

Un ejemplo del uso del cálculo relacional es:

Tomando la relación EMPLEADO de la figura 16, trátase de obtener una relación, llamada Q, que contenga un conjunto de valores NOMBRE-EMPLEADO para todos los empleados del departamento No. 721:

Q(EMPLEADO . NOMBRE-EMPLEADO): EMPLEADO . NRO-DEPT=721

Las ventajas del cálculo relacional sobre el álgebra relacional son:

1. El usuario no declara nada acerca de cómo el sistema se las arreglará para obtener el resultado deseado; por lo tanto el sistema está en libertad para optimizar el método.

2. Permite procedimientos de seguridad más discriminativos, pues pueden basarse en una definición de las propiedades de los datos requeridos más bien que un procedimiento declarado para leerlos.

3. El requerir datos por sus propiedades es más natural para el usuario que el requerirlos mediante la especificación de una serie de operaciones.

En cambio, el cálculo relacional tiene la desventaja, frente al álgebra, de ser más difícil de implementar.

La normalización elimina todos los dominios no simples, convirtiendo los datos a la forma de tablas bidimensionales. Es ésta la que llamamos primera formal normal. El proceso de normalización subsiguiente, examina las relaciones de la primera forma normal y puede partir algunas de ellas en relaciones aún más simples. Este proceso consta de dos pasos. En el primero, se reducen los datos a la segunda forma normal, mientras que el último paso nos lleva a la tercera forma normal. Las ideas básicas en las que se apoyan estas nuevas formas normales son simples, pero las ramificaciones de esta normalización son muchas y muy sutiles y varían según el tipo de uso de la base.

Al intentar la definición de las relaciones entre datos, el diseñador debe tratar de descubrir cuáles atributos dependen de cuáles otros, por medio de expresiones tales como "es dependiente de", "pertenece a", "está", "propiedad de", etc. A esto se le llama dependencia funcional.

Consideremos la relación:

EMPLEADO

NRO-EMPLEADO	NOMBRE-EMPLEADO	SALARIO	N-PROYECTO	FECHA-TERMINACION

Las dependencias funcionales en esta relación se ven en la figura 18.

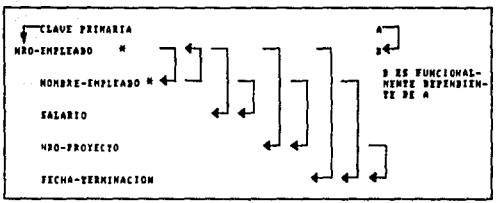
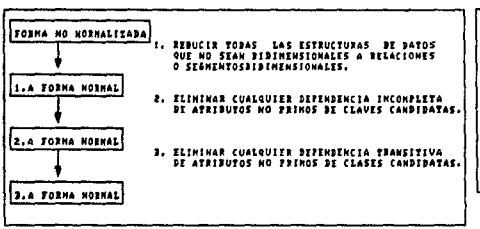
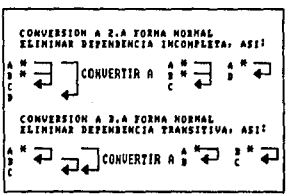


FIGURA 18. DEPENDENCIAS FUNCIONALES EN LA RELACION EMPLEADO. LOS ASTERISCOS DISTINGUEN A LOS ATRIBUTOS PRIMOS (MIEMBROS DE CLAVES CANDIDATAS).

NRO-EMPLEADO no es funcionalmente dependiente de SALARIO, porque varios empleados pueden tener el mismo salario. De igual modo, NRO-EMPLEADO no es funcionalmente dependiente de NRO-PROYECTO, pero sí lo es FECHA-TERMINACION. Ningún otro atributo de esta relación es totalmente dependiente de NRO-PROYECTO.

Una vez analizada la dependencia funcional podemos mencionar los pasos generales en el proceso de normalización que se ven en las siguientes figuras.



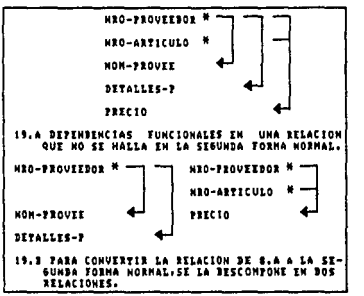


Para poder comprender mejor la segunda forma normal, tomamos la siguiente relación:

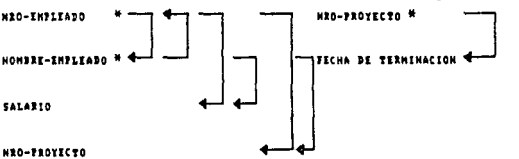
SUMINISTROS

NRO-PROVEEDOR	NRO-ARTICULO	NOM-PROVEE	DETALLES-P	PRECIO

Podemos observar que en la figura 19.a, existe una dependencia funcional en la relación que no se halla en la segunda forma normal y en la 19.b se convierte esta relación a la segunda forma normal, descomponiéndose en dos relaciones.



Para ejemplificar la tercera forma normal tomamos la figura 18, observando que FECHA-TERMINACION es funcionalmente dependiente de NRO-EMPLEADO. Por lo tanto, FECHA-TERMINACION es transitivamente dependiente de NRO-EMPLEADO. Entonces podemos convertir esta relación a tercera forma normal partiéndola de la manera siguiente:



A continuación se hablará de ciertas ventajas de la representación de datos de la forma normalizada:

- **Facilidad de uso.** La manera fácil de representar los datos para el usuario por medio de tablas bidimensionales.

- **Flexibilidad.** Las operaciones permiten separar y unir relaciones de modo de poder dar a los usuarios los archivos que necesitan.

- **Precisión.** Las relaciones (tablas) tienen un significado preciso y pueden ser manipuladas con la matemática del álgebra o el cálculo relacional.

- **Seguridad.** Es más fácil implementar los controles de seguridad.

- **Relacionabilidad.** Se tiene la máxima flexibilidad para relacionar atributos de diferentes conjuntos de registros o archivos.

- **Independencia de datos.** La adecuada independencia de datos es más facil de lograr en las bases relacionales.

- **Lenguaje para la manipulación de datos.** El lenguaje para la manipulación de datos podrá basarse en el álgebra o cálculo relacional.

- **Claridad.** Por medio de representaciones lógicas basadas en tablas y no en el uso de flechas.

Finalmente podemos decir que las bases de datos relacionales tienen 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.

BASE DE DATOS ORIENTADAS A OBJETOS

La programación orientada a objetos ha tomado gran auge en los últimos años por la tendencia a desarrollar aplicaciones complejas en tiempos más cortos reduciendo costos, a través del soporte de código reutilizable que es más fácil de modificar y mejorar.

Las bases de datos orientadas al objeto (BDOO) en esencia usan un modelo navegacional de computación, y no están basadas en teorías matemáticas como las relacionales, definiéndose la navegación como un proceso repetido de pasar de un registro actual a un registro objetivo. En cada punto de embarque es posible recuperar, modificar o almacenar un registro. La manipulación de los registros únicos obtenida en cada punto se logra fácilmente utilizando proposiciones en programación anfitriona. El mapa del que se dispone durante los viajes es el esquema externo asignado al programador.

La mayor diferencia entre BDOO y los modelos jerárquicos y relacionales es que los BDOO introducen una totalidad nueva de conceptos, conceptos que incluyen objetos, clase, herencia, métodos y mensajes.

OBJETOS. En una BDOO cualquier cosa es un objeto y es manipulado como tal. la fila/columna de una base relacional es sustituída por la noción de colecciones de objetos.

En general, una colección de objetos es ella misma un objeto y puede ser manipulada del mismo modo que se manipulan los restantes objetos.

CLASES. Las BDOO sustituyen la noción relación de tipos de datos atómicos con una noción jerárquica de clases y subclases. Por ejemplo, vehículos es una clase de objeto, y los miembros individuales (instancias) de esa clase incluirían un coche, una bicicleta o un tren.

La clase vehículos podría incluir subclases denominadas coches y trenes, representando una forma más especializada de vehículo. Análogamente, la clase coches podría incluir una subclase denominada convertibles, etc.

HERENCIA. Los objetos heredan las características de su clase y de todas las clases de nivel superior a las que pertenecen. Por ejemplo la clase coche tiene como atributos número de puertas, y la clase convertibles heredaría este atributo.

MENSAJES Y METODOS. Los objetos se comunican unos con otros mediante el envío y recepción de mensajes. Cuando recibe un mensaje, un objeto responde ejecutando un método, un programa almacenado dentro del objeto que determina cómo se procesa el mensaje. Esto indica que un objeto incluye un conjunto de comportamientos descritos por sus métodos. Generalmente un objeto comparte muchos de los mismos métodos con otros objetos de su clase.

Las dos mayores aplicaciones comerciales de las BDOO son la "documentación de bases de datos y la distribución de bases de datos".

Los primeros permiten construir ricos modelajes de datos, inferencias, y el almacenaje de objetos complejos tales como el de voz, video, vectores e imagenes gráficas.

En resumen, éstas soportan prolongadas iteraciones que las bases de datos pueden llevar varios días y múltiples versiones de datos. Los documentos de las bases de datos pueden también ser independientes de un lenguaje de programación particular que permita particionar datos a nivel objeto sobre muchas aplicaciones diferentes.

Se dice que las BDOO solapan con fácil semántica los modelos de datos y proveen mucho más desarrollo que el modelo relacional y son involucradas como un resultado de las limitaciones del modelo relacional.

BASES DE DATOS DISTRIBUIDA

En un sistema de base de datos distribuído (BDD) los datos se almacenan en un conjunto de nodos, los cuales pueden ser microcomputadoras, estaciones de trabajo, minicomputadoras y sistemas de cómputo grandes, éstos mantienen un sistema de base de datos local y se comunican entre sí a través de los medios de comunicación anteriormente mencionados.

Este conjunto de nodos puede participar en la ejecución de transacciones locales o globales, las primeras accesan información que reside sólo en ese nodo y las segundas accesan información de varios nodos, además de que requieren comunicación entre los mismos. Los nodos de un sistema distribuído de base de datos pueden estar dispersos de manera física ya sea en una área geográfica extensa o en una reducida. Cada uno está consciente de la existencia de los demás, y permite ejecutar transacciones tanto locales como globales. Algunas razones para crear sistemas de BDD son:

- Compartir y accesar la información de manera confiable y eficiente. Esto es, si varios nodos están conectados entre sí, entonces un usuario de un nodo puede accesar datos disponibles en otro nodo y cada nodo puede controlar los datos almacenados localmente. En un sistema distribuído existe un administrador global de la base de datos que se encarga de todo el sistema. Parte de estas responsabilidades se delega al administrador de base de datos de cada localidad. Dependiendo del diseño del sistema distribuído, cada administrador local podrá tener un grado de autonomía diferente, que se conoce como autonomía local. La posibilidad de contar con autonomía local es en muchos casos una ventaja importante de las bases distribuídas.

- Mejorar la confiabilidad y la disponibilidad. Si se presenta una falla en un nodo distribuído, es posible que los demás nodos sigan trabajando.

  El sistema debe detectar cuando falla un nodo y tomar las medidas necesarias para recuperarse de la falla, con el mínimo de complicaciones.

- Agilizar el proceso de las consultas. Si una consulta comprende datos de varios nodos, puede ser posible dividir la consulta en varias subconsultas que se ejecuten a la vez en diferentes nodos.

Una base de datos distribuída (BDD) implica conservar una distribución de responsabilidades a múltiples bases de datos, cada base en el conjunto tendrá sus conexiones internas y algunas con otros nodos, implicando una comunicación masiva de datos entre bases de datos distintas, y existiendo en cada nodo información acerca de los datos remotos, así como locales. Es decir una parte del esquema en cada nodo contendrá información tanto de él mismo como de los demás nodos. Un esquema local sólo tiene que conservar datos acerca de la ubicación de elementos remotos que puedan ser solicitados para consultas que se originen en el lugar de la parte del esquema.

Tal parte del esquema contendrá anotaciones de definición para campos en otros lugares, pero sólo es necesario conservar en la sección de almacenamiento del esquema los nombres y los sitios de atributos en lugares remotos. Aclarando que aún en este caso los esquemas distribuídos pueden volverse largos y difíciles de mantener, ya que la información del esquema crea réplicas o copias en nodos múltiples. Algunos de los datos pueden tener réplicas en más de un nodo. Una solicitud de recuperación sólo necesita dirigirse al nodo en donde es más fácil obtener esta información, o que tenga la carga más baja. Una actualización tendrá que dirigirse a todas las copias de los datos. Para ejecutar una transacción que incluya múltiples nodos, aquellas porciones de la transacción que no puedan ejecutarse localmente se transformarán en subtransacciones que se transmitirán a través de los enlaces de comunicación, para ser ejecutadas en nodos remotos. Para optimizar las transacciones en las BDD es necesario considerar los siguientes puntos:

Fuentes ¿ Qué sitios tienen los campos solicitados?

Tamaños de los segmentos de datos ¿ Cuál es el resultado parcial esperado del segmento de datos para la subtransacción? Esta pregunta se repite después de cada paso de procesamiento.

Capacidad de recuperación ¿ Cuál es la velocidad y el costo de recuperación en esos nodos?

Capacidad de comunicación ¿ Cuáles son las capacidades de transmisión de datos, es decir, las razones de transferencia de datos disponibles, entre los sitios?

Capacidad de procesamiento ¿ Cuál es la capacidad de los diferentes nodos para realizar cualquier procesamiento solicitado?

Nodo del resultado ¿La salida final se requiere en el nodo de la consulta, en otro nodo o en cualquier parte de la red?.

Finalmente, el objetivo principal de una BDD es procesar datos de varias bases independientes de la naturaleza de sus enlaces de comunicación, y, por lo tanto, de la presencia de una red que, aunque usual, no es esencial en una BDD.

2.3 FLEXIBILIDAD, INDEPENDENCIA E INTEGRIDAD

FLEXIBILIDAD

Es de particular importancia en el diseño de una base de datos que la información (datos) se almacene de manera que se le pueda utilizar indiferentemente para una amplia variedad de aplicaciones y que a la vez pueda cambiarse fácil y rápidamente la manera de usarla.

La implantación de una base de datos, busca que los datos sean aprovechados para tantas aplicaciones como sea posible, permitiendo no sólo la lectura de los datos almacenados, sino la contínua modificación de los que son necesarios, para el control de las operaciones, una fácil restructuración siempre que haya que agregar nuevos tipos de datos o nuevas aplicaciones.

En sí, la flexibilidad busca la forma deseable de almacenar los datos de uso frecuente, de manera que resulte fácil, confiable y rápido acceder a ellos, así como una manera simple y nítida de que sean concebidos, para finalmente dar a los usuarios los archivos que necesitan para sus aplicaciones y de la forma en que los quieren.

Dos aspectos del diseño de la base de datos son importantes con miras a lograr flexibilidad de uso que es esencial en la mayoría de las aplicaciones comerciales. Primero, los datos deben ser independientes de los programas que los utilizan, de modo que se les pueda enriquecer y reestructurar sin que resulte necesario modificar los programas existentes. Segundo, debe ser posible interrogar y explorar la base de datos sin necesidad de recurrir a la tediosa operación de escribir programas utilizando los lenguajes convencionales de programación. En lugar de éstos se utilizarán lenguajes especiales para averiguación.

INDEPENDENCIA

La independencia de los datos es la independencia deseable de las aplicaciones con respecto a los sistemas de hardware y software.

Cuando los primeros sistemas de bases de datos hubieron estado en uso durante cierto tiempo, se patentizó la necesidad de una mayor independencia de datos. Con el transcurso de los años, la cantidad de programas de una organización va creciendo hasta el punto de que en algún momento no se puede pensar en volver a escribir estos programas.

Uno de los objetivos más importantes en el diseño de una base de datos es la de planear de manera que se le pueda modificar sin necesidad de tener que alterar los programas de aplicación en uso.

Para lograr esta esencial protección, el diseño debe prestar atención a dos niveles de independencia de datos: La independencia física y la independencia lógica.

La vista que de los datos tiene el programador de aplicaciones debe ser independiente de la representación física /3 de aquéllos y el software de administración de datos debe ejecutar las conversiones necesarias para pasar de una a otra. Cuando se introduzcan cambios en la organización física de los datos o en el hardware, estos cambios se reflejarán en el software de administración de datos, pero no sobre los programas de aplicación, es decir, se entiende que pueden modificarse la distribución y la organización física de los datos sin afectar ni a la estructura lógica (se refiere a la forma en que los datos se presentan al programador de aplicaciones o a sus usuarios) ni a los programas de aplicación. Es ésta la que llamamos independencia física de los datos.

La vista de los datos que tiene el programador de aplicaciones debe estar protegida contra los cambios de la estructura lógica y contra los requerimientos propios de otros programas de aplicación. La representación lógica global de los datos en sí cambia frecuentemente en muchas instalaciones, a medida que se agregan nuevas aplicaciones o se modifican las viejas. Se agregan así nuevos campos en los registros lógicos y se crean nuevas relaciones entre los campos existentes. Lo que hasta entonces era una jerarquía de dos niveles se ha transformado posiblemente en una jerarquía de tres niveles, así sucesivamente. Es importante que estos cambios en la organización lógica de los datos se introduzcan sin tener que volver a escribir los programas de aplicación que no están directamente relacionados con aquéllos.

Para lograr este tipo de independencia, la vista de los datos que corresponde a cualquier programa de aplicación debe ser desvinculada de la representación lógica global de los datos. Debe ser posible agregar nuevos campos a un registro sin afectar los programas de aplicación existentes y que utilicen ese registro. Esta desvinculación es lo que se llama independencia lógica de los datos.

Toda organización de base de datos requiere por lo tanto tres vistas diferentes de los datos:

/3 FORMA EN COMO LOS DATOS SE REGISTRAN Y ORGANIZAN EN EL HARDWARE.

1. La representación física.
2. La representación lógica global de la base de datos.
3. Las representaciones individuales de los programas de aplicación.

Debe ser posible modificar la primera de éstas sin afectar a las otras. Como lo indica la figura 20, la desvinculación de la organización física respecto de las otras dos vistas, ha de ser tan completa como sea posible. También se debe poder modificar la representación lógica global de los datos del sistema sin reescribir ningún programa de aplicación. Cuando las necesidades de información de un programa de aplicación cambian, el cambio puede exigir la modificación de la descripción lógica global, pero no debe dar lugar a la necesidad de modificar otros programas. Como lo indica la figura 21, la barrera entre los programas de aplicación y la descripción lógica global de los datos tiene que ser lo suficientemente completa.

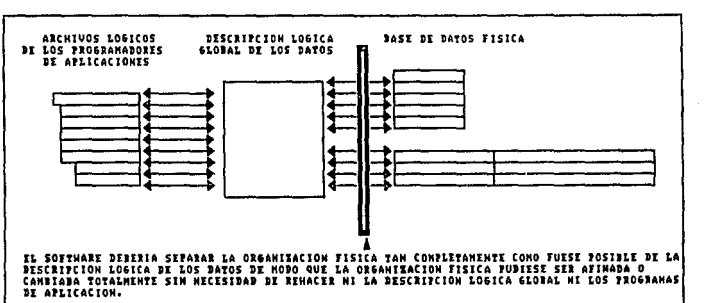


FIGURA 20

En general deberá asegurarse la independencia de datos que sugiere la figura 22. Algunos sistemas actuales proveen cierta independencia física pero poca o ninguna independencia lógica.

Se ha mencionado el término "estructura lógica general de los datos", el bloque que aparece en el centro de la ilustración la representa, y es llamada a menudo "vista lógica global de los datos". Esta vista puede ser enteramente distinta de la que ofrece la estructura física y de la propia de los programas de aplicación.

El software de la base de datos se encargará, en efecto, de convertir la vista que el programador de aplicación tiene de los datos en la vista lógica global, y transformará luego, esta vista lógica global en la representación física.

El propósito de la estructura que vemos en la figura es el de permitir la máxima libertad para cambiar las estructuras de los datos sin tener que rehacer mucho de lo ya hecho en la base de datos. La figura 22 muestra algunos cambios que se introducen frecuentemente en las bases de datos e indica si esos cambios pueden ejecutarse o no sin reestructurar la organización física del almacén, la vista lógica global de los datos o los programas de aplicación, excepto naturalmente el que ha provocado el cambio. Las cruces que aparecen en la figura 22 representan objetivos para el diseño contemporáneo del software de base de datos.

Al juzgar un sistema de administración de bases de datos el usuario no debe preguntarse si ese sistema provee la completa independencia de datos, sino en qué medida son los datos independientes. Dadas las instalaciones del sistema, ¿qué cambios posibles en la organización física exigen la modificación del esquema y qué posibles cambios en el formato de los datos, sus estructuras y sus métodos de localización exigen la modificación de los programas?.

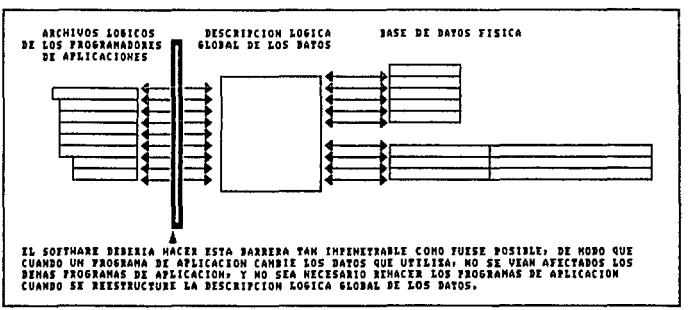


FIGURA 21

Algunos cambios que podrían ser necesarios introducir en los datos son:

1. Cambios en los campos, su nombre, su tamaño, el tipo etc.
2. Cambio en segmentos o registros lógicos.
3. Cambios en la estructura lógica.
4. Cambios en la etructura física.
5. Cambios en el hardware.

	NO HAY CAMBIOS EN LOS PROGRAMAS DE APLICACION.	NO HAY CAMBIO EN LA DESCRIPCION LOGICA DE LOS DATOS.	NO HAY CAMBIO EN LA ORGANIZACION DEL ALMACENAMIENTO FISICO DE LOS DATOS.
SI AGREGA UN NUEVO PROGRAMA DE APLICACION, EL QUE UTILIZA NUEVOS TIPOS DE DATOS.	X	X	X
UN PROGRAMA DE APLICACION EXIGE UNA REPRESENTACION MODIFICADA DE LOS DATOS EXISTENTES	X	X	X
SE AGREGA UN NUO PROGRAMA DE APLICACION QUE UTILIZA LOS TIPOS DE DATOS EXISTENTES	X		
SE INSERTAN NUEVAS OCURRENCIAS DE REGISTRO O SE ELIMINAN LAS VIEJAS	X	X	X
SE MEJORA LA DESCRIPCION LOGICA GLOBAL DE LOS DATOS O SE CREAN NUEVAS RELACIONES ENTRE LOS DATOS	X		
SE CONSOLIDAN DOS BASES DE DATOS	X		
SE MEJORA LA ORGANIZACION FISICA DE LOS DATOS, POSIBLEMENTE SE USAN DIFERENTES REPRESENTACIONES	X	X	
SE MODIFICAN LOS METODOS DE DIRECCIONAMIENO	X	X	
LOS DATOS SE MUDAN A UN TIPO DIFERENTE DE VOLUMEN	X	X	
SE MODIFICA EL SOFTWARE	X	X	
SE MODIFICA EL HARDWARE	X	X	

FIGURA 22. CARACTERISTICAS DE INDEPENDENCIA DE DATOS DESEABLE EN LOS SISTEMAS DE BASES DE DATOS MODERNOS

Ninguno de los sistemas de administración de bases de datos actuales proveen protección contra todos los tipos de cambio en los datos. A medida que mejora el software y el hardware de las bases de datos, aumenta también el grado de independencia posible a un costo razonable.

Algunos de los costos de mantenimiento de los programas en los que se incurre a causa de la insuficiente independencia de los datos, podrían evitarse si se introdujeran ciertas normas para la programación y alguna disciplina para el análisis de sistemas en la empresa.

INTEGRIDAD

La integridad es conservar la seguridad en un sistema en que se permite a múltiples usuarios el acceso a éste y compartir la base de datos. La integridad de una base de datos indica la ausencia de datos inconsistentes.

Cuando una base de datos incluye información utilizada por muchos usuarios, es importante que no puedan destruirse los datos almacenados ni las relaciones que existen entre los distintos campos. Ocasionalmente se producirán fallos de hardware y diversos tipos de accidente. El almacenamiento de los datos y los procedimientos de actualización e inserción deben asegurar que el sistema pueda recuperarse de estas contingencias sin daño para los datos. Toda instalación debe garantizar la integridad de la información que almacena. Además de proteger los datos contra posibles problemas sistémicos deben incluirse también procedimientos de chequeo que aseguren que los valores de los datos se ajusten a ciertas reglas prescritas de antemano. Estas pruebas podrán hacerse verificando las relaciones que existen entre varios valores de datos.

Un error en una sola transacción puede producir una falla si una actualización de la base de datos no se realiza hasta concluirla, pero hay técnicas como la confiabilidad de transacciones y bitácora de actividades que se ocupan de este problema.

La transacción se define como un programa que modifica el estado de la base de datos de manera que, si el estado inicial de dicha base de datos era correcto y la transacción no introduce un error debido a una falla o a datos incorrectos de entrada, el estado de la base de datos vuelve a ser correcto después de que la transacción haya concluído con éxito. Si la transacción no concluye adecuadamente ninguno de sus efectos deberá ser visible en la base de datos. La bitácora de actividades conserva el estado previo de la base de datos, posibilita la eliminación de una transacción específica, también considera la corrección de salida errónea para las transacciones de consulta, proporciona un número de secuencia, la fecha y el momento de llegada de una transacción, entre otros. 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 inconsistencías temporales provocadas por una actualización concurrente, y diseminar esta inconsistencia a través de la base de datos o en las salidas que se envían a los usuarios.

Sería deseable estar seguro de que los mecanismos empleados para proteger datos compartidos no puedan fallar. Estos mecanismos son los seguros, la hibernación y el punto muerto. Cuando se habla de seguros posiblemente se trate de algún medio para limitar y retrasar el acceso, comúnmente implantado por el sistema a través de uno o varios semáforos. Los semáforos simplemente consisten en indicar si un recurso asociado está desocupado u ocupado. La hibernación ocurre cuando una transacción no recibe recursos durante un período excesivamente largo. Los puntos muertos ocurren cuando dos o más transacciones solicitan recursos en forma incremental y se bloquean mutuamente impidiéndose una a otra la conclusión. En los sistemas prácticos de hoy en día es imposible probar que la protección de la integridad sea correcta. Es necesario que muchos niveles de hardware y software funcionen sin fallas para permitir que los mecanismos de protección de la integridad realicen sus funciones.

Otras consideraciones que ayudan a diseñar y operar una base de datos en forma tal que la integridad se conserve en un alto nivel son: La integridad de programación y la vigilancia de integridad.

- Integridad de programación, La mayor parte de los sistemas permiten a los programadores tomar decisiones referentes al uso de seguros. Pueden presentarse problemas debido a que se permita a un individuo tomar decisiones que afectan a programas escritos por otro.

  Si el programador no está restringido por el sistema de acceso de provocar una interferencia potencial en las actividades de otros, puede resultar necesario un mayor nivel de control. las operaciones de actualización pueden permitirse sólo si una reclamación ha precedido a la operación. Esto elimina la acción del programador, de decidir asegurar o no, aun cuando no garantice que los seguros cubran la región adecuada.

- Vigilancia de la integridad, ya que una sola falla de consistencia puede copiarse gradualmente a través de toda la base de datos, una estrategia de vigilancia regular de la base de datos es esencial siempre que se conserven datos a largo plazo. Si los resultados basados en datos almacenados durante muchos años son erróneos, la confianza del usuario en todo el trabajo realizado en años anteriores se pierde repentinamente.

  La vigilancia es posible en dos niveles: estructural y orientado al contenido. La vigilancia estructural puede realizarla el sistema de archivo sin participación del usuario. La vigilancia orientada al contenido requiere que el usuario proporcione afirmaciones referentes a los vínculos entre datos.

En cualquier caso la vigilancia sólo es posible si existe cierta redundancia en una base de datos.

Pero éstas razones traen como consecuencia o desventaja un costo mayor de desarrollo del software, una mayor posibilidad de errores y aumento en el costo extra del procesamiento ocasionado por el intercambio de mensajes y los cálculos adicionales que se requieren para coordinar los nodos.

2.4 SISTEMAS MANEJADORES DE BASE DE DATOS

Los sistemas manejadores de base de datos (SMBD) han desempeñado un papel central en la evolución de las PC's, al proporcionar a los usuarios finales poder y control para solucionar sus problemas en el manejo y aplicación de información, es por eso que estos sistemas nacen como solución para el control al acceso a las bases de datos, siendo un conjunto integrado de programas que crean, generan, ejecutan, controlan y mantienen las bases de datos, y como tal este conjunto integrado de programas debe incluir todo el software necesario para esa finalidad, el SMBD sabe acerca de los datos, y cómo están organizados en el disco. Así pues empresas dedicadas al ambiente computacional, viendo el amplio desarrollo que ofrece un SMBD, han invertido dinero y esfuerzo en el estudio y desarrollo de estos sistemas y se han preocupado por difundirlos al mercado mundial de computación.

La libertad de elegir un SMBD está condicionada frecuentemente por el hardware disponible, -esto es especialmente cierto para los SMBD suministrados por los fabricantes, puesto que los SMBD vendidos por las casa de software son generalmente más flexibles y pueden implementarse en una gama de computadoras-. Evidentemente son preferibles los SMBD independientes del computador, pues permiten el cambio de éste, cuando sea necesario, sin tener que cambiar también el SMBD.

2.4.1 OBJETIVOS

- **Organizar los datos con sencillez.** La estructura de los datos en un SMBD debe ser tan simple como sea posible. Cuanto más sencilla sea la estructura de la base de datos, tanto más fácil será manipular los datos.

- **Proporcionar respuestas a las consultas a su debido tiempo.** Un SMBD debe actuar lo suficientemente rápido.

- **Reducir costos.** El ahorro radica en el tiempo de programación. Con un SMBD, la solución de problemas simples tarda apenas unas horas en vez de días. Algunas aplicaciones complejas requieren un solo programador en vez de un equipo.

- **Salvaguardar la integridad de los datos.** Una base de datos suele estar constituída por varios archivos, registros, campos e interconexiones. Si algo va mal, una parte de la base de datos puede ser ilegible. El SMBD deberá autocomprobarse e inmediatamente comunicarle si encuentra alguna anomalía. Deberá siempre estar vigilando los datos incompatibles, no permitiendo su entrada, y si se deslizaran alertando al usuario de su presencia.

- **Permitir acceso a los usuarios.** En calidad de usuario de SMBD debe estar en condiciones de plantear una amplia gama de preguntas acerca de los datos. La mayoría de los SMBD tienen un lenguaje de consulta para preguntar estas cuestiones. Este lenguaje debe ser fácil de usar y sencillo de aprender. Si se consulta la base de datos, suele conseguir una pantalla llena de datos. A veces necesita un informe con una salida impresa de formateado atractivo. Un SMBD debe permitir a un usuario de computadora inexperto crear estos informes mediante un generador de informes.

- **Proporcionar niveles de seguridad.** Los buenos SMBD incluyen medios que restringen la posibilidad de que personas no autorizadas cambien o vean los datos. Esto suele hacerse con palabras de paso contraseñas, cada una de las cuales tiene su propio nivel de seguridad.

- **Proporcionar un funcionamiento con protección contra fallos.**

- **Cambios menos arduos.** Sin un SMBD, incluso el cambio más trivial puede requerir una nueva escritura de decenas de programas. Una vez que se haga la nueva escritura aparecen nuevos defectos que, a veces, causan pérdidas de tiempos y de datos. Los SMBD pueden hacer el cambio menos arduo. A menudo podrá redisponer o añadir campos sin tocar ningún programa, excepto aquellos que utilizan nuevos campos.

- **Permitir que los datos sean compartidos.** El SMBD debe permitir que los datos sean compartidos. En una organización con una computadora de una sola terminal, un SMBD debe permitir a personas diferentes usar los mismos datos.

- **Ofrecer flexibilidad de lenguaje.** Un lenguaje de nivel más bajo puede ser un lenguaje establecido como COBOL o BASIC, con conexiones dentro del SMBD, o puede ser un lenguaje especial incluído en el paquete del SMBD.

  Los lenguajes del SMBD introducen sus propias órdenes y estructuras adecuadas específicamente a la base de datos.

- **Producir su propia documentación.** Con demasiada frecuencia, todos los programadores escriben sistemas que nadie más puede entender. Pueden estar motivados por la seguridad de su trabajo o por su ignorancia pero el resultado es siempre inaceptable.
  Si el software produce automáticamente documentación, lleva ventaja en este juego.

En todos los SMBD, el diccionario de datos proporciona un buen comienzo hacia la documentación producida por la computadora, teniendo un libro con copias impresas de estos diccionarios conocerá la estructura de su base de datos en todo momento.

Casi todos los SMBD son controlados por menú, siendo éste realmente más fácil para un usuario inexperto, también ofrecen la posibilidad de conservar una secuencia de órdenes o elecciones de menús en forma de un archivo de disco, que se puede editar y solicitar por su nombre cuando se le quiera ejecutar. Este archivo de disco se podría considerar un programa, pero no en el mismo sentido que un programa en BASIC o en COBOL. Un programa real es una secuencia de instrucciones que permiten operar con los datos registro por registro. Con la adición de instrucciones de control, un programa puede probar condiciones y luego alterar su flujo lógico. Cualquier SMBD que utilice este tipo de programa se denomina de procedimiento. Cualquier SMBD que ofrece elecciones de menú, rellenado de espacios o realiza órdenes globales, se describe como de no procedimiento. Los paquetes que permiten ambos tipos de control son los más flexibles. Hay que tomar en cuenta diferentes criterios para comparar un SMBD con otro, éstos pueden ser tiempo de acceso, velocidad de procesamiento, y capacidad de almacenamiento de datos.

Hay que aclarar que los SMBD ya no están en su mayoría escritos en un lenguaje de alto nivel como BASIC, sino que por el contrario, los paquetes están regularmente escritos en un lenguaje ensamblador o en lenguaje C, ocasionando esto una alta rapidez en procesamiento.

2.4.2 CARACTERISTICAS

- **Diccionario de datos.** El diccionario de datos es el corazón de la base de datos. Le indica casi todo acerca de los datos.

- **Medios de consulta.** Debe haber una forma, para personas que no sean programadores, de ver y actualizar los datos, cuanto más sean potentes estos medios, tanto mejor, pero la potencia no deberá comprometer la facilidad de su empleo.

- **Generador de informes.** Hay una considerable variación de potencia en el generador de informes de cada SMBD, pero hay una solución alternativa entre la potencia y la facilidad de utilización. Si el SMBD tiene un lenguaje, establecido o incorporado, se puede utilizar para programar cualquier informe que el generador de informes no creará, a un costo bastante más bajo que contratáramos a un programador para realizar este reporte.

- **Compatibilidad de archivos con otros programas.** El SMBD debe ser capaz de leer datos de los programas en BASIC y COBOL cuando se encuentran en los sistemas de contabilidad existentes y, además, debe ser capaz de escribir datos para utilizarlos con estos programas.

- **Capacidad de reestructuración.** Fácil cambio de programas y datos sin alterar la estructura.

- **Manipulación efectiva de errores.** Cuando cometa un error, su SMBD no destruye todos sus datos. Por supuesto habrá hecho copias de reserva frecuentemente pero resulta molesto tener que usarlas todo el tiempo. Un buen SMBD proporciona mensajes de error útiles y claros y permite corregir el problema. Es difícil la recuperación a partir de problemas causados por el hardware, pero sí debe admitirse directorios y discos completos, impresoras fuera de línea, etc. Si el disco está lleno, el SMBD debe avisarle y permitirle el borrado de los archivos innecesarios.

- **Buena documentación.** Un manual ideal debe ofrecer unos conocimientos para poder recorrer el sistema paso a paso. El manual debe tener también una sección de referencia organizada por orden o función.

- **Archivos múltiples.** El SMBDR el cual es por definición, un sistema de archivos múltiples que permite utilizar varias bases de datos a la vez. La industria de la microcomputadora ha adaptado el concepto relacional a sus máquinas con un éxito considerable, siendo éste un resultado satisfactorio en los SMBDR y siendo más fácil de aprender que otras bases de datos, el SMBDR permite construir sistemas complejos por pasos tomando la estructura de una base de datos relacional.

- **Edición de pantalla completa.** Esta edición le muestra una pantalla con una serie de espacios en blanco, con etiquetas descriptivas correspondientes a los nombres de los campos. Es posible moverse libremente entre los campos utilizando teclas especiales. Después de introducir una pantalla llena de datos se utiliza otra tecla para liberar el registro y comenzar uno nuevo. Un programa de formato de visualización controla el posicionamiento y valida los datos introducidos.

- **Generación de formatos de presentación visual en pantalla.** El formato de visualización es lo que se ve cuando se comienza a introducir o cambiar datos para un registro base de datos dado. Está constituído por las etiquetas descriptivas y por los espacios en blanco para los campos, todos dispuestos de una manera agradable.

- **Capacidad de multiusuario.** Cada usuario será capaz de tener su programa para llamar al SMBD en una división individual y el SMBD, en sí mismo, reside en su propia división, accesible por todos los del usuario.

En la actualidad existen cierto número de SMBD disponibles. Las facilidades ofrecidas por ellos varían muchísimo según su nivel de sofisticación. Las ventajas de un SMBD respecto de un sistema convencional son :

- **Independencia de datos y programas.** Esa es una primera ventaja de una base de datos. Tanto la base de datos como el programa de usuario pueden alterarse independientemente uno de otro. Esto ahorra tiempo y dinero, que de otro modo sería necesario para modificarlo, precisamente para mantener su consistencia.

- **Fácil diseño de sistemas.** A un diseñador de sistemas en un entorno de una base de datos no le conciernen extensos diseños de archivos, ni la duplicación, ni su inconsistencia, etc., de un sistema convencional. En una base de datos, los datos existen en una forma apropiada para todas las aplicaciones, el diseñador sólo tiene que escoger la que necesita.

- **Fácil de programar.** La tarea de programación es mucho más reducida, porque los programadores están relevados de los detalles de procesado de archivos, actualización de archivos, y de numerosos procesos de ordenación.

- **Múltiples lenguajes anfitrión.** Se espera que un SMBD soporte cierto número de lenguajes anfitrión, de modo que un usuario pueda elegir aquél que es más conveniente para una aplicación particular.

- **Datos consistentes y actualizados.** Una base de datos reduce la duplicación de datos y ayuda a mantener los datos consistentes y actualizados. En un sistema convencional de datos, la duplicación de datos en varios archivos, conduce frecuentemente a la inconsistencia.

- **Utilización concurrente.** Una base de datos puede permitir que acudan a ella más de un programa al mismo tiempo, maximizando, por tanto, la utilización de recursos.

- **Protección de los datos.** La privacidad e integridad de los datos pueden controlarse mejor en una base de datos.

3. SISTEMAS DE BASE DE DATOS COMPARTIDAS (LAN).

En los 70's se llevó a cabo un intento conjunto de la industria del proceso de datos para realizar lo que hoy se conoce como Sistema Manejador de Base de Datos.

Un sistema manejador de base de datos posibilita el que diferentes usuarios tengan acceso a la misma información, pero organizada de una forma diferente que depende de las necesidades de cada usuario. Por ejemplo, la figura 1 muestra tres departamentos diferentes de una firma que pueden utilizar la misma base de datos para tres aplicaciones diferentes.

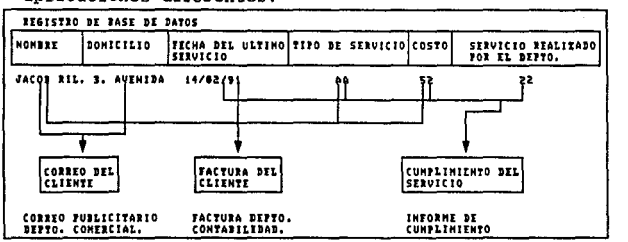


FIGURA 1. SMBD.

Cuando se instala una base de datos en una oficina central y se le mantiene mediante un grupo único dentro de la empresa, se obtiene una base de datos compartida; tener los datos en una oficina central resuelve muchas dificultades que implican la duplicación e integridad de los datos; no obstante, la compartición actual de los datos entre varios usuarios origina también su propio conjunto de problemas técnicos:

1.- Los grupos diferentes dentro de una empresa consideran los mismos datos de modos diferentes; en consecuencia, es preciso manejar estructuras diferentes de datos como norma general.

2.- Para garantizar la consistencia de la base de datos es preciso establecer reglas que deben seguir todos los usuarios.

3.- Hay que prever la posibilidad de permitir el acceso normal y la actualización de los registros individuales sin que los usuarios se destruyan los datos entre sí por accidente.

En temas anteriores se definieron las bases de datos distribuídas (BDD), como una solución para compartir y accesar datos, los cuales pueden ser almacenados en diferentes lugares llamados nodos que se encuentran conectados a la red, y los cuales permiten ejecutar transacciones locales y globales, tienen una autonomía de operación y saben que existen otros nodos, con los cuales pueden establecer comunicación.

Dentro de los sistemas manejadores de bases de datos distribuídas (SMBDD), existen los definidos cerrados y los abiertos. En los primeros, la BDD se construye desde el principio, siendo diseñada específicamente cada base de datos en el nodo para seguir los requisitos de la BDD dentro de las restricciones del SMBDD. Por lo tanto, se controlan las incompatibilidades o variaciones entre los nodos. Al contrario un SMBD abierto permite cualquier modelo de datos en los nodos, incluyendo bases de datos preexistentes. Por lo tanto, en teoría, puede existir cualquier número de incompatibilidades o variaciones entre nodos en una BDD abierta.

En cualquier caso, podemos reconocer dos tipos de usuarios, el usuario global, que procesa los datos de la BDD mediante el control de un SMBD, y el usuario local o nodal, que procesa los datos de la BDD mediante el control de un SMBD teóricamente olvidado de la existencia de la BDD
,entonces podemos decir que una BDD tiene dos niveles de control, el global y el nodal. Si un nodo por si mismo es una BDD, lo que es posible, entonces tendríamos muchos niveles más, pero, dentro del contexto de una BDD dada, tendríamos todavía sólo dos niveles global y nodal, independientemente de que si uno de sus nodos es también una BDD; el control global puede ser básicamente cualquiera de los siguientes:

Centralizado: Todo proceso global se controla por un nodo central, por el cual deben canalizarse todas las transacciones globales. El descentralizado: Cada nodo guarda una copia del SMBDD supervisando cada uno las transacciones enviadas por él. (ver figura 2)

El sistema es más estable que el centralizado, puesto que no hay ningún centro cuya caída puede incapacitar a la BDD total. Sin embargo la aplicación de los controles y la preservación de la consistencia es más difícil que la centralizada.

La autonomía de que goza un nodo en una base distribuída puede variar significativamente. En todas las BDD un SMBD en el nodo controla las actividades de cada nodo, pero los nodos no son necesariamente completamente autónomos.

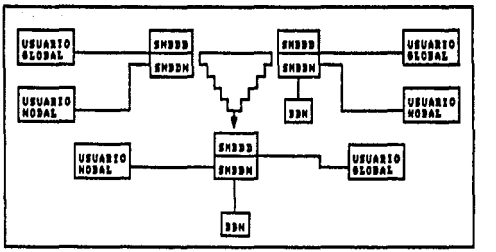


FIGURA 8: BASE DE DATOS DISTRIBUIDA CON CONTROL DESCENTRALIZADO.

Por ejemplo, en algunas BDD sólo se permiten transacciones globales (esto es, las transacciones de datos puramente nodales se tratan también como transacciones globales), mientras que en otras cada nodo es completamente autónomo, teniendo exclusivamente usuarios locales, los cuales no necesitan conocer la existencia de la BDD.

En algunas arquitecturas todos los datos pertenecen a la BDD, mientras que en otras los datos pertenecen a los nodos respectivos, que pueden decidir contribuir sólo con algunos de sus datos a la BDD. En el último caso el nodo puede retener el derecho a quitar sus datos, o a estipular quién puede usarlos y para qué finalidad. Teóricamente, debería ser posible también que un nodo se uniese a varias BDD.

Por autonomía nodal se dice que una base de datos nodal debería:

1) Poder tener usuarios nodales exclusivos que fueran independientes del SMBDD.

2) Ser capaz de unirse a una BDD como nodo y contribuir sólo con un subconjunto lógico de sus propios datos; el subconjunto puede contener desde ninguno hasta todos sus datos.

3) Retener el control total de sí mismo, con medios para eliminarse él mismo o sus datos de la BDD; y también ser capaz de especificar y reforzar el control de autorizaciones tanto para el acceso como para la actualización de cualquiera de sus datos.

4) Ser capaz de controlar, como nodo interno (a través de restricciones de privacidad apropiadas), aquellos de sus datos que pudieran repetirse y aquellos otros nodos, como nodos externos que estuvieran autorizados para recibir tales repeticiones.

5) Ser capaz de unir varias BDD, dando un subconjunto lógico de sus datos para cada una de las BDD. Una base de datos nodal puede ser también por sí misma una BDD, permitiendo así una jerarquía.

Hay que aclarar que los SMBDD que existen en la actualidad, ofrecen pocas capacidades, pero se habla extensamente de las características ideales que un SMBBD debe ofrecer, las cuales se presentan a continuación:

- Ubicación transparente.
- Soporte de diferentes sistemas.(PC's, Servidores, Minis,etc.)
- Trabajar de la misma manera en diferentes redes.
- Consultas, actualizaciones y transacciones distribuídas.
- Seguridad para la base de datos distribuída.
- Acceso uniforme y universal a todos los datos de la organización.

Ningún SMBDD cumple con todas estas características, sin embargo vendedores de SMBDD se han comprometido en luchar por lograrlas.

3.1 TECNICAS PARA COMPARTIR BASE DE DE DATOS EN UNA RED LOCAL.

Se ha mencionado que los SMBDD son una alternativa, para compartir los datos en redes, este punto analizará las técnicas como concurrencia, consistencia y seguridad para compartir datos en redes locales, partiendo de algunos conceptos importantes en el mundo de las LAN's, como el File Server o Servidor de Archivos y el de Cliente/Servidor entre otros.

En el capítulo 1 se menciona el concepto File Server o servidor de archivos el cual surge como un intento para resolver el problema de compartir y controlar los datos en una red local.

Este servidor por lo general es una microcomputadora con un microprocesador rápido, memoria adicional a fin de acelerar las unidades de almacenamiento y un disco duro de gran capacidad para almacenar los datos, ahí se carga el sistema operativo de la red y se conectan los dispositivos como impresoras, modems y graficadores entre otros, puede trabajar como servidor dedicado y no dedicado. El dedicado exclusivamente administra los recursos de la red, y el no dedicado, funciona también como una estación de trabajo.

En un SMBD para red tradicional, el servidor de archivos envía una copia del software del SMBD a la estación de trabajo y si por alguna razón la estación de trabajo hace una petición de datos, aunque el resultado de la petición sea un sólo registro, se envía todo el archivo o archivos que estén inmersos en la petición, esto ocasiona que si existen archivos grandes y hay varios usuarios con infinidad de peticiones, existirá un tráfico en la red que degradará el tiempo y la respuesta.

Un subconjunto de la categoría de servidores de archivos es el Database Server o servidor de base de datos, que es generalmente una microcomputadora con un procesador rápido corriendo un software especializado que permite dedicarla exclusivamente a las labores de búsqueda o archivo de datos, puede ser que en el CPU se dividan las funciones tanto del servidor de archivos como el de datos, también puede existir una tarjeta procesadora separada que está instalada en el servidor de archivos, y los requerimientos de bases de datos son direccionados a un coprocesador que puede estar en la estación de trabajo o en el propio servidor de archivos. Este servidor de base de datos está basado en el estándar de búsqueda de datos denominado SQL. El Structured Query Language (SQL) es un lenguaje universal, para los sistemas manejadores de base de datos relacionales (SMBDR).

El SQL no solamente simplifica los procedimientos de recuperación de datos, sino que también proporciona un lenguaje estándar de base de datos. Además, hace que la información sea intercambiable entre los archivos de base de datos bajo diferentes plataformas.

Cuando el servidor de base datos trabaja en ambiente de red, esta tecnología permite descargar al servidor de archivos o computadora central hasta 10 veces más rápida la operación del software y es una buena solución para aliviar el tráfico de datos dentro de la red, y sólo los registros que cumplen con la petición serán transferidos del servidor de datos a la estación de trabajo y no todo el archivo como sucede con el servidor de archivos.

En una red, una estación de trabajo puede accesar múltiples servidores de base de datos, simulando por la separación física una BDD, donde un usuario puede consultar datos sin saber en que nodo se encuentre éste, se pueden reemplazar de forma sencilla servidores de base de datos que estén en alguna plataforma por otra y permitan la conectividad de diferentes tipos de redes. Los servidores de base de datos reciben simples pedidos de reportes de las estaciones de trabajo y ejecutan el complejo código necesario para extraer y compilar los reportes de la BD. Esto reduce la carga en la red y coloca la fuerte actividad de procesamiento de E/S cerca de los dispositivos mecánicos de almacenamiento que contienen toda la información.

Cuando se menciona al servidor de base de datos se habla del concepto Cliente-Servidor (Front-end/Back-end) o proceso cooperativo. Esto significa que aquí las funciones del SMBD están en dos partes. Los frontales(front-end) de base de datos, tales como herramientas de consulta interactiva, escritores de información y programas de aplicación, se ejecutan en la PC. La máquina de soporte(back-end) de la BD que almacena y administra los datos se ejecuta en el servidor. Este servidor de datos integra las solicitudes desde todas las estaciones de trabajo, como reducción de números, transacciones de datos, acceso de archivos e implementaciones de datos y los resultados computados son enviados de regreso a las estaciones de trabajo.

Las aplicaciones intensas de base de datos son un claro ejemplo de la necesidad y el beneficio del proceso cooperativo. Cuando utilizamos un SMBD que reside en el disco duro del servidor, por ejemplo DBASE III, y varios usuarios están trabajando con la base de datos, el tráfico en el canal de comunicaciones entre el servidor de archivos y las estaciones de trabajo se vuelve muy intenso, ya que cada usuario requiere que la administración de datos entre el servidor de archivos y su estación de trabajo viaje por el canal de comunicación. (ver figura 3).

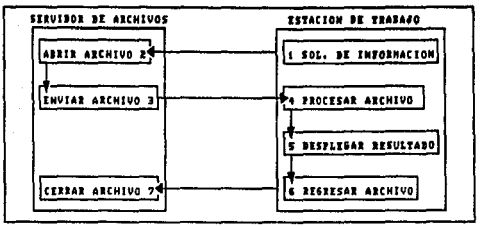


FIGURA 3. AMBIENTE TRADICIONAL DE SERVIDORES DE ARCHIVOS Y ESTACION DE TRABAJO

A diferencia de lo anterior, con un servidor de base de datos, una parte del proceso (la administración de datos) se lleva a cabo en el servidor de base de datos, y otra parte en la estación de trabajo. (ver figura 4)

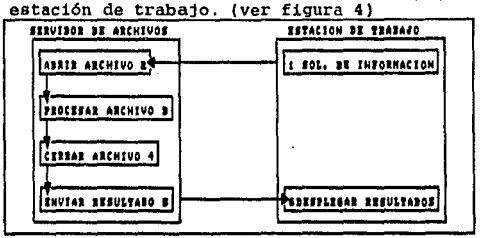


FIGURA 4. AMBIENTE DE PROCESO COOPERATIVO CLIENTE-SERVIDOR.

Como se puede observar en la figura 4, el canal de comunicaciones se utiliza una sóla vez, incrementando con esto la velocidad de respuesta del sistema en forma considerable y evitando la degradación del mismo.

Finalmente varios SMBD basados en SQL proporcionan una biblioteca de llamadas a funciones, las cuales componen una interfaz de programación de aplicaciones (API application programming interface), es decir son para transferir sentencias SQL /1 al SMBD. Un programa de aplicación llama a funciones API, y llama a otras funciones para recuperar resultados de consultas e información del estado procedente del SMBD.

La operación básica de un API SMBD es:

- El programa comienza el acceso a la base de datos con una llamada API que conecta el programa al SMBD y con frecuencia a una base de datos específica.

/1 UNA SENTENCIA SQL DEMANDA UNA ACCION ESPECIFICA POR PARTE DEL SMBD. SQL CONSTA DE 30 SENTENCIAS.

- Para enviar una sentencia SQL al SMBD, el programa construye la sentencia como una cadena de texto en un buffer /2 y luego hace una llamada API para transferir el contenido del buffer al SMBD.

- El programa hace llamadas API para comprobar el estado de su petición al SMBD y para manejar errores.

- Si la petición es una consulta, el programa utiliza llamadas API para recuperar los resultados en los buffers del programa. Típicamente, las llamadas devuelven datos fila a fila o columna a columna.

- El programa termina su acceso a la base de datos con una llamada API que le desconecta del SMBD.

Un API de SQL se utiliza con frecuencia cuando el programa de aplicación y la base de datos están sobre dos sistemas diferentes en una arquitectura cliente/servidor, como se muestra en la figura 5. En esta configuración el código para las funciones API está localizado en el sistema cliente, donde se ejecuta el programa de aplicación. El software SMBD está localizado en el sistema servidor, donde reside la base de datos. Las llamadas desde el programa de aplicación al API tiene lugar localmente dentro del sistema cliente, pero la comunicación entre el API y el SMBD tiene lugar sobre una red, minimizando la cantidad de tráfico de red entre el API y el SMBD.

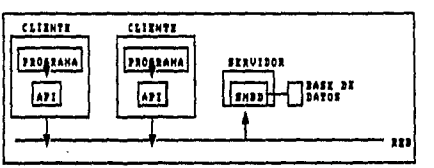


FIGURA 5. UN API SQL EN UNA ARQUITECTURA CLIENTE-SERVIDOR.

/2 BUFFER : AREA EN LA MEMORIA DE LA COMPUTADORA PARA ALMACENAR DATOS

3.1.1 CONCURRENCIA Y ACCESO A LOS DATOS.

La concurrencia nace cuando varias transacciones quieren accesar a los datos a la vez, ocasionando con ésto inconsistencia y pérdida de integridad de los mismos; así es que los SMBD en red, deben proveer facilidades para asegurar que no exista esta pérdida, así como soportar métodos para accesar datos distribuídos físicamente.

Para el control de la concurrencia, algunos SMBD permiten abrir archivos en modos compartidos o exclusivos . En el modo compartido, todos los usuarios en la red pueden leer o escribir archivos, asumiendo que cualquiera está capacitado para escribir a no ser que esté restringido por el uso de una instrucción que lo protege o por el sistema operativo de la red. En el modo exclusivo sólo puede trabajar un solo usuario con el archivo; los demás usuarios deben esperar hasta que el archivo haya sido relevado del uso exclusivo por dicho usuario. Es decir si un usuario intenta abrir un archivo mientras que éste haya sido abierto en modo exclusivo, entonces el SMBD no permitirá el acceso y desplegará un mensaje diciendo que el archivo es usado por otro.

Hay ocasiones en que las operaciones como remplazar, indexar o sortear, requieren que un archivo esté bloqueado por el modo compartido hasta que la operación haya sido completada. Sin embargo, sería egoista un uso de los recursos, si se bloquea la base de datos entera para examinar o actualizar un sólo registro. Cuando se editan los registros individuales en vez del archivo entero. La protección que realizan algunos SMBD con los usuarios de red consiste en intentar bloquear el archivo siempre que la operación que se requiera necesite bloquear un archivo, es decir no se puede bloquear el archivo si alguien ya lo ha bloqueado. Hay algunas órdenes que incluyen operaciones como agregar, copiar y borrar, entre otras. Si el archivo ha sido bloqueado automáticamente por el SMBD como resultado de una de estas órdenes, el archivo será desbloqueado automáticamente cuando el SMBD complete la operación.

Los candados son técnicas que también permiten un control de la concurrencia, consistiendo en marcar los datos ocupados al momento que se accesan. Los conceptos básicos de candados son sencillos, por ejemplo la figura 6 muestra un candado con dos transacciones concurrentes en un SMBD basado en SQL; cuando la transacción A accede a la base de datos , el SMBD bloquea automáticamente cada parte de la base de datos que la transacción consulta o modifica. La transacción B procede en paralelo, y el SMBD también bloquea las partes de la base de datos a las que ella accede.

Si la transacción B trata de acceder a la parte de la base de datos que ha sido bloqueada por la transacción A, el SMBD congela la transacción B, haciéndole que espere a que los datos sean desbloqueados. El SMBD libera los bloqueos mantenidos por la Transacción A solamente cuando ésta finaliza en una operación COMMIT /3 o ROLLBACK /4. El SMBD <<descongela>> la transacción B, permitiéndole que prosiga. La transacción B puede bloquear esa parte de la base de datos por su propia cuenta, protegiéndola de los efectos de otras transacciones.

Como muestra la figura 6, la técnica del candado proporciona a una transacción acceso temporal exclusivo a una parte de una base, impidiendo que otras transacciones modifiquen los datos encerrados (bloqueados).

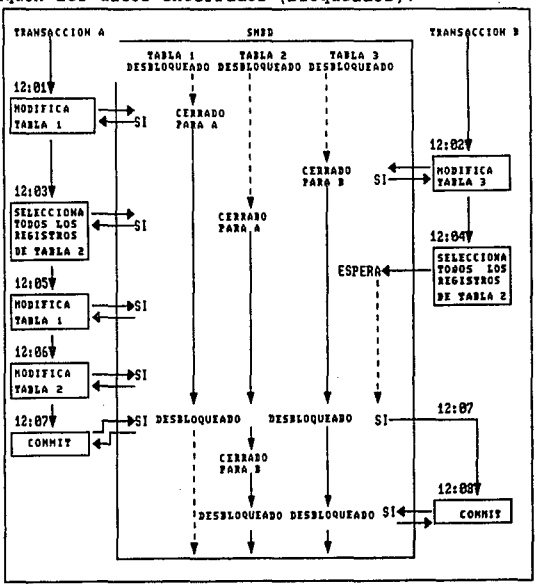


FIGURA 6. CANDADO CON DOS TRANSACCIONES CONCURRENTES.

/3 SEÑALA EL FINAL CON EXITO DE UNA TRANSACCION.
/4 SEÑALA EL FINAL SIN EXITO DE UNA TRANSACCION.

En términos de redes existen candados compartidos y candados exclusivos. Los candados compartidos se utilizan en el SMBD cuando una transacción desea leer datos de la base de datos. Otra transacción concurrente puede también adquirir un candado compartido sobre los mismos datos, permitiéndole que la otra transacción también lea los datos, es decir, al mismo tiempo un usuario puede poner un candado y accesar un registro o bloque de registros. Cuando se pone este tipo de candados sobre los registros sólo se puede leer. Los candados exclusivos se utilizan en el SMBD cuando una transacción desea actualizar datos de la base de datos. Cuando una transacción tiene un candado exclusivo sobre algunos datos, el resto de las transacciones no pueden adquirir ningún tipo de candado (compartido o exclusivo) sobre los datos.

La figura 7 muestra las reglas de los candados y sus combinaciones permitidas de candados que pueden mantener dos transacciones concurrentes.

		TRANSACCION B		
		DESPEJADO	CANDADO COMPARTIDO	CANDADO EXCLUSIVO
TRANSACCION A	DESPEJADO	SI	SI	SI
	CANDADO COMPARTIDO	SI	SI	NO
	CANDADO EXCLUSIVO	SI	NO	NO

FIGURA 7. REGLAS PARA CANDADOS COMPARTIDOS Y EXCLUSIVOS.

Si una transacción trata de adquirir un candado no permitido por las reglas anteriores, se congela hasta que otras transacciones desbloquen los datos que ella requiere.

Finalmente el objetivo de los candados sigue siendo el impedir la interferencia no deseada entre transacciones mientras se sigue proporcionando el mayor acceso concurrente posible a la base de datos.

¿Que pasa si un usuario tiene un candado sobre un registro 1 y otro usuario tiene otro candado sobre un registro 2, y cada uno necesita poner un candado al registro del otro?. El registro 1 está bloqueado por el registro 2 y éste a su vez está bloqueado por el primero, ya que ambos tienen un candado, implica ésto un bloqueo mutuo, al cual se le conoce como punto muerto. Para tratar los puntos muertos, algunos SMBD basados en SQL incluyen típicamente lógica que periódicamente (digamos, una vez por minuto) comprueba los candados mantenidos por varias transacciones;

Cuando detecta un punto muerto, el SMBD elige arbitrariamente una de las transacciones como <<perdedora>> y le da marcha atrás. Esto libera los candados mantenidos por la transacción perdedora, permitiendo al <<ganador>> del punto muerto proseguir. El programa perdedor recibe un código de error informativo que le dice que ha perdido un punto muerto y que su transacción actual ha sido vuelta atrás.

Este esquema para romper puntos muertos significa que cualquier sentencia SQL puede devolver potencialmente un código de error <<perdedor de puntos muertos>>, incluso si no hay nada incorrecto en la sentencia. La transacción que intenta la sentencia es vuelta atrás no debido a ningún fallo de ella misma, sino debido a la actividad concurrente de la base de datos.

La figura 8 muestra una situación de punto muerto. El programa A actualiza la tabla 1, bloqueando por tanto parte de ella. Mientras tanto, el programa B actualiza la tabla 2 bloqueando parte de ella. Ahora el programa A trata de actualizar la tabla 2 y el programa B trata de actualizar la tabla 1, en cada caso tratando de acceder a la parte de la tabla que ha sido previamente cerrada (bloqueada) por el otro programa. Sin intervención externa, cada programa esperará eternamente a que el otro programa comisione su transacción y desbloquée los datos.

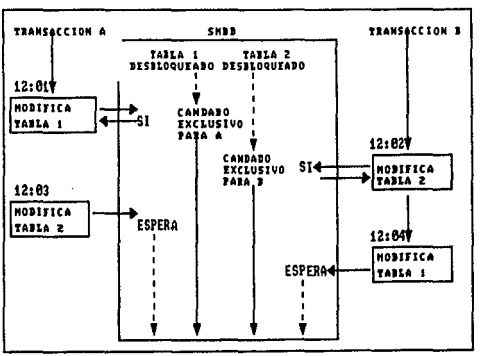


FIGURA 8. UN PUNTO MUERTO DE TRANSACCIONES.

Una función de los servidores de base de datos es el bloque automático para prevenir las concurrencias entre transacciones que puede llevar a la inconsistencia y pérdida de datos. Todos los servidores de base de datos manejan candados automáticos en sus tablas o a nivel registro.

Como la información está centralizada, los servidores permiten tener candados compartidos, permitiendo a los programas leer datos aunque estén bloqueados o bloquear datos cuando se está actualizando, previniendo que otros usuarios traten de actualizar antes de que termine la transacción.

Acceso a los datos

Para ofrecer acceso a las bases de datos distribuídas, la empresa IBM proporciona un excelente esquema para comprender la administración de los datos distribuídos, basados en cuatro etapas:

1. Petición remota, En esta etapa, el usuario de una PC puede emitir una sentencia SQL que consulte o actualize datos en una única base de datos remota.

2. Transacción remota, Aquí el usuario de PC puede emitir una serie de sentencias SQL que consulten o actualizen datos en una base de datos remota y luego comisionar /5 o volver atrás la serie entera de sentencias como una única transacción, esta transacción requiere al menos un SMBD sobre la PC además del sistema donde está localizada la base de datos, el SMBD debe extenderse a través de la red para asegurar que los sistemas local y remoto tengan siempre la misma opinión referente, así la transacción ha sido comisionada. Sin embargo la responsabilidad efectiva de mantener la integridad de la base de datos sigue estando en el SMBD remoto, por ejemplo la arquitectura cliente-servidor del SMBD SQL SERVER se basa en una capacidad de transacción remota, con el usuario localizado en una PC y la base de datos localizada a través de una LAN sobre un sistema servidor.

3. Transacción distribuída, En este caso, cada sentencia SQL individualmente consulta o actualiza una única base de datos sobre un único sistema de cómputo remoto. Sin embargo, la secuencia de sentencias SQL dentro de una transacción puede acceder a dos o más bases de datos localizadas sobre sistemas diferentes.

/5 DAR CUMPLIMIENTO.

Cuando la transacción se comisiona o se vuelve atrás, el SMBD garantiza que todas las partes de la transacción, sobre todos los sistemas empleados en la transacción serán comisionados o vueltos atrás. El SMBD garantiza especificamente que no habrá una transacción parcial, en donde la transacción se comisiona en un sistema y vuelta atrás en otro.

4. Peticiones distribuídas, en esta etapa, una sola sentencia SQL puede referenciar tablas de dos o más bases de datos localizadas en diferentes sistemas. El SMBD es responsable de llevar a cabo automáticamente la sentencia a través de la red. Una secuencia de sentencias de petición distribuída pueden agruparse para formar una transacción; el SMBD debe garantizar la integridad de la transacción distribuída sobre todos los sistemas que estén implicados. La figura 9 muestra los 4 pasos de acceso distribuído.

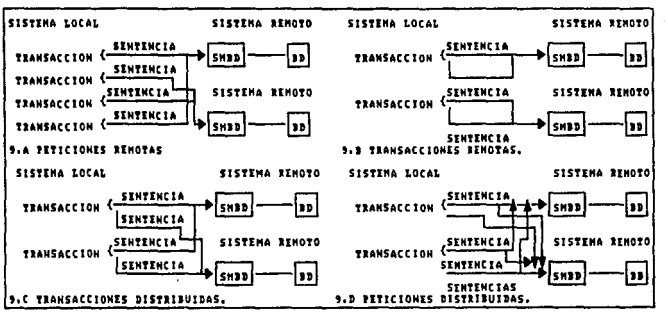


FIGURA 9. PASOS DE ACCESO DISTRIBUIDO.

Cuando dos o más usuarios acceden concurrentemente a una base de datos, el SMBD basado en SQL no solamente debe recuperarse adecuadamente de los fallos o errores del sistema, también debe asegurarse que las acciones de los usuarios no interfieran unas con otras. Idealmente, cada usuario debería ser capaz de acceder a la base de datos como si tuviera acceso exclusivo a ella, sin preocuparse de las acciones del resto de los usuarios.

Si dos transacciones, A y B se están ejecutando concurrentemente, el SMBD basado en SQL asegura que los resultados serán los mismos en cualquier caso tanto si: la transacción A se ejecuta primero, seguida de la B, como si la B se ejecuta primero, seguida de la A. Este concepto es conocido como la serialidad de las transacciones. Efectivamente, significa que cada usuario de la base de datos puede acceder a ésta como si no hubiera otros usuarios accediendo concurrentemente a ella.

El hecho de que SQL aisle a un usuario de las acciones de otros usuarios concurrentes no significa, sin embargo, que se pueda ignorar todo acerca de los otros usuarios. De hecho la situación es en gran medida la opuesta. Puesto que otros usuarios desean actualizar concurrentemente la base de datos, deberían mantenerse las transacciones tan breves y simples como fuera posible, para maximizar la cantidad de procesamiento paralelo que pueda generarse.

Continuando con acceso a los datos; un SMBD no permite que un usuario o programa de aplicación sepa dónde se almacenan los datos. Los datos remotos deben accesarse y procesarse del mismo modo que los locales. Esto se conoce como trasparencia de localización. (ver figura 10).

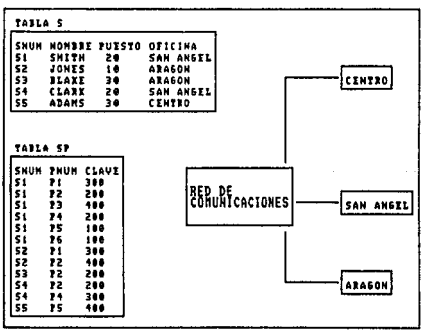


FIGURA 10. UN SMBD DISTRIBUIDO PERMITE A LOS DATOS REMOTOS SER ACCESADOS Y PROCESADOS EN LA MISMA FORMA QUE LOS DATOS LOCALES.

Al SMBD corresponde averiguar que una tabla se almacena localmente (tabla S) y la otra en Aragón (tabla SP). En la mayoría de los productos de SMBD, se utiliza un diccionario de datos para grabar el nombre de cada tabla y su localización física. Lo ideal es que estos diccionarios soporten un sistema global de nombramientos y manejen situaciones con tablas de nombres duplicadas en la red.

Al tratar con sistemas distribuídos, se debe distinguir entre proceso distribuído de base de datos y proceso distribuído de aplicaciones. Un verdadero sistema distribuído de base de datos permite que éstos se distribuyan a través de la red y que las aplicaciones tengan acceso a estos datos como si se les almacenara localmente. En tal ambiente, el SMBD provee facilidades (detección de puntos muertos y otros) para asegurar la integridad cuando las aplicaciones leen y modifican los datos que se guardan en múltiples ubicaciones físicas. Este tipo de sistema distribuído sería necesario en una instalación para descentralizar el proceso operacional. Como por ejemplo, al almacenar los datos localmente en vez de en una base central de datos. Este tipo de proceso operativo tiene requerimientos muy estrictos de integridad y desempeño.

Por otro lado, en el proceso distribuído de aplicaciones la distribución se da en las aplicaciones, en vez de en los datos. Una arquitectura Cliente-Servidor constituye un buen ejemplo de este tipo de procesamiento. En este ambiente, parte del procesamiento lo realiza una estación de trabajo del cliente y otra parte un servidor remoto de base de datos (la estación de trabajo del cliente maneja los servicios de presentación y la lógica del procesamiento, mientras que el servidor de base de datos ejecuta los comandos SQL que le fueron pasados a través de la red, desde la estación de trabajo del cliente). El proceso Servidor-Cliente con frecuencia se utiliza para dar acceso a los datos guardados en una computadora departamental remota o en una base de datos centralizada a los usuarios de estación de trabajo.

Desde luego, es posible usar una combinación de proceso distribuído de aplicación y de proceso distribuído de base de datos. En el ambiente Cliente-Servidor, por ejemplo, el servidor remoto de base de datos puede ser una parte de un sistema distribuído de base de datos, permitiendo que tanto el proceso de los datos como el de las aplicaciones sea distribuído.

El acceso a datos remotos tiene un efecto negativo sobre el desempeño, debido a la inconveniencia de mandar solicitudes de base de datos y los datos resultantes a través de la red.

De cualquier modo, los SMBD relacionales tienen una menor inconveniencia con las redes que una base de datos con tecnología más antigua. Por otro lado, los sistemas relacionales procesan una serie de registros a la vez.

La capacidad de procesado por nivel de serie del SQL permite a los usuarios, en una sola solicitud, especificar la serie de registros que requieren de un sistema remoto; también permite que el sistema remoto mande un lote de filas de resultados de regreso a través de la red y así reduce el tiempo para obtener los datos remotos.

Otro aspecto arquitectónico de un SMBD relacional que puede reducir el tiempo de acceso para los datos distribuídos, es el optimizador SQL. La labor del optimizador es analizar una sentencia SQL y determinar el modo más eficiente de tener acceso físico a los datos.

En un ambiente distribuído, el optimizador relacional debe calcular los costos adicionales de la red para obtener datos remotos cuando determina las rutas de acceso de datos que se usarán. Esto es llamado optimización de la consulta distribuída. Muchos productos no tienen esta característica. En lugar de esto, simplemente separan los sentencias SQL y mandan los fragmentos apropiados a los sistemas locales y remotos para su procesamiento.

En un sistema distribuído de base de datos, los datos se almacenan donde se usan más constantemente, lo que minimiza la cantidad de proceso remoto que se tiene que hacer. De cualquier modo, parte de los datos pueden ser obtenidos frecuentemente desde varias ubicaciones distintas. En el ejemplo de la figura 10, puede ser que San. Angel utilice el 35% y el 5% restante por Centro. Algunos sistemas distribuídos de base de datos tienen la habilidad de guardar una copia o réplica de una tabla en una o más ubicaciones remotas.

Si en el ejemplo, se pudiera tener una réplica de la tabla S del 60% al 95%. Por supuesto esto es cierto, solo para el acceso de lectura (si una aplicación modifica la copia de Aragón, el SMBD debe enviar los cambios de regreso a San. Angel para mantener actualizada la versión de esta última colonia). Es importante mencionar que el mantenimiento y el acceso de las réplicas es manejado por el SMBD y es normalmente transparente para las aplicaciones y para los usuarios.

3.1.2 CONSISTENCIA EN LA BASE DE DATOS.

La consistencia de una base de datos constituye un depósito seguro para los datos. Sin embargo, existe un área en la que es posible que se generen fallas, aunque todo lo demás aparentemente funcione bien. Esta área problemática existe debido a la interferencia de múltiples transacciones que están activas dentro de un sistema de computación. La interferencia puede estar relacionada con la competencia así como la cooperación. Dentro de la competencia, si en una operación de base de datos existen recursos adicionales que deben compartirse. Todos los usuarios desearán tener acceso al esquema, muchos tal vez seleccionen un archivo en particular, o un índice y posiblemente varios deseen tener acceso al mismo dato. Los datos son perfectamente consistentes hasta que una o más transacciones intenten hacer modificaciones. Entonces posiblemente a algunos de los usuarios se les niegue el acceso a los datos para evitar diseminar informacion inconsistente; dentro de la cooperacion cuando una transaccion ha diseminado múltiples procesos, es necesario coordinar sus actividades. Es posible que estos procesos consten de múltiples secciones y que a su vez diseminen nuevos procesos.

Mientras se realiza una transacción de actualización, la consistencia de una base de datos puede estar temporalmente alterada. Un dato reflejará la actualización, mientras que otro aún no ha sido modificado y debido a que las transacciones concurrentes pueden ocasionar inconsistencia en los datos, las técnicas vistas para la concurrencia , también sirven como herramienta para mantener la consistencia de los datos.

En un SMBD basado en SQL, un conjunto de sentencias forman una unidad de trabajo llamada transacción, esta sola transacción indica al SMBD que la secuencia de sentencias entera debe ser ejecutada autómaticamente (todas las sentencias deben completarse para que la base de datos esté en un estado consistente), es decir o todas las sentencias son ejecutadas con éxito, o ninguna de las sentencias es ejecutada.

El SMBD es responsable de mantener este compromiso incluso si el programa de aplicación aborta o se produce una falla de hardware a mitad de la transacción, tal como se ve en la figura 11. En cada caso, el SMBD debe asegurarse que cuando se complete una recuperación de la falla, la base de datos nunca refleja una <<transacción parcial>>.

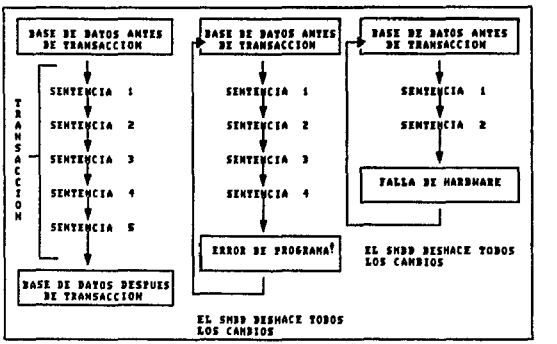


FIGURA 11.

SQL soporta las bases de datos mediante dos sentencias:

COMMIT y ROLLBACK.

La sentencia COMMIT señala el final correcto de una transacción. Informa al SMBD que la transacción está ahora completa; todas las sentencias que forman la transacción han sido ejecutadas, y la base de datos es auto consistente.

La sentencia ROLLBACK señala el final sin éxito de una transacción. Informa al SMBD que el usuario no desea completar la transacción; en vez de ello, el SMBD debe deshacer los cambios efectuados a la base de datos durante la transacción. En efecto, el SMBD restaura la base de datos a su estado antes que la transacción comenzara.

LOS SMBD basados en SQL como SYBASE y SQL SERVER, incluyen 4 sentencias de procesamiento de transacciones:

1) BEGIN TRANSACTION. Señala el comienzo de una transacción.

2) COMMIT TRANSACTION. Señala el final con exito de una transacción

3) SAVE TRANSACTION. Establece un punto de guarda a mitad de una transacción.

4) ROLLBACK TRANSACTION. Tiene 2 papeles. Si se designa un punto de guarda en la sentencia ROLLBACK, el SMBD deshace los cambios de la base de datos efectuados desde el punto de guarda, dando marcha atrás efectivamente a la transacción hasta el punto en donde la sentencia SAVE TRANSACTION fue ejecutada. Si no hay ningún punto de guarda designado, la sentencia ROLLBACK deshace todos los cambios efectuados desde la sentencia BEGIN TRANSACTION. Ver figura 12

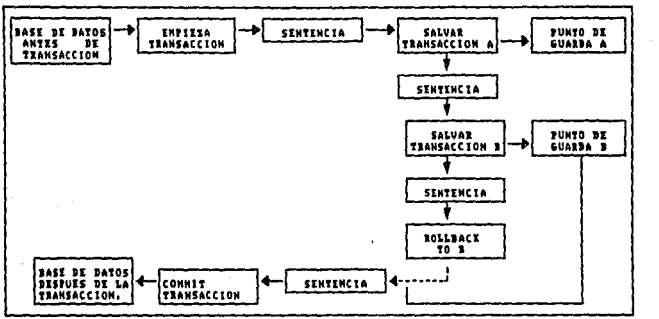


FIGURA 12.

¿Como puede el SMBD basado en SQL deshacer los cambios efectuados a una base de datos, especialmente si ocurre un fallo del sistema a mitad de una transacción? Las técnicas actuales utilizadas por los productos SMBD basados en SQL varian, pero casi todos ellos se basan en un registro de transacción, tal como se muestra en la figura 13.

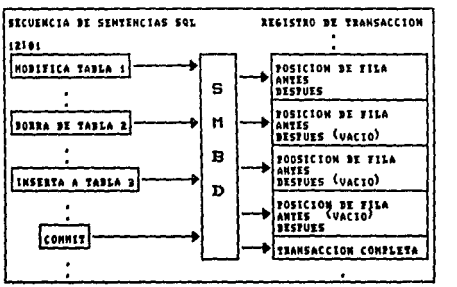


FIGURA 13

He aquí cómo funciona el registro de transacción, de forma simplificada. Cuando un usuario ejecuta una sentencia SQL que modifica la base de datos, el SMBD escribe automáticamente una anotación en el registro de transacción mostrando dos copias de cada fila afectada por la sentencia.

Una copia muestra la fila antes del cambio y la otra copia muestra la fila después del cambio. Sólo después que se escribe el registro modifica el SMBD realmente la fila en el disco. Si el usuario ejecuta posteriormente una sentencia COMMIT, el fin de transacción se anota en el registro de transacción. Si el usuario ejecuta una sentencia ROLLBACK el SMBD examina el registro para encontrar las imagenes <<de antes>> de las filas que han sido modificadas desde que comenzó la transacción. Utilizando estas imagenes, el SMBD restaura las filas a su estado anterior, deshaciendo efectivamente todas las modificaciones a la base de datos efectuadas durante la transacción.

SQL garantiza que los datos recuperados durante una transacción serán autoconsistentes, no afectados por transacciones de otros usuarios. Esto significa que una vez que el programa recupera un registro de la base de datos, ningún otro usuario puede modificar el registro hasta que la transacción finalice, ya que el programa podría tratar de recuperar el registro de nuevo posteriormente dentro de la misma transacción, y el SMBD debe garantizar que se examinen los mismos datos.

Si suponemos que existe una BDD, y se tiene una tabla de depósito, como la de la figura 14, que se va almacenar en la base de datos. Hay varios factores que deben tomarse en cuenta al almacenar esta tabla en la base de datos. Estos son repetición, fragmentación y repetición- fragmentación.

NOMBRE-SUC	NUMERO-CUENTA	NOMBRE-CLIENTE	SALDO
HIDALGO	305	LOPEZ	500
HIDALGO	226	BENITEZ	336
VALLE	117	BENITEZ	295
VALLE	402	GONZALEZ	1000
HIDALGO	155	GONZALEZ	62
VALLE	408	GONZALEZ	1123
VALLE	639	GARCIA	750

FIGURA 14. EJEMPLO DE LA TABLA DEPOSITO

- Repetición. El sistema mantiene varias copias idénticas de la tabla. Si la tabla depósito está repetida, se almacena una copia en dos o más localidades, en caso extremo se tiene repetición total, en la que se almacena una copia de la relación en cada una de las localidades del sistema.

Existen algunas ventajas en la repetición, por ejemplo, si falla una de las localidades que contiene la tabla depósito, puede disponerse de ésta en otra localidad, pudiéndo continuar consultas que impliquen a depósito a pesar de haber fallado una localidad. Mientras más copias de depósito existan, mayor será la probabilidad de que los datos requeridos se encuentren en la localidad donde se está ejecutando la transacción.

Existen actualizaciones que en un SMBD son triviales, pero en un SMBDD se pueden convertir en complejas, ya que debe asegurarse en este caso, que todas las copias de la tabla depósito sean consistentes, pues de otra manera pueden hacerse cálculos erróneos. Esto implica que cada vez que se actualice depósito, la actualización debe propagarse a todas las localidades que contengan copias, lo que resulta en un mayor tiempo extra. Por ejemplo, en un sistema bancario, donde se repite la información de cuentas en varias localidades, es necesario que las transacciones garanticen que el saldo de una cuenta determinada sea el mismo en todas las localidades, es por eso que es escencial que un SMBD tenga la habilidad de proteger la consistencia de una transacción distribuida.

- Fragmentación. Aqui cada tabla se divide en varios fragmentos, donde cada fragmento se almacena en una localidad diferente y contiene información suficiente para reconstruir la tabla. Existen dos esquemas diferentes para fragmentar una relación: fragmentación horizontal y vertical. La fragmentación horizontal divide a la relación asignando cada tupla de depósito a uno o más fragmentos, por ejemplo, la tabla puede dividirse en n fragmentos diferentes, cada uno de los cuales consiste en tuplas de cuentas que pertenecen a una sucursal determinada. si el sistema tiene sólo dos sucursales, Hidalgo y Valle, entonces existen dos fragmentos diferentes:

depósito1 = nombre-suc = "Hidalgo""
depósito2 = nombre-suc = "valle"

Estos dos fragmentos se muestran en la figura 15.

NOMBRE-SUC	NUMERO-CUENTA	NOMBRE-CLIENTE	SALDO
HIDALGO	305	LOPEZ	500
HIDALGO	226	BENITEZ	336
HIDALGO	155	GONZALEZ	62

a)

NOMBRE-SUC	NUMERO-CUENTA	NOMBRE-CLIENTE	SALDO
VALLE	117	BENITEZ	205
VALLE	402	GONZALEZ	1000
VALLE	408	GONZALEZ	1123
VALLE	639	GARCIA	750

b)

FIGURA 15. FRAGMENTACION HORIZONTAL DE LA RELACION DEPOSITO.

La fragmentación vertical se lleva a cabo al agregar un atributo especial, llamado id-tupla, al esquema R. Un id-tupla es una dirección física o lógica de una tupla.

Puesto que cada tupla en depósito debe tener una dirección única, el atributo id-tupla es una llave del esquema ampliado.

En la figura 16. se muestra la tabla depósito, que es la tabla depósito de la figura 14 ya con id-tuplas. La figura 17. muestra una descomposición vertical del esquema-depósito U id-tupla en:

Esquema-depósito-3=(nombre-suc, nombre-cliente, id-tupla)
Esquema-depósito-4=(numero-cuenta, saldo, id-tupla).

NOMBRE-SUC	NUMERO-CUENTA	NOMBRE-CLIENTE	SALDO	ID-TUPLA
HIDALGO	305	LOPEZ	500	1
HIDALGO	226	BENITEZ	336	2
VALLE	117	BENITEZ	205	3
VALLE	402	GONZALEZ	1000	4
HIDALGO	155	GONZALEZ	62	5
VALLE	408	GONZALEZ	1123	6
VALLE	638	GARCIA	750	7

FIGURA 16. LA TABLA DE DEPOSITO DE LA FIG. 14 CON ID-TUPLA

NOMBRE-SUC	NOMBRE-CLIENTE	ID-TUPLA
HIDALGO	LOPEZ	1
HIDALGO	BENITEZ	2
VALLE	BENITEZ	3
VALLE	GONZALEZ	4
HIDALGO	GONZALEZ	5
VALLE	GONZALEZ	6
VALLE	GARCIA	7

a)

NUMERO-CUENTA	SALDO	ID-TUPLA
305	500	1
226	336	2
117	205	3
402	1000	4
155	62	5
408	1123	6
638	750	7

b)

FIGURA 17. FRAGMENTACION VERTICAL DE LA RELACION DEPOSITO

Puesto que el valor de id-tupla representa una dirección, es posible formar una pareja con una tupla de depósito-3 y la tupla correspondiente de depósito-4 empleando la dirección dada por el valor de id-tupla. Esta direción permite recuperar en forma directa la tupla sin necesidad de un índice. Existe también la fragmentación mixta en la cual se aplican fragmentos ya sean horizontales o verticales sobre la tabla.

- Repetición-Fragmentación. Esta es una combinación de los dos conceptos antes mencionados. la relación se divide en varios fragmentos. El sistema mantiene varias copias idénticas de cada uno de los fragmentos.

En las BDD se involucran varias localidades, las cuales pueden ser susceptibles a fallas como interrupción de una línea de comunicación, falla total de una localidad, pérdida de mensajes y fragmentación de la red.

Por tanto, para que el sistema sea consistente, es menester que detecte cualquiera de estas fallas, que reconfigure el sistema de manera que pueda reaunudarse el proceso, y que se recupere una vez que haya sido reparado el procesador o la línea de comunicación afectados, por ejemplo si una localidad se da cuenta de que existe una falla, debe iniciar un procedimiento de reconfiguración que le permita mantener la consistencia y continuar con sus operaciones normales, es decir:

- Si en la localidad que está fuera de servicio se almacena información repetida, debe actualizarse el catálogo de manera que las consultas no hagan referencia a la copia que se encuentra en dicha localidad.

- Si en el momento de presentarse la falla, existían transacciones activas en la localidad que quedó fuera de servicio, deben abortarse. Es conveniente abortar estas transacciones tan pronto como sea posible, ya que puede darse el caso de que hayan puesto candados a información que se encuentra en localidades que siguen activas.

El esquema de reconfiguración que se adopte para resolver una falla, debe estar planeado para que funcione correctamente aun cuando la red quede fragmentada; cuando una localidad que quedó fuera de servicio se recupera, debe iniciar un procedimiento de actualización de sus tablas para que reflejen los cambios que tuvieron lugar mientras estaba inactiva.

En una BDD para garantizar una atomicidad o una mayor consistencia en la base de datos, es preciso que todos las localidades en las que se ejecuta una transacción T coincidan en el resultado final de la ejecución.

T debe quedar cometida o abortada en todas las localidades. Para garantizar ésta consistencia, los SMBD comerciales manejan un método llamado Protocolo de Comisión de Dos Fases(PCDF). Para entender mejor en la figura 18 se muestra, un ejemplo de PCDF en un SMBD basado en SQL.

1. El programa del Sistema A emite un COMMIT para la transacción (distribuida) actual, que ha actualizado tablas en el sistema B y el sistema C. El sistema A actuará como coordinador del proceso de comisión, coordinando las actividades del software SMBD en los sistemas B y C.

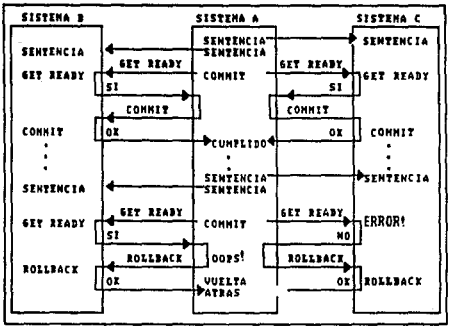


FIGURA 10. EL PROTOCOLO DE COMISION DE DOS FASES.

2. El sistema A envía un mensaje GET READY a los sistemas B y C, y anota el mensaje a su propio registro de transacciones.

3. Cuando el SMBD del sistema B o C recibe el mensaje GET READY, debe prepararse para comisionar o para volver atrás la transacción actual. Si el SMBD puede pasar a este estado de <<listo para comisionar>>, replica SI al sistema A y anota este hecho en su registro de transacciones locales; Si no puede pasar a este estado, replica NO.

4. El sistema A espera las contestaciones a su mensaje GET READY. Si todas las réplicas son SI, el sistema A envía un mensaje COMMIT al los sistemas B y C, y anota la decisión en su registro de transacciones. Si alguna de las réplicas es NO, o si todas las réplicas no se reciben antes de que se cumpla un cierto plazo de tiempo, el sistema A envía un mensaje ROLLBACK a ambos sistemas y anota esa decisión en su registro de transacciones.

5. Cuando el SMBD de los sistemas B o C recibe el mensaje COMMIT o ROLLBACK, debe hacer lo que se dice. El SMBD cede la capacidad de decidir el destino de la transacción autónomamente cuando replicó SI al mensaje GET READY del paso 3.

   El SMBD comisiona o vuelve atrás su parte de la transacción según se le indica, escribe el mensaje COMMIT o ROLLBACK en su registro de transacciones, y devuelve un mensaje OK al sistema A.

6. Cuando el sistema A ha recibido todos los mensajes OK, sabe que la transacción ha sido comisionada o vuelta atrás y devuelve el valor SQLCODE /6 adecuado al programa.

El protocolo de comisión de dos fases garantiza la integridad de las transacciones distribuídas, pero genera una gran cantidad de tráfico de red.

Debido a su fuerte recargo de red, las transacciones distribuídas pueden tener un efecto negativo serio sobre el rendimiento de la base de datos. Por esta razón, las bases de datos distribuídas deben ser cuidadosamente diseñadas para que los datos frecuentemente accedidos (o al menos frecuentemente actualizados) se encuentren en un sistema local o en un único sistema remoto. Si es posible, las transacciones que actualizan dos o más sistemas remotos deberían ser una ocurrencia relativamente rara.

/6 VARIABLE QUE INDICA TERMINACION O ERROR DE LA SENTENCIA

3.1.3. SEGURIDAD.

Los productos de SMBD basados en red, han tomado como hecho importante la seguridad de los registros. El SMBD debe proteger a los usuarios que editan el mismo archivo, de tal manera que no se sobre escriban. Existen tres tipos de mecanismos de seguridad de registros: Automático, manual o interactivo.

Automático.- Este método protege un registro siempre que un usuario lo selecciona para edición.

Manual.- Este método da control de la seguridad de un registro al usuario.

Interactivo.- Este método permite a muchos usuarios editar el mismo registro simultáneamente. Los cambios hechos por cualquier usuario aparecen instantáneamente en las pantallas de los otros usuarios.

Por ejemplo, los SMBD DBASE IV y BTRIEVE proporcionan llaves de registro automático y manual. Sin embargo, los usuarios de DBASE IV han reportado problemas con esta llave de registro automático. DBASE IV no apoya directamente la seguridad de registro interactivo, sí proporciona una herramienta alrededor de él. La función CHANGE, muestra si otro usuario ha cambiado el registro que se está editando. Esta función permite a los usuarios construir su propio mecanismo de seguridad de registro interactivo, el SMBD CLIPPER solo proporciona seguridad de registros manual y el DATAFLEX apoya la seguridad de registro manual, así como interactivo.

Como se sabe, la seguridad de los datos almacenados es un factor importante, si no hasta primordial. Los SMBD basados en SQL dan una importancia especial a este punto, tratando de sastisfacer algunos de los requerimientos de seguridad más comunes como:

- Los datos de cualquier tabla dada deberían ser accesibles a algunos usuarios, pero el acceso a otros debería ser impedido.
- Algunos usuarios deberían tener permitido actualizar datos en una tabla particular; a otros sólo se les debería permitir recuperar datos.
- Para algunas tablas, el acceso debería estar restringido en base a las columnas.

El SMBD es responsable de implementar un esquema de seguridad. El lenguaje como SQL define un panorama general para la seguridad de la base de datos, y las sentencias SQL se utilizan para especificar restricciones de seguridad. El esquema de seguridad de SQL se basa en tres conceptos principales: usuarios, los objetos de la base de datos y los privilegios.

- **Los usuarios** son los actores de la base de datos. Cada vez que el SMBD recupera, inserta, borra o actualiza datos, lo hace a cuenta de algún usuario. El SMBD permitirá o prohibirá la acción, dependiendo de qué usuario esté efectuando la petición.

  Cada usuario de una base de datos basada en SQL tiene asignado un id-usuario, un nombre breve que identifica al usuario dentro del software SMBD. EL id-usuario se encuentra en el núcleo de la seguridad SQL. Toda sentencia SQL ejecutada por el SMBD se lleva a cabo a cuenta de un id-usuario específico. El id-usuario determina si la sentencia va a ser permitida o prohibida por el SMBD.

  La mayoría de las implementaciones SQL comerciales establecen un id-usuario para cada sesión de base de datos.

  Generalmente debe suministrarse tanto un id-usuario como una contraseña para verificar que el usuario está en efecto autorizado a utilizar el id-usuario que suministra. Aunque los id-usuario y las contraseñas son habituales en la mayoría de los productos SQL, las técnicas específicas utilizadas para especificar el id-usuario y la contraseña varían de un producto a otro.

  Muchos otros productos SMBD, incluyendo INGRES e INFORMIX, utilizan los nombres de usuarios del sistema operativo del computador como id-usuario de la base de datos. Por tanto no se tiene que especificar un id-usuario de base de datos y una contraseña aparte.

  La seguridad también se aplica al acceso programado a una base de datos, de modo que el SMBD debe determinar y validar el id-usuario para cada programa de aplicación que pretenda acceder a la base de datos. De nuevo, las técnicas y reglas para establecer el id-usuario varian de un producto de SMBD a otro. Cuando existe usuarios con necesidades similares, el esquema de seguridad de SQL ANSI/ISO /7 , se pueden manejar grupos de usuarios en uno de dos modos:

/7 ESTANDAR OFICIAL, PERO HAY MUCHAS VARIACIONES CON OTROS.

- Se puede asignar el mismo id-usuario a todas las personas de grupo, simplificando la administración de la seguridad, ya que permite especificar los privilegios de acceso a datos una vez para el único id-usuario, pero en este esquema las personas que comparten al id-usuario no se distinguen unas de otras en las visualizaciones del operador del sistema ni en los informes de SMBD.

- Se puede asignar un id-usuario diferente a cada persona del grupo permitiendo diferenciar a los usuarios y estableciendo privilegios diferentes para los usuarios individuales con posteridad.

  Los SMBD SYBASE y SQL SERVER ofrecen una tercera alternativa para manejar grupos de usuarios similares. Soportan id-grupo, que identifica grupos de id-usuario relacionados. Los privilegios pueden ser concedidos tanto a id-usuarios individuales como a id-grupo, y un usuario puede efctuar una accion de base de datos si se le permite por su privilegio bien como id-usuario o bien como id-grupo. Los id-grupos simplifican por tanto la administración de privilegios proporcionados a grupos de usuarios. Sin embargo, no son estándar y son específicos de SQL SERVER Y SYBASE.

- Los objetos de la base de datos son los elementos a los cuales se puede aplicar la protección de seguridad SQL. La seguridad se aplica generalmente a tablas y vistas /8 , pero otros objetos tales como formularios, programas de aplicación y bases de datos enteras también pueden ser protegidos. La mayoría de los usuarios tendrán permiso para utilizar ciertos objetos de la base de datos, pero tendrán prohibido el uso de otros.

  La mayoría de los productos SQL soportan objetos de seguridad adicionales. En una base de datos SQL SERVER, por ejemplo, un procedimiento almacenado es un objeto importante de la base de datos. El esquema de seguridad SQL determina que usuarios pueden crear y suprimir procedimientos almacenados y que usuarios tienen permitido ejecutarlos.

- Los privilegios son las acciones que un usuario tiene permitido efectuar para un determinado objeto de la base de datos. Un usuario puede tener permiso para seleccionar e insertar sobre filas en una tabla determinada, pero puede carecer de permisos para borrar

/8 TABLA VIRTUAL DEFINIDA MEDIANTE UNA CONSULTA, PARECE CONTENER FILAS Y COLUMNAS DE DATOS, AL IGUAL QUE UNA TABLA REAL, PERO LOS DATOS VISTOS SON LOS RESULTADOS DE LA CONSULTA.

o modificar filas de la tabla. Un usuario diferente puede tener un conjunto diferente de privilegios.

El conjunto de acciones que un usuario puede efectuar sobre un objeto de base de datos se denomina los privilegios para el objeto. El estándar SQL ANSI/ISO especifica privilegios para tablas y vistas, que casi son todas soportadas por el SMBD basado en SQL como:

- El privilegio de permitir recuperar datos de una tabla o vista (lectura).

- El privilegio de permitir insertar nuevas filas de datos en una tabla o vista (escritura).

- El privilegio de permitir modificar filas de datos de una tabla o vista (reescritura).

Si se crea una tabla, quien la hace se convierte en propietario y recibe todos los privilegios antes mencionados, otros usuarios no tienen inicialmente privilegios sobre la tabla creada, pero si se les puede proporcionar ciertos privilegios, estos usuarios tienen permitido el uso, pero no pueden transmitir esos privilegios a otros usuarios. De este modo el propietario mantiene un control muy estricto sobre quién tiene permisos para utilizar la tabla y sobre que formas de acceso se permiten.

Ocasionalmente puede desearse permitir a otros usuarios que concedan privilegios sobre un objeto que no es de su propiedad, una vez concedido estos privilegios, ese usuario puede conceder estos privilegios y la opción de concesión a otros usuarios. Aquellos otros usuarios pueden, a su vez continuar concediendo tanto los privilegios como la opción de concesión. Por esta razón se debería utilizar gran cuidado cuando se poporcionan a otros usuarios la opción de concesión.

En los SMBD basados en SQL, los privilegios que se han concedido pueden ser retirados todos o parte de los privilegios que previamente se han concedidos a uno o más id-usuarios, pero solo retirará aquellos privilegios que el propietario haya concedido a otro o otros usuarios. Ese usuario puede tener además privilegios que le fueron concedidos por otros usuarios y esos privilegios no quedan afectados por los privilegios retirados que el propietario haya emitido. Especificamente si dos usuarios diferentes conceden el mismo privilegio sobre el mismo objeto a un usuario y uno de ellos posteriormente revoca el privilegio, la concesión del segundo usuario seguirá permitiendo al usuario acceder al objeto.

Si se crea una vista, quien la hace se convierte en un propietario, sin embargo no recibe necesariamente privilegios totales sobre ella, para crear una vista el SMBD debe proporcionar el privilegio recuperar para la vista automáticamente. Para cada uno de los otros privilegios (insertar, borrar y modificar), el SMBD proporciona el privilegio sobre la vista solamente si se dispone de ese mismo privilegio sobre todas las tablas fuente para la vista.

La figura 19 muestra como podrían ser utilizados estos conceptos de seguridad en un esquema de seguridad de una base de datos.

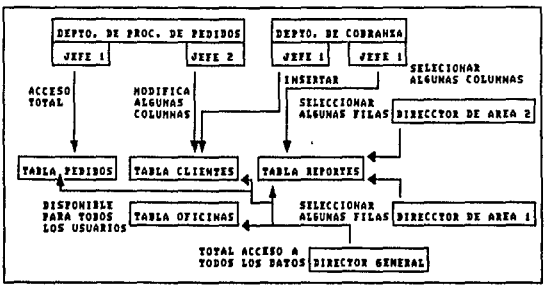


FIGURA 19

3.2 LIMITACIONES DE SISTEMAS MANEJADORES DE BASE DE DATOS EN UNA RED LOCAL.

La mayoría de los fabricantes de SMBD ofrecen la posibilidad integrada, de enlace en redes, teniendo técnicas (antes vistas) importantes para la manipulación de la base de datos. Algunos ofrecen la posibilidad de colocar un programa nuevo de software en una red para ofrecer acceso compartido a datos y lenguajes de programación que se pueden utilizar para desarrollar aplicaciones adecuadas a las necesidades de cada usuario, incluyendo menus y reportes, esta posibilidad de que múltiples usuarios tengan acceso a datos comunes en red es un beneficio enorme para la productividad, lo mismo que la tarea de actualizar y respaldar información en una empresa que comparte un conjunto de datos en común, se vuelve mucho más sencilla y más confiable.

No obstante, las versiones de SMBD para red tienen algunas limitaciones. Por ejemplo:

- Número máximo de tablas abiertas.
- Número máximo de indexaciones por tabla.
- Máximo de renglones por tabla.
- Máximo de campos por renglón.
- Tamaño máximo de registros.
- Contraseñas de seguridad para usuarios,bases de datos y tablas.
- Dar soporte a un amplio universo de plataformas multiusuario. Un SMBD debe poder conectarse con todos los recursos de la red cualquiera que sea su origen.
- Poder para proporcionar un control avanzado de impresoras en red. Teniendo como buena opción un servidor de base de datos.
- Capacidad de memoria y disco duro.
- Capacidad para acceso múltiple y simultáneo.
- Si está basado en SQL. Los productos SMBD basados en SQL han producido un mercado de herramientas de base de datos. Estas herramientas soportan el desarrollo de aplicaciones como 4GL (lenguajes de cuarta generación), paquetes de formularios y paquetes de ingeniería de software asistida por computadora (CASE Computer Aided Software Engineering), herramientas de acceso a datos,

tales como interfaces de consulta gráfica, programas de inspección de base de datos y escritores de informes, incluyendo utilidades de copias de seguridad, recuperación y monitores de rendimiento de base de datos. Herramientas gráficas de interfase de usuario que soportan entornos populares de PC y estaciones de trabajo.

- Poder para soporte de API. En este enfoque el programa se comunica con el SMBD a través de las llamadas API y utiliza estas llamadas para recuperar resultados de consulta. Este tratamiento no exige un precompilador.

- Apoyo a TTS (Transaction Tracking Systems), Sistema de registros de transacciones). El TTS lleva a cabo un registro completo de todas las aplicaciones transaccionales y permite hacer una auditoría de transacciones, regenerar completa la base de datos en caso de fallas, desde la primera hasta la última.

- Interfaz de usuario gráfico. SQL WINDOWS proporciona una interfaz de usuario gráfico bajo MS-WINDOWS, soportando el SQL BASE y otros SMBD basados en SQL.

Algunas de estas limitaciones se describen en los siguientes 4 SMBDs :

DBASE IV. 1.1. De ASHTON-TATE, con un número máximo de tablas abiertas de 10, indexaciones por tabla 57, máximo de renglones por tabla 1000 millones, máximo de campos por renglón de 255, 4000 bytes de tamaño máximo de records, soporta contraseñas (passwords) de seguridad para usuarios, bases de datos, tablas y niveles de acceso, en soporte multiusuario proporciona protección de archivos/tablas, apoya a impresoras en red pero solo en algunas plataformas, requiere 516k de memoria utilizable, trabaja bajo un número limitado de plataformas. Su generador de formas no soporta varias para la entrada de las bases de datos sin algunas manipulaciones del código,el paquete no soporte el ratón o mouse.Trabaja con el estandar DOS así como con NOVELL, 3COM, IBM PC-LAN, UNGER MANN-BASS NET/ONE. También se ejecuta en una MACINTOSH, aunque la versión MAC no es para múltiusuarios y no es completamente compatible con la versión para PC, como frontal proporciona acceso a las bases de datos de SQL SERVER a través de una LAN OS/2, requeriendo de un disco duro mínimo de 20mb, el dbase IV autónomo corre en una PC IBM estándar y soporta sentencias de SQL que accesan a una base de datos local. Cuando se quiere trabajar en red se requiere de una expansión de la memoria MS-DOS.

CLIPPER 5.01. De Nantucket, el número máximo de tablas abiertas es de 250, su máximo de indexación por tabla es de 15, el máximo de renglones por tabla es 1000 millones, con 1000 campos por renglón y 65,534 bytes como tamaño máximo de records, no hay contraseñas de seguridad ni para usuarios,bases, tablas y niveles de acceso, su soporte multiusuario es solo para protección de archivos/tablas, toma los programas escritos en DBASE para ejecutarse y los compila como programas independientes, no apoya a TTS, no tiene soporte explícito de impresoras en red, tiene un servidor de base de datos. Los usuarios pueden soportar estas limitaciones enlazando otros programas. Por ejemplo, CLIPPER recientemente anunció enlaces con el servidor SQL de NOVELL. También hay disponibilidad de agregados internos que proporcionan apoyo de impresoras en red, se ejecuta solo bajo cualquier red que soporte a la red DOS 3.1, incluyendo NOVELL, 3COM, BANYAN e IBM, los programas creados en CLIPPER pueden ejecutarse en mucho menos memoria que bajo DBASE, requiere un disco duro mínimo de 20mb y memoria mínima de 640k. Una limitación extra es que su generador de reporte no es tan flexible como otros.

INFORMIX-SQL. De INFORMIX, no tiene límites para tablas abiertas, indexaciones por tabla, renglones por tabla y campos por renglón, su tamaño máximo de records es de 32,768 bytes, sus contraseñas de seguridad solo son para usuarios y niveles de acceso, su soporte multiusuario es para protección de archivos/tablas y record de protección, no tiene pantallas de menús y ventanillas a la vista como los analizados posteriormente, como frontal de servidor de base de datos INFORMIX-ON LINE navega en los sistemas operativos para red LAN MANAGER/X, NETWARE y VINEX, el sistema operativo es MS-DOS, OS/2, XENIX, UNIX, DEC VMS, IBM VMS, MACINTOSH, los protocolos son IPX/SPX, NETBIOS, NAMED PIPES, TCP/IP y SNA, las hojas manejables en él son WINGZ, y SMARTWARE, soportando APIs, su interfase gráfica usada es WINGZ, está basado en SQL y sus lenguajes incorporados son ADA, C y COBOL.

PARADOX 3.5. De la empresa BORLAND, tiene un soporte máximo de 24 tablas abiertas, su número máximo de indexaciones por tabla es de 1 por campo, permite 256 millones de renglones por tablas y un máximo de 255 campos por renglón, tamaño máximo de records de 4000 bytes, indexados, permite todas las contraseñas de seguridad para usuarios, bases de datos, tablas y niveles de acceso, permite en el soporte multiusuario protección de archivos/tablas y record de protección.

Su capacidad recomendada en RAM es de 640 Kb y disco duro entre 3 y 4 Kb, contiene un generador de formas, de reportes y un lenguaje de base de datos. Tiene un lenguaje de programación robusto (PAL, Paradox Application Language), pero es recomendable dominar Paradox antes de enfrentar el PAL, los usuarios de Paradox no pueden editar el mismo archivo a menos que utilicen ambos la misma forma de tablas múltiples. De modo que si un usuario está actualizando la forma "empleado" de la compañía, otros usuarios no pueden usar otras formas lo cual presenta problemas en la red, otra limitación de Paradox en red, para editar registros es el "CO-EDIT" que no apoya a TTS, aunque Paradox si apoya a TTS no está efectivamente disponible en una red, Paradox no proporciona soporte de impresoras en red avanzada sobre lo que proporciona una red local. Los usuarios, que quieren un control completo de las impresoras de una red necesitarán otros productos, no corre bajo ninguna otra plataforma que no sea DOS, y en los sistemas operativos para red NetWare Ver. 2.15, Lan Manager, 3Com y Vines, no tiene producto de servidor de base de datos y es un producto que usa memoria RAM intensa, no considera en absoluto el SQL pero un software suministrado por Borland: AQL LINK, le permite incluir las consultas SQL en el lenguaje PAL y comunicarse con quienes alimentan a SQL. No contiene menús amigables para las herramientas de diseño y no especifica el contenido de ayuda por menú como lo hace DBASE IV.

4. EL MANEJADOR DE BASES DE DATOS SQL SERVER Y LA RED LOCAL.

Como se sabe el programa encargado de controlar una base de datos se denomina SMBD. En el caso de SQL cuando se necesita recuperar información de la base, el SMBD procesa la petición SQL, recupera los datos solicitados y los devuelve denominándose a este proceso QUERY o consulta, aclarando que SQL no es solamente una consulta, sino también es una herramienta utilizada para comunicarse y controlar todas las funciones que un SMBD desarrolla (definir, recuperar, manipular, compartir, e integrar datos entre otras).

SQL es también un lenguaje cliente/servidor donde los programas de la estación de trabajo utilizan SQL para comunicarse sobre una red de área local con servidores de base de datos que almacenan los datos compartidos, efectuando en esta arquitectura de cliente/servidor que tanto PCs como servidores trabajen mejor. En este papel SQL sirve como enlace entre los frontales optimizados para interacción con el usuario y los sistemas de apoyo (Back-end) especializados para administración de bases de datos, permitiendo también que las PCs funcionen como frontales de bases de datos mayores en minicomputadoras.

Uno de los principales SMBD basados en SQL es sin duda SQL SERVER anunciado en 1988 por las empresas Ashton-Tate y Microsoft trabajando en LANs basadas en el sistema operativo OS/2 y con la arquitectura cliente/servidor. Como se sabe en esta arquitectura existe un servidor de base de datos que almacena las bases de datos compartidas y las funciones del SMBD están divididas en dos partes. Los frontales (front-end) de base de datos como programas de aplicación que se ejecutan en las PCs y la máquina de soporte (back-end) de la base de datos que almacena y administra los datos ejecutándose en el servidor. Las funciones de intensiva relación con el usuario tales como el manejo de la entrada y vista de los datos, se concentran en el PC. Las funciones intensivas en proceso de datos, tales como la entrada/salida de archivos y el procesamiento de consulta, se concentran en el servidor de base de datos.

SQL SERVER soporta casi todas las sentencias del estándar oficial para SQL, el SQL ANSI (American National Standars Institute) y es uno de los mejores SMBD con mayor seguridad en la información y control de usuarios de una red local. La versión de SQL SERVER 1.11 para OS/2 cuesta $2,995 dólares hasta 10 usuarios y $7,995 dólares para más de diez.

4.1 SISTEMA SQL SERVER COMO MANEJADOR DE BASE DE DATOS EN LA RED LOCAL.

La introducción de SQL SERVER de Ashton Tate/Microsoft centró la atención en el potencial de SQL sobre redes de área local basadas en el sistema operativo OS/2. Comenzándose a examinar seriamente las redes de área local y la arquitectura cliente/servidor como alternativa a una minicomputadora centralizada para aplicaciones de base de datos. La figura 1 muestra la arquitectura cliente/servidor utilizada por SQL SERVER en una red de área local para la administración de bases de datos.

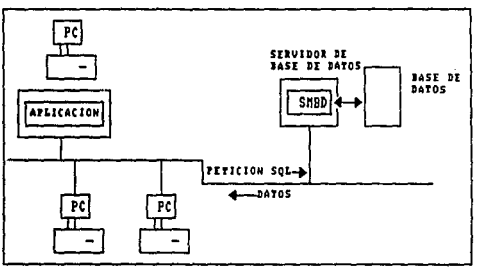


FIGURA 1. EL SMBD SQL SERVER EN UNA ARQUITECTURA CLIENTE/SERVIDOR

Para continuar este punto se harán notar algunas características únicas de SQL SERVER y la relación con el lenguaje SQL. Para analizar algunas cuestiones de SQL SERVER se utilizará a lo largo del tema, la estructura de la base de datos que se muestra a continuación.

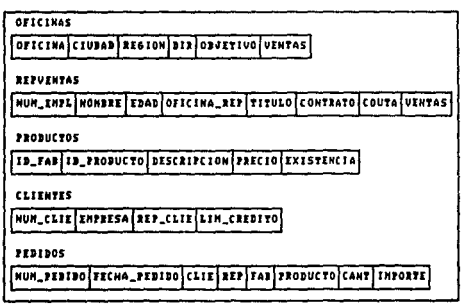


FIGURA 2. ESTRUCTURA DE LA BASE DE DATOS EJEMPLO

En todas las sentencias SQL una palabra clave como CREATE (crear), INSERT (insertar) y DELETE (borrar) son verbos típicos que nos fácilitan el entendimiento de este lenguaje, continuando con una o más cláusulas que comienzan con una palabra clave, tal como WHERE (donde), FROM (de) e INTO (dentro). En la figura 3 se muestra la estructura de una sentencia SQL.

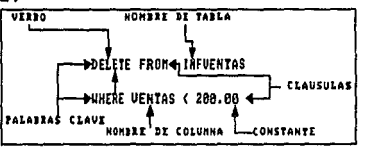


FIGURA 3. ESTRUCTURA DE UNA SENTENCIA SQL

Los tipos de datos soportados por SQL server son:

CHAR(n)	Cadenas de caracteres de longitud fija.
VARCHAR(n)	Cadenas de caracteres con longitud variable. Los datos VARCHAR permiten que una columna almacene cadenas de caracteres que varían de longitud de un renglón a otra, hasta una cierta longitud máxima.

```
TEXT              TEXTO.
TINYINT ----
SMALLINT     --->  ENTEROS.
INT      ----
MONEY               MONETARIOS.
FLOAT               COMA FLOTANTE
DATETIME            FECHA/HORA.
BIT                 BOOLEANO.
BINARY(n)  --
VARBINARY(n) -->  FLUJO DE BYTES.
IMAGE       --
SYSNAME     --
USER_TYPE_NAME -->OTROS.
```

Una consulta simple en SQL SERVER sería :

```
SELECT nombre,contrato
  FROM repventas
 WHERE  contrato >= '06/14/1991 12:00PM'
```

Que nos indica que seleccione las columnas nombre y contrato de la tabla ventas donde el contrato sea mayor o igual a la fecha y hora dada.

Si la fecha de contrato de un nombre vendedor estuviera almacenada en la base como el mediodía de junio 14, 1991, el vendedor no es incluído en los resultados de la consulta SQL SERVER. Si no se especifica una hora concreta en la fecha, SQL SERVER pone por omisión la hora de medianoche. Por tanto pudiésemos poner en WHERE:

```
WHERE contrato >= '06/14/1991'
```

Cuando hablamos de constantes numéricas éstas se deben escribir como números decimales ordinarios con el signo más o menos opcional por delante.

```
34   -45  3000.00   +3245.567
```

Las constantes en coma flotante se especifican utilizando la notación E. Donde E se lee 'por diez elevada a'.

```
2.5E2   -5.3455E1   2.5E-7
```

En las constantes de cadenas SQL SERVER se aceptan cadenas encerradas entre comillas dobles, así como sencillas.

```
'Juan Pérez' --> "Juan Pérez"
```

SQL SERVER soporta datos fecha/hora, y acepta una variedad de formatos diferentes para las constantes de fecha y hora. El SMBD acepta automáticamente todas las formas alternativas, y se pueden entremezclar si así se quiere. He aquí algunos ejemplos de constantes de fecha legales en SQL SERVER:

```
March 15,1991  Mar 15 1990  3/15/1991   3-15-90   1990 MAR 15
```

Y he aquí algunas constantes de tiempo legales:

```
15:30:25   3:30:25 PM   3:30:25 pm   3 PM
```

El lenguaje SQL incluye también constantes simbólicas especiales que devuelven valores de datos mantenidos por el propio SMBD como CURRENT DATE y CURRENT TIME, SQL SERVER, proporciona acceso a valores del sistema mediante funciones internas en lugar de constantes simbólicas. Un ejemplo es el siguiente:

```
SELECT nombre,contrato
  FROM repventas
 WHERE contrato > GETDAT E()
```

Donde se selecciona el nombre y contrato en la tabla repventas donde el contrato sea mayor que la fecha actual.

Si queremos visualizar todo el contenido de las columnas de una tabla SQL permite utilizar un (*) por ejemplo:

```
SELECT  *
  FROM  oficinas
```

oficina	ciudad	región	dir	objetivo	ventas
22	D.F..	CENTRO	108	$300,000	$186,000
11	MERIDA	SUR	110	$250,000	$248,000
13	PUEBLA	CENTRO	123	$128,000	$128,000

Los resultados de la consulta contiene las seis columnas de la tabla oficinas, en el mismo orden de izquierda a derecha que tienen en la tabla. Sin embargo, SQL SERVER trata el (*) como cualquier otro elemento de la lista de selección. Por tanto la consulta:

```
SELECT   *,(ventas-objetivo)
  FROM   oficinas
```

es legal para SQL SERVER.

Cuando se habla de consulta multitabla se dice que se solicitan datos procedentes de dos o más tablas en la base de datos por ejemplo:

```
SELECT   número,importe,empresa,lm-crédito
  FROM   pedidos,clientes
 WHERE   clie = num_clie
```

Aquí, se lista los pedidos mostrando su número, importe, empresa y crédito, utilizando dos tablas y la condición de búsqueda: clie = num_clie que es comparar columnas de dos tablas. Por otra parte el término <<composición>> se aplica a cualquier consulta que combina datos de dos tablas mediante comparación de los valores en una pareja de columnas de las tablas. La operación de composición SQL combina información procedente de dos tablas mediante la formación de pares de renglones relacionados en las dos tablas. Los pares de renglones que componen la tabla compuesta son los de las columnas correspondientes en cada una de las dos tablas que tienen el mismo valor. Si uno de los renglones de una tabla no se empareja en este proceso, la composición puede producir resultados inesperados como ilustran estas consultas:

Lista las vendedoras y las oficinas en que trabajan.

```
SELECT   nombre,oficina_rep
  FROM   repventas
```

nombre	oficina_rep
Blanca	13
María	11
Teresa	null
Paola	22
Nancy	12

Lista las vendedoras y las ciudades en que trabajan.

```
SELECT   nombre,ciudad
  FROM   repventas,oficinas
 WHERE   oficina_rep = oficina
```

nombre	ciudad
María	D.F..
Paola	Merida
Blanca	Puebla
Nancy	D.F..

Aparentemente, sería de esperar que estas dos consultas produjeran el mismo número de renglones, pero la primera lista muestra cinco vendedoras, y la segunda sólo cuatro, ya que Teresa está actualmente sin asignar y tiene un valor null en la columna oficina_rep. Este valor null no se corresponde con ninguno de los números de oficina en la tabla oficinas, por lo que oficina de Teresa en la tabla repventas está sin emparejar. Como resultado, desaparece en la composición.

Como queremos encontrar una consulta real, SQL SERVER permite generar resultados con un tipo de composición llamada <externa>, la cual es un componente útil del modelo relacional y un modo más natural de expresar un cierto tipo de petición de consulta. Por lo tanto la consulta sería:

```
SELECT   nombre,ciudad
  FROM   repventas,oficinas
 WHERE   oficina_rep *=* oficina
```

nombre	ciudad
Teresa	null
Maria	D.F..
Paola	Mérida
Blanca	Puebla
Nancy	D.F..

Para comprender mejor la composición externa, supongamos que queremos encontrar las parejas de chico/chica que viven en la misma ciudad incluyendo las chicas y chicos desparejadas, de las tablas mostradas::

Tabla chicas

nombre	ciudad
María	Mérida
Nancy	null
Susana	D.F..
Bety	D.F..
Ana	Puebla

Tabla chicos

nombre	ciudad
Juan	Mérida
Enrique	Mérida
Gabriel	null
Samuel	D.F..
Javier	Hidalgo

Tenemos que la consulta seria:

```
SELECT   *
  FROM   chicas,chicos
 WHERE   chicas_c *=* chicos_c   resultando
```

chicas.nombre	chicas.ciudad	chicos.nombre	chicos.ciudad
María	Mérida	Juan	Mérida
María	Mérida	Enrique	Mérida
Susana	D.F..	Samuel	D.F..
Bety	D.F..	Samuel	D.F..
Ana	Puebla	null	null
Nancy	null	null	null
null	null	Javier	Hidalgo
null	null	Gabriel	null

Si observamos los dos renglones de Ana y Nancy, provienen de renglones no coincidentes de la tabla chicas. Estos renglones han sido null-ampliadas emparejándolos a un renglón imaginario de todo null en las tablas chicos, y añadidas a los resultados de la consulta, lo mismo sucedió para Javier y Gabriel, esto nos indica que la composición externa es preservadora de información. Cada renglón de la tabla chicos está representado en los resultados (alguno más de una vez) y lo mismo para la tabla chicas. Finalmente podemos decir que la notación <<*=*>> utilizada por SQL SERVER, indica una composición externa, o sea para indicar la composición de dos tablas, TBL1 y TBL2, sobre las columnas COL1 y COL2, se coloca un asterisco delante y detrás del operador de comparación.

Pero para la composición externa a tres o más tablas SQL SERVER no las tiene aún bien definidas ya que es imposible especificar el orden de evaluación de las composiciones externas.

```
TBL1 comp-externa TBL2 comp-externa TLB3
```

Pero el resultado depende del orden en que se efectúen las operaciones de composición externa. Los resultados de:

```
(TBL1 comp-externa TBL2) comp-externa TBL3
```

serán en general diferentes de los resultados de

```
TBL1 comp-externa (TBL2 comp-externa TBL3).
```

4.2 LA INTEGRACION DE LA BASE DE DATOS EN LA RED.

Para preservar la consistencia y corrección de los datos almacenados, un SMBD relacional impone típicamente una o más restricciones de integridad de datos. Estas restricciones restringen los valores que pueden ser insertados en la base de datos o creados mediante una actualización de esta. SQL SERVER es un SMBD relacional que proporciona una capacidad de validación de datos al permitir crear una regla que determine que datos pueden ser introducidos en una columna particular. SQL SERVER comprueba la regla cada vez que se intenta una sentencia INSERT 1/ o UPDATE 2/ para la tabla que contiene la columna. Las reglas de SQL SERVER están escritas en el dialecto Transact-SQL utilizado por SQL SERVER. Por ejemplo, he aquí una sentencia Transact-SQL que establece una regla para una columna cuota en una tabla repventas:

```
CREATE RULE limite_cuota
        AS # VALUE BETWEEN 0.00 AND 500000.00
```

Esta regla impide insertar o actualizar una cuota con un valor negativo o un valor superior a $500.000. Como se muestra en el ejemplo, SQL SERVER permite asignar un nombre a la regla (<<limite_cuota>>) en este ejemplo). Sin embargo, las reglas SQL SERVER no pueden referenciar columnas u otros objetos de la base de datos.

SQL SERVER permite definir procedimientos en el transact-SQL que es un lenguaje de cuarta generación, siendo éstos compilados.

Cuando se habla de un disparador en SQL SERVER indica que para cualquier evento que provoca un cambio en el contenido de una tabla, un usuario puede especificar una acción asociada que el SMBD debería efectuar. Los tres eventos que pueden disparar una acción son INSERT, DELETE, o UPDATE renglones de tablas. La acción disparada por un evento se especifica mediante una secuencia de sentencias SQL, escritas en el dialecto Transact-SQL. Para entenderlo mejor supongamos que se añade un nuevo pedido a una tabla llamada pedidos, estos dos cambios también podrían tener lugar en la base:

- La columna ventas del vendedor que aceptó la orden debería incrementarse en el importe del pedido.

- El valor de existencias para el producto ordenado debería disminuir en la cantidad solicitada.

Esta sentencia de Transact-SQL define un disparador, de nombre nvopedido, que hace que estas actualizaciones sucedan automáticamente:

1/ INSERTA DATOS A LA TABLA ABIERTA.
2/ MODIFICA DATOS EN LA TABLA ABIERTA.

```
CREATE TRIGGER nvopedido
     ON pedidos
    FOR INSERT
     AS UPDATE repventas
          SET ventas = ventas + INSERTED.importe
         FROM repventas,INSERTED
        WHERE repventas.num_empl = INSERTED.REP
      UPDATE productos
          SET existencias = existencias - INSERTED.cant
         FROM productos,INSERTED
        WHERE productos.id_dir = INSERTED.fab
          AND productos.id_producto = INSERTED.producto
```

La primera parte le dice a SQL SERVER que el disparador va a ser invocado cada vez que se intente una sentencia INSERT sobre la tabla pedidos. Después de la palabra AS define la acción del disparador, siendo una secuencia de dos sentencias UPDATE, una para la tabla repventas y otra para productos. El renglón que está siendo insertado es referido utilizando el nombre de pseudotabla inserted dentro de las sentencias UPDATE. Como se muestra Transact-SQL extiende el lenguaje SQL sustancialmente para soportar disparadores. Otras extensiones no mostradas aquí incluyen tests IF/THEN/ELSE, iteraciones, llamadas de procedimientos, e incluso sentencias PRINT que visualizan mensajes de usuario. La principal ventaja de los disparadores es que las reglas comerciales pueden almacenarse en la base de datos y ser forzadas consistentemente con cada actualización de la base de datos, reduciendo sustancialmente la complejidad de los programas de aplicación que acceden a la base de datos.

El modelo de procesamiento de transacciones de el SMBD Sybase mostrado en el capítulo tres, también es utilizado por SQL SERVER, para la integridad de los datos. En la base de datos ejemplo, para la recepción de un nuevo pedido se genera una serie de cuatro actualizaciones a la base de datos:

- Añadir el nuevo pedido a la tabla pedidos.
- Actualizar el total de ventas para el vendedor que acepto el pedido.
- Actualizar el total de ventas para la oficina del vendedor.
- Actualizar el total de existencias para el producto ordenado.

Para dejar la base de datos en un estado autoconsistente, las cuatro actualizaciones deben producirse como una unidad. Si ocurre un error en el sistema u otro error crea una situación en donde algunas de las actualizaciones son procesadas y otras no, la integridad de la base de datos se perderá. Análogamente, si otro usuario calcula totales o porcentajes en medio de la secuencia de actualizaciones, los cálculos serán incorrectos. La secuencia de actualizaciones deben ser una proposición <<todo o nada>> en la base.

SQL SERVER proporciona precisamente esta capacidad a través de su característica de procesamiento de transacciones.

A veces es apropiado exigir que una columna que no es clave primaria de una tabla contenga un valor único en cada renglón. Por ejemplo, supongamos que se desea restringir los datos en la tabla repventas de modo que no pueda haber dos vendedores con exactamente el mismo nombre en la tabla. Se podría conseguir este objetivo imponiendo una restricción de unicidad sobre la columna nombre. EL SMBD fuerza una restricción de unicidad del mismo modo que fuerza la restricción de clave primaria. Cualquier intento de insertar o actualizar un renglón en la tabla que viole la restricción de unicidad fallaría. SQL SERVER utiliza la sentencia CREATE INDEX para implementar restricciones de unicidad.

4.2.1 CARACTERISTICAS

Para la creación de una base de datos SQL SERVER incluye una sentencia CREATE DATABASE como parte de su lenguaje de definición de datos. Una sentencia adicional DROP DATABASE destruye bases de datos creadas previamente. La sentencia CREATE TABLE define una nueva tabla en la base de datos y la prepara para aceptar datos, sin embargo esta sentencia varía de un producto SMBD a otro, ya que cada SMBD soporta su propio conjunto de tipos de datos y utiliza sus propias palabras claves, SQL SERVER difiere mucho de otros productos SMBD en su manejo de los valores null. El estándar especifica que una columna puede contener valores null a menos que específicamente se declare not null, SQL SERVER utiliza el convenio opuesto, suponiendo que los valores null no son permitidos a menos que la columna se declare explícitamente null.

La sentencia CREATE TABLE incluye típicamente una o más cláusulas opcionales que especifican características de almacenamiento físico para la tabla. Generalmente estas cláusulas sólo las utiliza el administrador de la base de datos para optimizar el rendimiento de una base de datos de producción. SQL SERVER ofrece un planteamiento que permite al administrador especificar múltiples archivos de bases de datos nominados que son utilizados para almacenar datos. La sentencia CREATE BATABASE de SQL SERVER puede especificar uno o más archivos de bases de datos:

```
CREATE DATABASE  datosop
    ON bdarch1,bdarch2,bdarch3
```

El planteamiento SQL SERVER permite que los contenidos de una base de datos sean distribuídos a través de varios volúmenes de discos.

La sentencia CREATE TIGGER crea un disparador. El disparador puede ser suprimído posteriormente de la base de datos utilizando la sentencia DROP TIGGER. la tabla 1 muestra como SQL SERVER utiliza los verbos CREATE, DROP, ALTER (crea, borrar y modificar).

CREATE/DROP/ALTER DATABASE	base de datos
CREATE/DROP DEFAULT	valor de columna por omisión
CREATE/DROP PROCEDURE	procedimiento almacenado
CREATE/DROP RULE	regla de integridad de columna
CREATE/DROP TIGGER	disparador almacenado

La figura 4 muestra la arquitectura SQL SERVER de bases de datos múltiples en donde cada una tiene asignado un nombre único

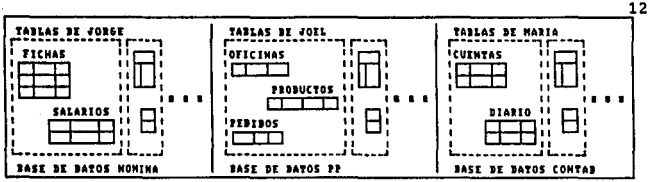


FIGURA 4. ARQUITECTURA DE BASES DE DATOS MULTIPLES

Como muestra la figura 4, cada una de las bases de datos está dedicada a una aplicación particular, esta arquitectura divide las tareas de administración de base de datos en partes más pequeñas y más manejables, donde cada persona responsable es el administrador de su propia base y si hay que añadir una nueva aplicación se desarrollará en su propia base de datos sin afectar a las demás. También es probable que los usuarios y los programadores puedan recordar la estructura general de sus propias bases de datos. Pero una desventaja de esta arquitectura es que las bases de datos individuales puedan convertirse en islas de información, desconectadas unas de otras.

SQL SERVER extiende la <<notación punto>> para nombres de tabla añadiendo como prefijo del nombre de la base de datos, el nombre del propietario, separado por un punto (.):

```
PP.Joel.Oficinas
```

Se refiere a la tabla Oficinas propiedad del usuario Joel en la base de datos de procesamiento de pedidos PP, y esta otra consulta compone la tabla repventas en la base de datos de nómina con la tabla oficina:

```
SELECT    PP.Joel.oficinas.ciudad,nomina.jorge.nombre
  FROM    PP.Joel.oficinas,payroll,jorge.repventas
 WHERE    PP.Joel.oficinas.dir = payroll.jorge.repventas.n_empl
```

Afortunadamente, tales consultas entre bases de datos son la excepción en vez de la regla, y normalmente pueden utilizarse nombres de usuario de base de datos por omisión.

Otra característica de SQL SERVER es cuando se quiere encontrar resultados detallados con subtotales, o para obtener subtotales multinivel, se añade una cláusula COMPUTE al final de la sentencia SELECT, la cual calcula subtotales y sub-subtotales como lo muestra el siguiente ejemplo:

Calcula los pedidos totales para cada cliente de cada vendedor, ordenados por vendedor, y dentro de cada vendedor por cliente.

```
SELECT    rep,clie,importe
  FROM    pedidos
ORDER BY    rep,clie
COMPUTE     SUM(importe) BY rep,clie
COMPUTE     SUM(importe),AVG(importe) BY rep
```

donde ORDER BY ordena por rep y luego clie dentro de rep, las funciones SUM() y AVG() suma los valores de la columna y encuentra el promedio de la columna respectivamente.

La clausula COMPUTE no es estándar en absoluto y de hecho es propia del dialecto Transact-SQL de SQL SERVER. Además, viola las reglas básicas de las consultas relacionales, ya que los resultados de la sentencia SELECT no son una tabla, sino una extraña combinación de diferentes tipos de renglones. No obstante como muestra el ejemplo puede ser muy útil.

Otras características como control de acceso, estadísticas de utilización, ventajas y limitaciones se explicarán más adelante.

4.2.1.1 CONTROL DE ACCESO.

En el capítulo 3 se habló de las etapas del acceso a datos distribuídos de IBM, las transacciones remotas requieren típicamente un SMBD sobre el PC además del sistema donde está localizada la base de datos. La lógica de transacción del SMBD debe extenderse a través de la red para asegurar que los sistemas local y remoto tengan siempre la misma opinión referente a si la transacción ha sido cumplimentada. Sin embargo, la responsabilidad efectiva de mantener la integridad de la base de datos sigue estando en el SMBD remoto. La arquitectura cliente/servidor de SQL SERVER se basa en una capacidad de transacción remota, con el usuario localizado en una PC y la base de datos localizada a través de un LAN sobre un sistema servidor.

También se estudió las transacciones distribuídas, donde SQL SERVER las soporta, pero pone el énfasis en la transacción distribuída sobre el programa de aplicación. El programa debe conectarse explícitamente dos SQL SERVER y debe enviar las sentencias SQL adecuadas sobre cada conexión. Cuando sea tiempo de cumplimentar o volver atrás una transacción, las sentencias COMMIT y ROLLBACK estándar no se utilizan. En su lugar, el programa utiliza un conjunto especial de llamadas de cumplimentación en dos fase en el API de SQL SERVER para sincronizar las cumplimentaciones de las dos copias de SQL SERVER. Pero este tipo de transacción ha sido criticado por su falta de transparencia.

Para el acceso a una base de datos en una arquitectura de bases de datos múltiples, resulta más complejo, ya que debe informarse al SMBD de qué base de datos se desea utilizar. Para el acceso en programación, el SMBD extiende generalmente el lenguaje SQL incorporado con una sentencia que conecta el programa a una base de datos particular. La forma utilizada por SQL SERVER es:

```
USE DATABASEB 'PP'
```

Hablando de seguridad, cada usuario de una base de datos SQL tiene asignado un id-usuario, el cual es un nombre breve que lo identifica dentro del software SMBD siendo el núcleo de la seguridad SQL. Toda sentencia SQL ejecutada por el SMBD se lleva a cabo a cuenta de un id-usuario específico, que determina si la sentencia va a ser permitida o prohibida por el SMBD. En SQL SERVER, los id-usuarios pueden tener hasta 30 caracteres.

En una base de datos grande existen con frecuencia grupos de usuarios con necesidades similares para accesar a los datos. Para manejar estos grupos SQL SERVER soporta id-grupo, que indica grupos de id-usuario relacionados.

Los privilegios pueden ser concedidos tanto a id-usuarios individuales como a id-grupo, y un usuario puede efectuar una acción de base de datos si se le permite por su privilegio como id-usuario o id-grupo. Los id-grupos simplifican por tanto la administración de privilegios proporcionados a grupos de usuarios.

Las protecciones de seguridad SQL se aplican a objetos específicos contenidos en una base de datos. Así cada tabla y vista pueden ser individualmente protegidas. El acceso a una tabla o vista puede ser permitido por ciertos id-usuario y prohibido por otros id-usuario. Otro privilegio adicional de SQL SERVER sobre tablas o vistas es el SELECT extendido, que puede ser especificado por columnas individuales de una tabla o vista. Por tanto un usuario puede tener permitido recuperar ciertas columnas de una tabla mientras que otro usuario está restringido a otras columnas. También SQL SERVER soporta un privilegio para objetos de seguridad EXECUTE para procedimientos almacenados, que determina si un usuario tiene permitido ejecutar tal procedimiento.

La sentencia GRANT se utiliza para conceder privilegios de seguridad sobre objetos de la base de datos a usuarios específicos. Normalmente la sentencia GRANT es utilizada por el propietario de una tabla o vista para proporcionar a otros usuarios acceso a los datos por ejemplo:

Da privilegios totales sobre la tabla repventas a Pedro

```
GRANT ALL PRIVILEGES
   ON   repventas
   TO Pedro
```

Da a todos los usuarios acceso SELECT a la tabla oficinas

```
GRANT SELECT
   ON   oficinas
   TO   PUBLIC
```

SQL SERVER permite una lista de columnas para el privilegio SELECT. Por ejemplo, la siguiente sentencia es legal para SQL SERVER:

Proporciona a los usuarios de cobro de cuentas de acceso de sólo lectura a las columnas de número y nombre de empleado y oficina de ventas de la tabla repventas.

```
GRANT SELECT(num_empl,nombre,oficina_rep)
   ON   repventas
   TO   usuariocc
```

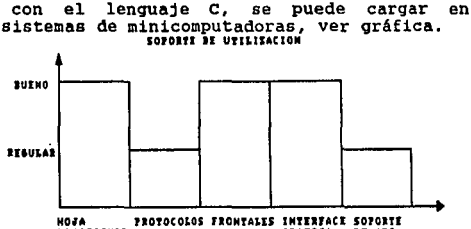
De la misma manera, los privilegios que se han concedido con GRANT pueden ser retirados con la sentencia REVOKE.

Para el problema de los datos inconsistentes o control de concurrencia SQL SERVER utiliza técnicas de candados sofisticadas para manejar transacciones concurrentes de muchos usuarios simultáneos como son los candados compartidos y exclusivos vistos en el capítulo 3.

4.2.1.2 ESTADISTICAS DE UTILIZACION

Las estadísticas de utilización para SQL SERVER son protocolos soportados, rendimiento en la velocidad de respuesta a requerimientos de varios usuarios, pruebas en la instalación Front-Ends soportados, uso de APIs entre otras.

El protocolo soportado es el NAMED PIPES 3/ de aquí que podemos decir que los sistemas operativos para red con los que puede trabajar SQL SERVER es LAN MANAGER, NETWARE y VINES. Las hojas electrónicas soportadas son LOTUS 123 V3.0, MS-EXCELL/SQL. Para desarrollar aplicaciones para la LAN de tipo complejas y amigables se dice que SQL SERVER soporta alrededor de 80 frontales en las estaciones de trabajo; algunos de ellos son los SMBD: DBASE IV, BORLAND, PARADOX/SQL, DATAEASE SQL, CLIPPER V5.0, ADVANCED REVELATION V2.0, XDB-SQL.SQL WINDOWS V1.2 como interfase gráfica de usuario, el propio Transact-SQL que es un lenguaje de cuarta generación creado por el SMBD SYBASE y el SDF que es una utilería para frontales de tipo amigable, que permite al administrador del sistema ejecutar tareas de administración y permitiendo a los usuarios hacer consultas (query), así mismo permitiendo editar, guardar, copiar, imprimir, etc, un query. El API de SQL SERVER al cual se le llama biblioteca de la base de datos o DBLIB consta de unas 100 funciones disponibles para un programa de aplicación. SQL SERVER y su API son también un excelente ejemplo de un SMBD diseñado sobre el terreno en torno a una arquitectura cliente/servidor. El API es la única interfase ofrecida por el SMBD que tiene un amplio soporte por parte de la empresa de MICROSOFT, ASTHON TATE, LOTUS y otros, trabajando básicamente con el lenguaje C, se puede cargar en una variedad de sistemas de minicomputadoras, ver gráfica.

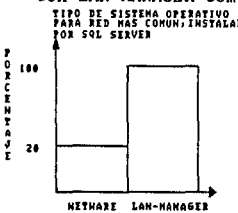


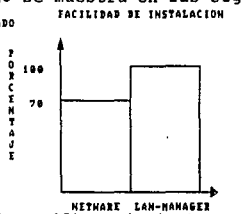
Si se trabaja SQL SERVER en una red con sistema operativo LAN MANAGER las estaciones utilizadas como servidores de base de datos pueden utilizar los sistemas operativos MS-DOS y OS/2.

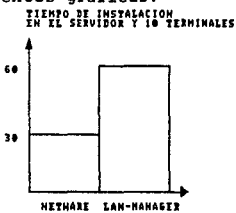
3/PROTOCOLO DE COMUNICACION PARA MANEJAR EL ESTABLECIMIENTO Y TERMINACION DE UNA SESION ENTRE DOS USUARIOS DE LA RED O ENTRE UN USUARIO Y EL FILE SERVER

Por otra parte si se utiliza el sistema operativo para red NETWARE (regularmente cuando se utiliza cualquier versión de este sistema operativo se habla de red local NOVELL, por el nombre de la empresa que lo creó), solo se puede soportar el sistema operativo OS/2 y para la instalación del SQL SERVER es necesario tanto en el servidor de datos como en las estaciones de trabajo cargar un producto llamado NOVELL REQUESTOR que sirva para conectar las estaciones OS/2 a la red, es decir iniciar una sesión con el file server, así mismo como hacer uso de utilerías que corren bajo el OS/2 y que carga en un nivel más alto el protocolo IPX/SPX 4/ al NAMED PIPES.

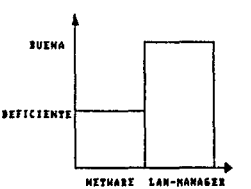
Finalmente NOVELL REQUESTOR permite a las redes NOVELL imitar el esquema que utiliza LAN MANAGER. Pero la facilidad para instalar indudablemente es más fácil y común en una red con LAN MANAGER como se muestra en las siguientes gráficas:







Por otra parte la gráfica siguiente muestra que todo tipo de documentación para SQL SERVER en LAN MANAGER es mucho mayor que para NETWARE.



Un boletín informativo muestra otros tipos de estadísticas de utilización al manejar SQL SERVER en una red de área local con un servidor de base de datos de los más poderosos por su avanzada tecnología llamado SUN SPARC SERVER 490, siendo diseñado para maximizar el funcionamiento de la red

---

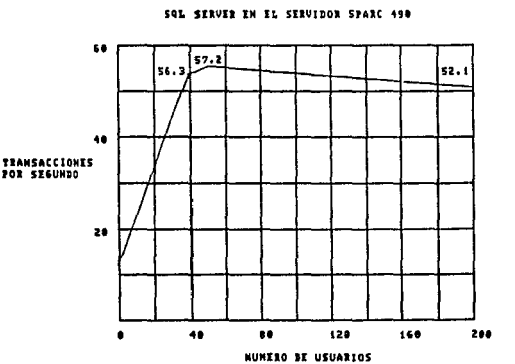
4/PROTOCOLO DE COMUNICACION RAPIDO Y EFICIENTE PARA LAN

mientras soporta un gran número de usuarios y estaciones de trabajo SUN 5/.

Después de ejecutar pruebas se llegó al resultado de que SQL SERVER en un SPARC SERVER 490 tuvo un alcance de 57 transacciones por segundo, y teniendo la capacidad para soportar 200 usuarios con 52 transacciones por segundo cada uno.

Otra demostración de la fuerza de SQL SERVER, en un solo servidor SPARC 490, fue que soportó 1000 usuarios simultáneos junto con sus consultas (querys) y sus requerimientos de candados; demostrando estos bancos de prueba que SQL SERVER tiene la capacidad para accesos rápidos y probando que este SMBD provee altos niveles de alcance para un gran número de usuarios con tiempo de respuesta rápida en grandes empresas.

La gráfica muestra que SQL SERVER puede escalar un gran número de usuarios sin comprometer sus alcances.



Por otra parte las siguientes pruebas fueron efectuadas de una red local que cuenta con 5 estaciones de trabajo, un servidor de archivos y uno de base de datos, con procesadores 80486 y disco duro de 600MB y 80mb respectivamente, ambos trabajando a 16Mhz, las estaciones de trabajo son PC AT-compatibles de 12Mhz, sin disco duro. El sistema operativo de red fue LAN MANAGER 2.0 y probando con una base de datos de 36818 registros.

5/SUN DE SUS SIGLAS STANDFORD UNIVERSITY NETWORKS, SE DICE DE ESTACIONES Y SERVIDORES INTELIGENTES, CON MAYOR VELOCIDAD Y CAPACIDAD.

Las pruebas fueron efectuadas en sincronización, es decir todas las estaciones de trabajo empiezan a la vez. Cada prueba fue corrida 5 veces, descartando la primera vez y promediando las cuatro restantes. Todos los programas fueron requeridos para usar la base de datos de modo compartido.

Aunque se probaron 5 versiones de cada prueba, agregamos una estación de trabajo cada vez, y las gráficas reportan los resultados de una, tres y cinco estaciones de trabajo. Los porcentajes con dos y cuatro estaciones siguen la curva demostrada en estos resultados.

Como se ha dicho todas las pruebas se efectuaron sobre una base de datos, pero cada estación de trabajo buscó ó editó diferentes registros. Así la primera estación buscó ó editó el registro 22222, la segunda el 22223, y así sucesivamente.

Las siguientes pruebas fueron:

1. Prueba de creación y búsqueda de índices (satisfactoria).

   El programa busca y encuentra un registro específico mientras el índice está activo, sólo para la prueba de una estación es donde se crea y se busca el índice.

2. Prueba de búsqueda de índices (insatisfactoria).

   El programa busca un registro específico mientras el índice está activo. El registro no existe y es forzado a buscarlo en toda la base de datos.

3. Prueba de búsqueda secuencial (satisfactoria).

   El programa busca y encuentra un registro específico mientras el índice no está activo.

4. Prueba de búsqueda secuencial (insatisfactoria).

   El programa busca un registro mientras el índice no está activo. El registro no existe, y el programa es forzado a buscarlo en toda la base de datos.

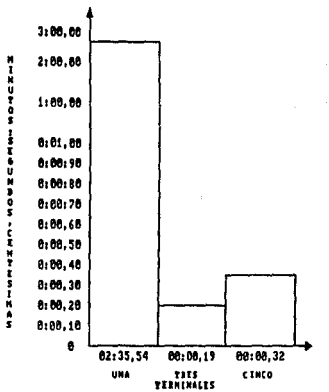
5. Prueba buscando y agregando registros indexados.

   El programa busca un registro específico cuando el índice está activo, como no lo encuentra, el programa entonces lo agrega a la base de datos indexada.

Los resultados son arrojados en las siguientes gráficas

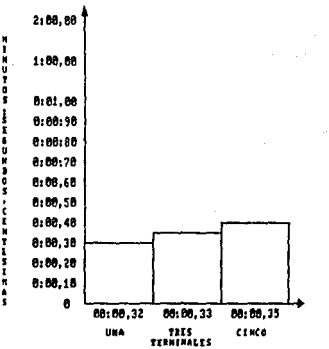
PRUEBA 1

BUSQUEDA POR INDICE SATISFACTORIA



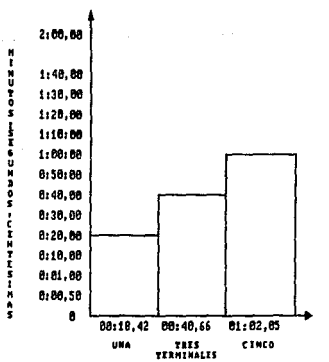
PRUEBA 2

BUSQUEDA SECUENCIAL INSATISFACTORIA



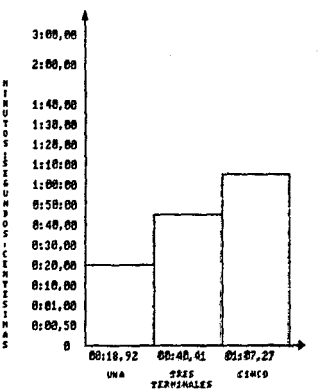
PRUEBA 3

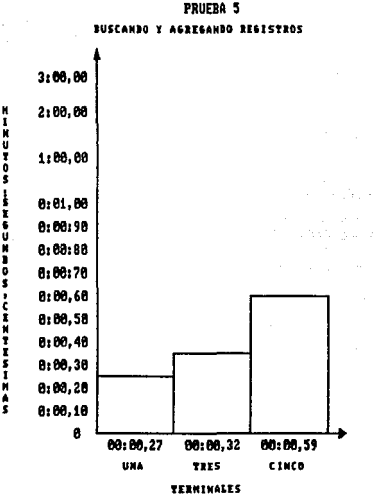
BUSQUEDA SECUENCIAL SATISFACTORIA



PRUEBA 4

BUSQUEDA SECUENCIAL SATISFACTORIA





En el capítulo 6 se efectuará un estudio comparativo de estos resultados con los del siguiente capítulo.

4.2.1.3 VENTAJAS.

En este punto se analizarán las que a consideración del estudio son las 2 ventajas mayores de SQL SERVER: El catálogo del sistema y el API de SQL SERVER.

Una de las ventajas de SQL SERVER es el catálogo de sistema , el cual hace posible tener herramientas de consultas de fácil actualización (<<amistosas>>), es decir permitir a los usuarios un acceso simple y transparente de la base de datos, sin tener que aprender el lenguaje SQL, conteniendo información referente a tablas, columnas, usuarios, vistas y privilegios. La tabla 1 muestra los nombres de las tablas de sistema que proporcionan esta información en SQL SERVER:

TABLAS	COLUMNAS	USUARIOS	VISTAS	PRIVILEGIOS
SYSOBJECTS SYSKEYS	SYSCOLUMNS	SYSUSERS SYSLOGINS SYSALTERNATES	SYSDEPENDS SYSCOMMENTS	SYSPROTECTS

TABLA 1. NOMBRE DE TABLAS DE SISTEMA DE SQL SERVER

SQL SERVER tiene una tabla de sistema que lleva la cuenta de las tablas existentes en la base de datos, llamada SYSOBJECTS, descrita en la tabla 2

NOMBRE DE COLUMNA	TIPOS DE DATOS	INFORMACION
NAME	SYSNAME	NOMBRE DEL OBJETO
ID	INT	NUMERO ID INTERNO DEL OBJETO
UID	SMALLINT	ID-USUARIO DEL PROPIETARIO DEL OBJETO
TYPE	CHAR(2)	CODIGO DE TIPO DE OBJETO*
RESDATE	DATETIME	RESERVADO PARA USO FUTURO
CRDATE	DATETIME	FECHA/HORA EN QUE EL OBJETO FUE CREADO
EXPDATE	DATETIME	RESERVADO PARA USO FUTURO
DELTRIG	INT	ID DE PROCEDIMIENTO DE DISPARADOR DELETE
INSTRIG	INT	ID DE PROCEDIMIENTO DE DISPARADOR INSERT
UPDTRIG	INT	ID DE PROCEDIMIENTO DE DISPARADOR UPDATE
SELTRIG	INT	RESERVADO PARA USO FUTURO

* S=TABLA DE SISTEMA, U=TABLA DE USUARIO, V=VISTA, L= REGISTRO(LOG), P=PROCEDIMIENTO ALMACENADO, R=REGLA, D=OMISION, TR=DISPARADOR

TABLA 2. COLUMNAS SELECCIONADAS DE LA TABLA SYSOBJECTS -SQL SERVER-

La tabla SYSOBJECTS almacena información acerca de las tablas y vistas, y de otros objetos tales como procedimientos almacenados, reglas y disparadores. Se puede observar como la tabla SYSOBJECTS utiliza un número de identificación interno en lugar de un nombre para identificar al propietario de la tabla. SQL SERVER también tiene una tabla de sistema que lleva la cuenta de las columnas de la base de datos llamada SYSCOLUMNS Hay un renglón en la tabla SYSCOLUMNS por cada columna de cada tabla o vista de la base de datos. La tabla 3 muestra la definición de SYSCOLUMNS:

NOMBRE DE COLUMNA	TIPOS DE DATOS	INFORMACION
ID	INT	ID INTERNO DE LA TABLA QUE CONTIENE LA COLUMNA
NUMBER	SMALLINT	CERO PARA DEFINICIONES DE COLUMNA
COLID	TINYINT	NUMERO DE ID DE COLUMNA INTERNO
STATUS	TINYINT	SE PERMITE NULL
TYPE	TINYINT	CODIGO DE TIPO DE DATOS PARA LA COLUMNA
LENGTH	TINYINT	LONGITUD DE LA COLUMNA
OFFSET	TINYINT	DESPLAZAMIENTO DE COLUMNA FISICO DESDE EL COMIENZO DE LA FILA
USERTYPE	SMALLINT	CODIGO DE TIPO DE DATOS PARA TIPOS DE DATOS DEFINIDOS POR USUARIO
COLFAULT	INT	ID DE PROCEDIMIENTO ALMACENADO PARA VALOR POR OMISION
DOMAIN	INT	ID DE PROCEDIMIENTO ALMACENADO PARA REGLA DE CELUMNA
NAME	INT	NOMBRE DE COLUMNA
PRINTFMT	VARCHAR(255)	FORMATO DE VISUALIZACION PARA LA COLUMNA

TABLA 3. TABLA SYSCOLUMNS -SQL SERVER-

El catálogo de sistema contiene generalmente una tabla que identifica a los usuarios que tienen autorizado el acceso a la base de datos. SQL SERVER puede utilizar esta tabla de sistema para validar el nombre de un usuario y su contraseña cuando un usuario intenta conectarse por primera vez a la base de datos. La tabla puede almacenar también otros datos referentes al usuario. SQL SERVER almacena información de usuario en la tabla de sistema SYSUSERS, mostrada en la tabla 4. Cada renglón de esta tabla describe un solo usuario o un grupo de usuarios en el esquema de seguridad.

NOMBRE DE COLUMNA	TIPOS DE DATOS	INFORMACION
SUID	SMALLINT	NUMERO ID-USUARIO SERVIDOR INTERNO
UID	SMALLINT	NUMERO ID-USUARIO INTERNO EN ESTA BASE DE DATOS
GID	SMALLINT	NOMBRE-ID-GRUPO DE USUARIO INTERNO EN LA BASE DE DATOS
NAME	SYSNAME	NOMBRE DE USUARIO O GRUPO

TABLA 4. COLUMNAS SELECCIONADAS DE LA TABLA SYSUSERS -SQL SERVER-

He aquí dos consultas equivalentes que listan los usuarios autorizados para SQL SERVER:

Lista todos los id-usuario conocidos a SQL SERVER

```
SELECT name
  FROM SYSUSERS
 WHERE uid <> gid
```

También el catálogo almacena información necesaria para mantener la seguridad de la base de datos. Este esquema de autorización utilizado por SQL SERVER, trata a las tablas, los procedimientos almacenados, los disparadores y otras entidades uniformemente como objetos a los cuales se aplican los privilegios. Esta estructura se refleja en la tabla de sistema SYSPROTECTS, mostrada en la tabla 5, que implanta el esquema de privilegios completo para SQL SERVER. cada renglón de la tabla representa una única sentencia GRANT o REVOKE que ha sido emitida.

Finalmente SQL SERVER soporta múltiples bases de datos designadas. Tiene una tabla de sistema llamada SYSDATABASES que identifica las bases de datos administradas por un mismo servidor.

NOMBRE DE COLUMNA	TIPOS DE DATOS	INFORMACION
ID	INT	ID INTERNO DEL OBJETO PROTEGIDO
UID	SMALLINT	ID INTERNO DEL USUARIO O GRUPO CON PRIVILEGIO
ACTION	TINYINT	CODIGO NUMERICO DE PRIVILEGIO (POR EJEMPLO, SELECT= 193),
PROTECTTYPE	TINYINT	CODIGO NUMERICO PARA CONCESION O REVOCACION
COLUMNS	VARBINARY(32)	MAPA DE BITS PARA PRIVILEGIOS DE ACTUALIZACION A NIVEL DE COLUMNA

TABLA 5. SYSPROTECTS -SQL SERVER-

En el capítulo 3 se explicó el API de SQL, SQL SERVER es uno de los SMBD que contiene un API de mayor importancia, teniendo un amplio soporte por parte de diferentes empresas como Microsoft y otros. EL API de SQL SERVER ha sido diseñado sobre la arquitectura cliente/servidor, y es llamado también biblioteca de la base de datos o DBLIB, consta de 100 funciones para un programa de aplicación, algunas de ellas son:

dbopen() Abre una conexión al SQL SERVER
dbcmd() Transfiere el texto de sentencia SQL a dblib
dbcolname() Obtiene el nombre de una columna de resultados

Un programa simple de SQL SERVER que actualiza una base de datos puede utilizar un conjunto muy pequeño de llamadas DBLIB para efectuar este trabajo. El programa de la figura 5 muestra una aplicación de actualización de cuotas para la tabla repventas.

1. El programa prepara un registro de presentación, relleno con el nombre de usuario, su contraseña y la información necesaria para conectarse al SQL SERVER.

2. dbopen() establece conexión con SQL SERVER.

3. El programa construye una sentencia SQL en un buffer e invoca a dbcmd() para transferir el texto SQL a DBLIB.

4. El programa invoca a dbsqlexec(), instruyendo a SQL SERVER para que ejecute la sentencia previamente transferida con dbcmd().

5. El programa llama a dbresults() para determinar el éxito o fallo de la sentencia.

6. El programa invoca a dbexit() para dar por concluida la conexión a SQL SERVER.

Este programa de SQL SERVER está totalmente en lenguaje C, aceptable por el compilador, y no requiere técnicas de codificación especiales. Este programa envía una única sentencia SQL a SQL SERVER y comprueba su estado.

Para manejar las consultas programadas en SQL SERVER, un programa envía una sentencia SELECT a SQL SERVER y utiliza DBLIB para recuperar los resultados renglón a renglón.

```
MAIN()
{
 LOGINREC  *REGCONEX;        /* ESTRUCTURA DE DATOS PARA INFORMACION */
 DBPROCESS *PROCDB;          /* ESTRUCTURA DE DATOS PARA CONEXION */
 CHAR      CAD_IMPORTE[31]; /* IMPORTE INTRODUCIDO POR EL USUARIO,(COMO CADENA)*/
 INT       ESTADO;           /* ESTADO DEVUELTO POR LA LLAMADA DBLIB */

/* OBTIENE ESTRUCTURA DE PRESENTACION Y DEFINE NOMBRE Y CONTRASENA USUARIO */
 REGCONEX = DBLOGIN();
 DBSETLUSER(REGCONEX,"SCOTT");  <-- 1
 DBSETLPWD(REGCONEX,"TIGRE");

 /*CONECTA CON SQL SERVER*/
 PROCDB =  DBOPEN(REGCONEX," ");  <-- 2

 /* PIDE AL USUARIO QUE ESCRIBA LA CANTIDAD DE AUMENTO/DISMINUCION DE CUOTAS */
 PRINTF("ELEVAR/BEDAJAR LAS CUOTAS EN CUANTO!");
 GETS(CAD_IMPORTE);

 /* TRANSFIERE SENTENCIA SQL A DBLIB */
 DBCMD(PROCDB,"UPDATE REVISTAS SET CUOTA = CUOTA + ");  <-- 3
 DBCMD(PROCDB, CAD_IMPORTE);

 /* MANDA A SQL SERVER EJECUTAR LA SENTENCIA */
 DBSQLEXEC(PROCDB);  <-- 4

 /* OBTIENE RESULTADOS DE LA EJECUCION DE LA SENTENCIA */  <-- 5
 ESTADO = DBRESULTS(PROCDB);
 IF (ESTADO != SUCCEED)
    PRINTF("ERROR DURANTE ACTUALIZACION.\N");
 ELSE
    PRINTF("ACTUALIZACION CON EXITO.\N");

 /* SUSPENDE LA CONEXION CON SQL SERVER */
 DBEXIT(PROCDB);  <-- 6
 EXIT(0);
}
```

FIGURA 5. PROGRAMA SENCILLO SQL SERVER

Una de las características más importantes de SQL SERVER es su soporte de procedimientos almacenados. Un procedimiento almacenado es una secuencia de sentencias de Transact-SQL a las que se les asigna un nombre, se compila y se almacena en una base de datos SQL SERVER. Ya que este procedimiento fue almacenado en la base, un programa de aplicación puede llamarlo por su nombre, utilizando DBLIB.

La figura 6 es una definición de procedimiento, escrita en el dialecto Transact-SQL de SERVER. El procedimiento, denominado obt_infoclie(), acepta un nombre de cliente y ejecuta dos consultas. El argumento del procedimiento, num_clie, es una variable del lenguaje Transact-SQL, que puede ser utilizada dentro del procedimiento almacenado en otras sentencias Transact-SQL. En este ejemplo, la variable se utiliza dentro de las cláusulas WHERE de las dos sentencias SELECT. Cuando SQL SERVER ejecuta la sentencia CREATE PROCEDURE, compila el procedimiento y lo almacena en la base de datos.

```
CREATE PROCEDURE OBT_INFOCLIE (@ NUM_CLIE INTEGER ) AS
/* RECUPERA EL NOMBRE DE LA EMPRESA DEL CLIENTE Y SU LIMITE DE CREDITO */
 SELECT EMPRESA,LIMITE_CREDITO FROM CLIENTES
 WHERE NUM_CLIE = @NUM_CLIE
/* RECUPERA INFORMACION REFERENTE A CADA PEDIDO ORDENADO POR EL CLIENTE */
 SELECT NUM_PEDIDO, IMPORTE
   FROM PEDIDOS
  WHERE CLI = @NUM_CLIE
```

FIGURA 6. UN PROCEDIMIENTO ALMACENADO DE SQL SERVER

La figura 7 muestra un extracto de una aplicación de búsqueda sobre clientes. Pide al usuario que escriba un número de cliente y luego invoca el procedimiento almacenado en la figura 6 para recuperar el nombre y el límite de crédito del cliente desde la tabla clientes y para recuperar el nombre y el límite de crédito del cliente desde la tabla clientes y para recuperar todos los pedidos actuales remitidos por ese cliente desde la tabla pedidos.

Observe que el programa llama dos veces a dbrsults() para procesar los resultados de las dos consultas en el procedimiento almacenado.

En la figura 6 tan sólo esboza la capacidad de los procedimientos almacenados de SQL SERVER. EL lenguaje Transact-SQL proporciona un conjunto completo de extensiones procedurales a SQL, incluyendo condiciones (IF/THEN/ELSE), bloques de sentencias (BEGIN/END) entre otros. Creando esto procedimientos almacenados más complejos.

```
MAIN()
{
 LOGINREC   *REGCONEX;          /* ESTRUCTURA DE DATOS PARA INFORMACION */
 DBPROCESS  *PROCDB;            /* ESTRUCTURA DE DATOS PARA CONEXION */
 CHAR       NUMERO[11];         /* NUMERO DE CLIENTE INTRODUCIDO POR EL USUARIO */
 CHAR       EMPRESA[31];        /* NOMBRE DE EMPRESA, RECUPERADA */
 FLOAT      LIMITE;             /* LIMITE DE CREDITO, RECUPERADO */
 LONG       NUM_PEDID;          /* NUMERO PEDIDO, RECUPERADO */
 FLOAT      IMPORTE;            /* IMPORTE DE PEDIDO, RECUPERADO */
    .
    .
    .
/*PIDE AL USUARIO QUE ESCRIBA UN NUMERO DE CLIENTE */
PRINTF(" INTRODUZCA UN NUMERO DE CLIENTE ");
GETS(NUMERO);

/* MANDA A SQL SERVER EJECUTAR EL PROCEDIMIENTO ALMACENADO */
DBCMD(PROCDB,"EXECUTE OBT_INFOCLIE ");
DBCMD(PROCDB, NUMERO);

/* MANDA A SQL SERVER EJECUTAR LA SENTENCIA */
DBSQLEXEC(PROCDB);

/* RECUPERA RESULTADOS DE LA PRIMERA CONSULTA Y LOS VISUALIZA */
DBRESULT(PROCDB);
DBBIND(PROCDB,1,NTBSTRINGBIND,31,&EMPRESA)
DBBIND(PROCDB,2,FLT4BIND,0,&CREDITO);
ESTADO = DBNEXTROW(PROCDB);
IF (ESTADO != SUCCEED) { PRINTF("NO EXISTE TAL CLIENTE.\n");
EXIT(1);}
PRINTF("CLIENTE:%S LIMITE CREDITO:%F\n",EMPRESA,LIMITE);

/*PROCESA LA SEGUNDA CONSULTA, IMPRIMIENDO INFORMACION DE PEDIDO */
DBRESULTS(PROCDB);
DBBIND(PROCDB,1,INTBIND,0,&NUM_PEDIDO);
DBBIND(PROCDB,2,FLT4BIND,0,&IMPORTE);
WHILE (ESTADO = DBNEXTROW(PROCDB)== SUCCEED) {
   /* IMPRIME DATOS PARA ESTE NUMERODE */
   PRINTF("NUMERO DE PEDIDO %ld IMPORTE:%f\n",NUM_PEDIDO,IMPORTE);}
}
```

FIGURA 7. UTILIZACION DE UN PROCEDIMIENTO ALMACENADO EN SQL SERVER

Los procedimientos almacenados también juegan un papel importante en el rendimiento de red de la arquitectura cliente/servidor.

En esta arquitectura, el programa de aplicación y el API de SQL residen en un sistema cliente (típicamente una PC) mientras que SQL SERVER reside en un sistema servidor diferente, conectado a los sistemas clientes mediante una red de área local. Para obtener buenos resultados de esta arquitectura , el tráfico sobre la red debe ser minimizado, y los procedimientos almacenados ayudan a lograr este objetivo.

La figura 8 muestra una implementación cliente/servidor utilizando el API de SQL SERVER y un procedimiento almacenado, efectuando el programa una actualización y una consulta que genera una docena de renglones de resultados. En esta figura una única sentencia SQL que invoca el procedimiento almacenado se envía a través de la red desde el cliente al servidor. SQL SERVER ejecuta el procedimiento almacenado, generando un flujo de mensajes de estado y de resultados de consulta que se devuelven a través de la red al sistema cliente. En este caso los resultados son relativamente pequeños y el SQL SERVER los envía de vuelta al sistema cliente, en donde DBLIB los almacena en buffers. Las llamadas dbnextrow() del programa DBLIB son entonces satisfechas a partir de los datos contenidos en los buffers. Es decir el esquema SQL SERVER genera una única ronda a través de la red.

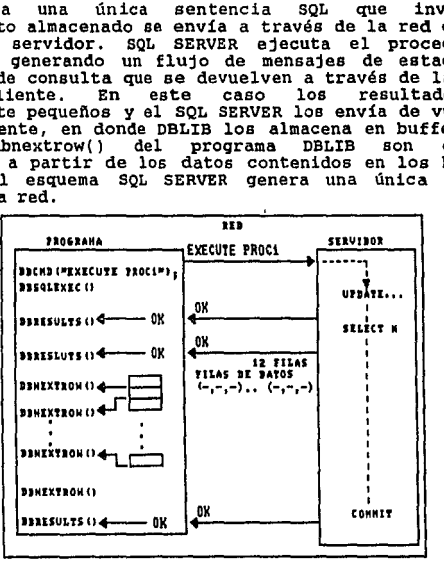


FIGURA 8. TRAFICO DE RED PARA UNA TRANSACCION EN SQL SERVER

En SQL SERVER, cada procedimiento almacenado se identifica mediante un nombre de procedimiento. Además, el procedimiento compilado puede constar de una secuencia completa de sentencias SQL, minimizando el número de solicitudes que debe hacer el programa SMBD.

Los procedimientos almacenados de SQL SERVER proporcionan una ventaja adicional. Pueden incluir sentencias de Transact-SQL que interactúen con otro software en el servidor aparte del propio SMBD. Por ejemplo, en procedimiento almacenado puede enviar un mensaje por correo a otro usuario o desencadenar la ejecución de un programa escrito por el usuario sobre el sistema servidor. Con esta capacidad, SQL SERVER puede ir más allá del papel de un servidor de base de datos y tomar un conjunto más amplio de responsabilidades como servidor de aplicaciones.

4.2.1.4 LIMITACIONES

Como se sabe este SMBD trabaja en la arquitectura cliente/servidor corriendo sólo en el sistema operativo OS/2, solamente soporta los tres sistemas operativos para red mencionados en estadísticas de utilización, requiere para la estación o estaciones que fungirán como servidores de base de datos una PC tipo AT (advanced tecnology) compatible con IBM, como mínimo un procesador 80286, el sistema operativo OS/2 versión 1.1 o mayor, un requerimiento mínimo de memoria RAM o sea que la memoria debe soportar tanto a OS/2 como a SQL SERVER, 30 MB (Megabytes) en disco duro o mayor de preferencia. Para las estaciones de trabajo cualquier PC compatible con IBM que contenga como mínimo, 640 KB (kilobytes) de memoria RAM y lo que ocupe la versión del sistema operativo (OS/2) manejado. Provee un buen control avanzado de impresoras, obviamente esta basado en el estándar SQL, y soporta API e interfaz de usuario gráfico como se vió en estadísticas de utilización, y gran capacidad para acceso múltiple y simultáneo, pero sólo permite tener como frontal a las aplicaciones hechas en lenguaje C.

SQL SERVER tiene un numero ilimitado para abiertas tablas, el número máximo de indexaciones por tabla es ilimitado, lo mismo que el número de renglones por tabla y el número de campos por renglón.

Hablando de SQL SERVER como SMBD hay ocasiones que es conveniente combinar los resultados de dos o más consultas en una única tabla de resultados totales. SQL soporta esta capacidad gracias a la característica UNION de la sentencia SELECT por ejemplo:

```
SELECT   fábrica,producto
  FROM   productos
 WHERE   precio > 2000.00
 UNION
SELECT   id_fabri,id_produ
  FROM   pedidos
 WHERE   precio > 3000.00
```

Aquí UNION produce una única tabla de resultados que combina los renglones de la primera consulta con los renglones de los resultados de la segunda consulta, pero esta característica SQL SERVER no la soporta.

El procesamiento de consultas SQL SERVER utiliza una conexión separada entre el programa y el SMBD. En respuesta a un lote de sentencias que contiene una o más sentencias SELECT, SQL SERVER envía los resultados de la consulta de vuelta al software DBLIB, el cual los maneja. La recuperación renglón a renglón es administrada por las llamadas API de DBLIB, no por las sentencias del lenguaje SQL.

Como resultado, SQL SERVER no puede soportar actualizaciones posicionadas, ya que el propio SMBD no tiene noción de un renglón <<actual>>. El dialecto Transact-SQL solamente soporta las sentencias UPDATE y DELETE con búsqueda.

La falta de actualización posicionada puede ser una limitación real en aplicaciones que permiten al usuario inspeccionar un conjunto de resultados y actualizar ocasionalmente el registro actualmente a la vista. Para minimizar esta limitación, el API DBLIB incluye un conjunto especial de funciones en modo inspección, resumidas en la tabla 3. Utilizando estas funciones, un programa puede crear su propia capacidad de actualización posicionada.

FUNCION	DESCRIPCION
DBTABCOUNT()	INFORMA DEL NUMERO DE TABLAS FUENTES PARA LA CONSULTA ACTUAL
DBTABNAME()	INFORMA DEL NOMBRE DE UNA DE LAS TABLAS FUENTE
DBTABBROWSE()	INFORMA DE SI UNA TABLA PARTICULAR PUEDE SER ACTUALIZADA
DBCOLBROWSE()	INFORMA SI UNA COLUMNA PARTICULAR PUEDE SER ACTUALIZADA
DBTABSOURCE()	INFORMA DEL NOMBRE DE LA TABLA ESPECIFICADA DE RESULTADOS
DBCOLSOURCE()	INFORMA DEL NOMBRE DE LA COLUMNA ESPECIFICADA DE RESULTADOS
DBQUAL()	PRODUCE UNA CLAUSULA WHERE QUE SELECCIONARA LA FILA DE RESULTADOS ACTUALMENTE EN PROCESO

TABLA 3. FUNCIONES DBLIB DE MODO INSPECCION (BROWSE)

5. EL MANEJADOR DE BASE DE DATOS FORCE Y LA RED LOCAL.

Hace sólo unos años los usuarios de red se sentían frustrados por la falta de software de aplicación. Después del esfuerzo de seleccionar una red y ponerla a funcionar en forma óptima, la adaptación de aplicaciones monousuario es antidecisiva, por decir lo menos. Eso ha cambiado para bien, en la mayoría de los SMBD existentes, ya se permite la posibilidad de que múltiples usuarios tengan acceso a datos comunes en una red de área local siendo ésto, un beneficio enorme para la productividad. Es decir ofrecer un SMBD nuevo en una red para ofrecer acceso compartido a datos y lenguajes de programación que se puedan utilizar para desarrollar aplicaciones adecuadas a las necesidades de cada usuario es una gran ventaja. Además, cuando múltiples usuarios de una empresa comparten un conjunto de datos en común, la tarea de actualizar y respaldar información se vuelve más sencilla y más confiable. Es por esto, que resulta alentador ver el renovado interés que ponen los fabricantes de base de datos en la funcionalidad de las redes.

Con la mayoría de los SMBD se puede colocar de inmediato una base de datos enlazada a una red en un entorno multiusuario uno de estos SMBD es FORCE de la empresa SOPHCO INC, saliendo su última versión a mediados de 1990. FORCE es un compilador para lenguajes conocidos como XBASE 1/ con un gran número de adiciones que lo convierten en un lenguaje de proposito general.

El compilador FORCE produce un archivo objeto nativo en la misma manera que los compiladores del lenguaje C que conforman el estándar de microsoft C. Es decir un archivo objeto nativo significa que puede ser incluído en otro compilador lo que se llama un valor agregado. Este diseño permite enlazar módulos de FORCE con lenguaje C lo más sencillo posible.

FORCE es un lenguaje sumamente estructurado como C o Pascal de propósito general, pero con énfasis específica en eficientes programas profesionales de manejo de base de datos. La sintaxis que utiliza es compatible con el más popular SMBD DBASE y combinando sus más eficientes características éste SMBD, con un desarrollo estructurado que conteniene un poderoso diseño de base de datos, FORCE es un digno SMBD a estudiar.

FORCE convierte los archivos fuente en pequeños y rápidos programas ejecutables (.exe). El precio de FORCE es de aproximadamente 600 dólares y para su uso en una red de área local solamente se instala en la computadora que fungirá como servidor de archivos.

1/MARCO DE DIFERENTES SMBD CON COMANDOS Y APLICACIONES DE DBASE

5.1 SISTEMA FORCE COMO MANEJADOR DE BASE DE DATOS EN UNA RED LOCAL.

Como se sabe la introducción de la computadora personal y de las redes de área local condujo al desarrollo de la arquitectura servidora de archivos, mostrada en la figura 1. En esta arquitectura una aplicación que corre en una computadora personal puede acceder en forma transparente a datos localizados en un servidor de archivos, que almacena los archivos compartidos. Cuando una aplicación en PC solicita los datos de un archivo compartido, el software de red (en el caso de FORCE es cualquier versión del sistema operativo NETWARE) recupera automáticamente el bloque solicitado del archivo en el servidor. FORCE soporta esta estrategia servidora de archivos, donde cada computadora personal puede ejecutar su propia copia del software SMBD FORCE.

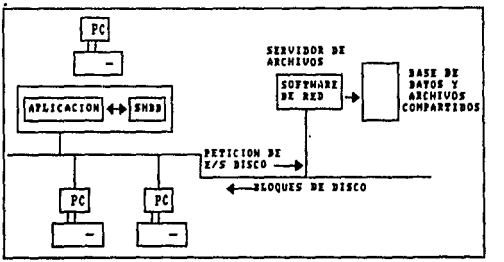


FIGURA 1. EL SMBD FORCE EN UNA ARQUITECTURA CLIENTE/SERVIDOR

Para continuar este punto se harán notar algunas características escenciales para conocer al FORCE.

Un programa en FORCE es altamente estructurado, ya que éste es un compilador que requiere que las funciones y procedimientos sean definidos antes de cualquier línea del programa que puede usarlos. El programa principal es siempre un procedimiento llamado force_main y obvio es el último procedimento en el programa.

La figura 3 muestra el ejemplo de un programa común en FORCE.

1. Aquí se definen los archivos de encabezado, los cuales tienen extensión (.hdr) por medio de la directiva #INCLUDE, la cual indica al compilador que incluya archivos que contengan cualquier código desde una directiva simple hasta un programa completo. Los archivos incluídos son almacenados para ser usados frecuentemente tal como funciones o procedimientos.

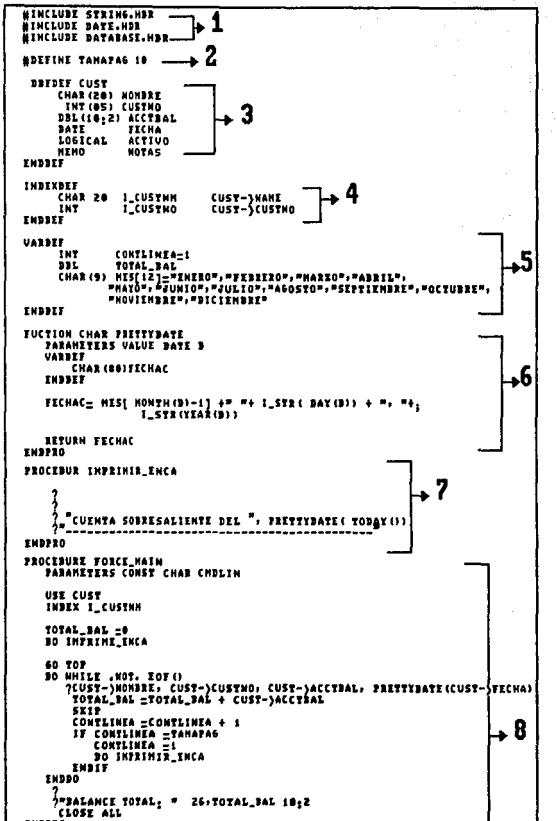


FIGURA 3. PROGRAMA EN FORCE

Para usar una función o procedimiento de la librería en un programa hay que incluir el archivo de encabezado apropiado usando la directiva #INCLUDE, en este ejemplo string.hdr, date.hdr y database.hdr son compilados primero. La función month() es declarada en el archivo date.hdr y en la linea #INCLUDE date.hdr indicamos que esta función puede ser usada. La tabla muestra los archivos de encabezado que maneja FORCE.

COLOR.HDR	ESTE ARCHIVO CONTIENE DEFINICION DE COLORES
DATA.HDR	DEFINE SIMBOLOS GLOBALES PARA SER USADOS EN EL CONTROL DEL DESARROLLO DE FORCE A UN BAJO NIVEL
DATABASE.HDR	CONTIENE FUNCIONES PARA MANEJO DE BASE DE DATOS
DATE.HDR	CONTIENE FUNCIONES PARA MANEJO DE FECHAS
ERROR.HDR	CONTIENE FUNCIONES PARA MANEJO DE ERRORES
FILEIO.HDR	CONTIENE FUNCIONES DE BAJO NIVEL PARA MANEJO DE ARCHIVOS
IO.HDR	CONTIENE FUNCIONES QUE REGRESAN INFORMACION SOBRE EL MANEJO DEL CURSOR EN LA PANTALLA
KEYS.HDR	CONTIENE DEFINICION DE MACROS PARA CADA TECLA O EL TECLADO.
MATH.HDR	CONTIENE FUNCIONES MATEMATICAS
MEMO.HDR	CONTIENE FUNCIONES PARA MANEJO DE CAMPOS TIPO MEMO
MIXED.HDR	CONTIENE FUNCIONES PARA ENLAZAR UN PROGRAMA CON OBJETOS GENERADOS EN LENGUAJE C
PICX.HDR	CONTIENE FUNCIONES PARA MANEJAR Y MANIPULAR LISTAS PICX
STRING.HDR	CONTIENE FUNCIONES PARA MANIPULACION DE CADENAS
SYSTEM.HDR	CONTIENE FUNCIONES QUE REGRESAN INFORMACION HACERCA DEL HARDWARE Y EL SISTEMA OPERATIVO
TSR.HDR	CONTIENE FUNCIONES PARA MANEJO DE PROGRAMAS RESIDENTES EN MEMORIA

2. Cuando FORCE alcanza la linea #DEFINE TAMAPAG 10, toma el nombre de la macro TAMAPAG y su valor representado por la cadena 10. Cuando FORCE encuentra la línea IF contlin > &TAMAPAG, remplaza la cadena &TAMAPAG con la cadena 10, resultando IF contlin > 10, entonces pasa a la siguiente línea. Una vez definida no se puede cambiar.

Las macros de FORCE se utilizan cuando comparamos el valor regresado de una función, por ejemplo, inkey(). El codigo IF k = &k_ESC es más entendible que IF k = 27.

3. Las líneas seguidas de la palabra clave DBFDEF y hasta ENDDEF describen los campos en la base de datos. Para declarar un campo se necesita su tipo de dato, tamaño y nombre, en este ejemplo se crea un alias 2/ cust que es con el que se hace referencia al querer utilizar la base de datos a lo largo del programa

4. FORCE requiere que todos los índices se encuentren dentro del bloque INDEXDEF y ENDDEF. Cada línea entre estas palabras claves especifica un índice para el alias. Un índice para el alias especifica tres cosas: El tipo de expresión (CHAR, DATE, etc, menos FILE, MEMO Y LOGICAL), el nombre del índice y el alias con el nombre de campo que va hacer indexado. En el punto 4 de la figura 3, el primer índice es CHAR(20) I_CUSTNM CUST->NOMBRE.

5. Las variables son declaradas antes de ser usadas. El bloque VARDEF contiene la definición de variables hasta encontrarse con ENDDEF. Cada línea en un bloque VARDEF declara una o más variables de un tipo dado, en este caso declara una variable entera (INT), una de coma flotante (DBL) y un arreglo de tipo caracter (CHAR), El MES[12] le dice al compilador que hay 12 elementos en el arreglo con 9 caracteres cada uno, siendo en este caso los elementos del arreglo constantes.

6. Las funciones deben ser declaradas antes de ser llamadas. En el bloque FUNCTION y ENDPRO son declaradas tanto parámetros como variables y el conjunto de instrucciones que se necesitan en la función. Las variables definidas dentro de ésta son locales, es decir existen sólo en la función.

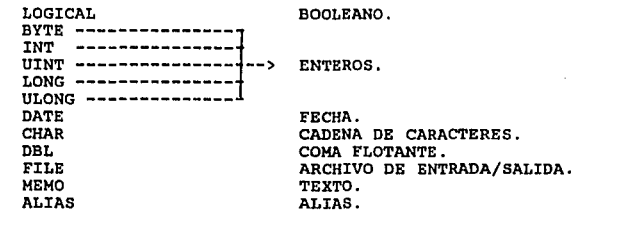
2/NOMBRE DE UNA IMAGEN DE LA ESTRUCTURA DE LA BASE DE DATOS CON LA QUE SE TRABAJA EN UN PROGRAMA FORCE.

La palabra clave RETURN regresa un valor según el tipo que se haya declarado en la función.

7.Los procedimientos son rutinas independientes declarados antes de ser llamados. En el bloque PROCEDURE Y ENDPRO son declaradas tanto variables como el conjunto de instrucciones que se necesitan en el procedimiento y también las variables definidas dentro de él son locales.

8. Todos los programas empiezan con el procedimiento force_main y obvio es el último procedimiento en el programa y por lo tanto el programa principal que hace referencia a las funciones y procedimientos escritos anteriormente.

Los tipos de datos soportados por FORCE son:



Cada tipo, excepto FILE o MEMO, pueden ser como un tipo de regresador de una función. El tipo FILE permite sólo ser almacenado y no puede ser manipulado, los tipos enteros, coma flotante, fecha y caracter pueden ser manipulados con operadores relacionales o matemáticos. Algunas funciones especiales pueden manipular MEMO como cadenas de caracteres. Un tipo ALIAS puede ser asignado o pasar como un parametro.

La tabla tipo y almacenamiento resume los requerimientos para cada tipo de dato y el rango del valor que puede ser asignado y almacenada para cada tipo. El tipo FILE no esta incluido ya que los requerimientos de almacenaje dependen de las versiones del compilador.

El tipo BYTE almacena el valor entero de el caracter ASCII de 0 a 255. Ya que una variable puede basarse en otra variable, los bytes puedes ser usados para manipular cadenas de caracteres. EL tipo puede ser usado para almacenar numeros que no requieren un lugar decimal, tales como contadores, índices de arreglos y variables para renglones y columnas.

El tipo DATE es representado por 4 bytes, los cuales FORCE interpreta como un valor ULONG. La fecha 0 es un dato invalido. Que es, el tipo DATE almacena un número de dias. FORCE influye en el valor del dato como el dia 1 es Enero , año 1, conteniendo un logical 0 para falso y 1 para verdadero.

TIPO Y ALMACENAMIENTO

TYPE	ALMACENAMIENTO	RANGO
BYTE	1 BYTE	-128 a 127
CHAR	1 a 256 BYTES	CUALQUIER CARACTER ASCII EXCEPTO 0
DATE	4 BYTES 4,294 ,967,295	DIA 1 - DIA
DBL	8 BYTES	2.2E-308 a 1.7E+308 y -1.7E+308 a -2.2E-308
FILE	1K	ASIGNADO POR LIBRERIA
INT	2 BYTES	-32,768 a 32,767
LOGICAL	1 BYTE	TRUE o FALSE
LONG	4 BYTES	-2,147,483,648 a 2,147,483,647
UINT	2 BYTES	0 a 65,535
ULONG	4 BYTES	0 a 4,294,967,295

EL tipo DBL requiere una serie de rutinas de librería runtime para operaciones simples tales como adición y substracción como procesador de separador numérico. Como consecuencia , los tipos DBL requieren mucho más tiempo de proceso y resultado en programas lentos. Los datos tipo DBL usan el formato IEE(Instituto de Ingenieros Electricos y Electrónicos). El tipo DBL usa 8 bytes, 11 bits por un exponente y 52 bits por una mantisa normalizada.

FORCE no da límites arbitrarios para los rangos de los valores, los límites se dan por la arquitectura de la computadora 808x o por el estándar. Los valores LOGICAL son 2 bytes que permiten ser fácilmente manipulados con otros lenguajes; los valores bajos y altos de INT , UINT, LONG y ULONG pueden variar conforme FORCE es trabajado en diferentes procesadores.

FORCE usa constantes enteras, de coma flotante, de caracteres, logicas y fecha. Una constante entera es una secuencia de dígitos decimales representando un valor entero, por ejemplo, un número sin un lugar decimal o la secuencia de números (1,2,3,..,n). La forma de una constante decimal es "nnnn...n" donde n es un número de 0 a 9. Un tipo especial de entero byte puede ser especificado para encerrar un sólo caracter dentro de un apóstrofe. Por ejemplo, el caracter 'a' es una constante byte. FORCE implementa una función asc("a") cuando compila.

Esto es 'a' representa un valor byte de 97 que es el valor ASCII de 'a'. Una forma hexadecimal para una constante entera es "OxN" donde N es un caracter de '0-9', 'a-f' o 'A-F'. el Ox es requerido. Todas las constantes enteras son positivas al menos que se especifique con el signo '-'. Ejemplos:

```
? 1, 535, -34  && constantes enteras decimal.
? 'a', 'A'     && constante entera BYTE.
                  esta linea imporime "97 65".
?Ox1,Ox80,OxAF && constantes enteras hexadecimales
```

Una constante de coma flotante es formada por:

```
[+||-]digitos[.digitos][[E||e][+||-]digitos]
```

FORCE considera caracter al lugar decimal "." para especificar una constante como flotante. Así, si un valor entero puede ser representado como coma flotante se incluye el punto decimal. Ejemplo: 123.0

A una constante de coma flotante es asignada un número que incluye elementos enteros, fraccionales y exponentes. Los digitos son caracteres de '0' a '9'. El caracter 'E' delinea los elementos de exponentes y es asumido en base 10. El mayor y menor número real puede ser especificado con una constante doble de 1.7E307 y 1.7E-307. Para especificar un número menor que 1, usar el predecesor "0". ejemplo 0.123.

Las constantes de coma flotante consisten hasta de 15 lugares decimales. Ejemplos:

```
?1.25          && 1-1/4
?0.50          && 1/2
?1e-2          && 1/100
?37.345
?-25.34E10
```

Una constante caracter es definida como cualquier secuencia de carácteres, excepto la linea NULL o simple o doble comilla ('o"). Como se ha dicho un caracter con una sola comilla como 'a' es considerado como un número tipo byte. Si queremos manejar (' o ") como caracter lo ponemos de esta manera:

```
? "implica poner ' como caracter" o
? 'implica poner " como caracter'
```

Las constantes caracter son almacenadas en memoria con doble o simple comilla y terminan con un ASCII '0' o 'null'.

Una constante lógica puede ser .T. o .t., .F. o .f. donde el espacio en blanco no se permite entre los dos carácteres ".".

FORCE acepta a la constante DATE cuando las variables fecha son inicializadas una constante DATE es en la forma MM/DD/YYYY. Ejemplo:

```
VARDEF
      DATE a = 01/02/1990
      DATE b = 08/13/1991
ENDDEF.
```

En el programa de la figura 3, en el punto 3 se habló de la declaración de variables por medio del bloque VARDEF-ENDDEF. y se mencionó que cuando el VARDEF existe en un procedimiento o función entonces las variables son consideradas como locales. FORCE soporta tres tipos diferentes de ámbito de variables: globales, locales y privadas.

Las variables globales pueden ser referencias por todos los procedimientos o funciones dentro del módulo en el cual la variables declarada y en cualquier procedimiento o función dentro de un módulo diferente.

Las variables locales solo pueden estar en el procedimiento ó función en las cuales son declaradas.

Las variables privadas permiten todos los procedimientos o funciones solamente en el módulo en el cual han sido declaradas.

5.2 LA INTEGRACION DE LA BASE DE DATOS EN LA RED.

El término integridad de datos se refiere a la correción y completitud de los datos en una base de datos. Cuando los contenidos de una base de datos se modifican, la integridad de los datos almacenados puede perderse de muchas maneras diferentes. Por ejemplo:

- Añadir datos no válidos a la base de datos.
- Modificación de datos existentes tomando un valor incorrecto.
- Los cambios a la base de datos pueden perderse debido a un error del sistema.

FORCE íncluye un conjunto de comandos y funciones que permiten mantener tanto la integridad de la base de datos como el control de acceso en una red de área local permitiendo a los usuarios de esta red compartir datos en un cien porciento íntegros.

Estas herramientas de programación adicionales , que constan de varias órdenes y funciones, son necesarias para prevenir problemas potenciales cuando los usuarios comparten archivos. Las colisiones entre usuarios, originadas cuando los usuarios intentan utilizar el mismo archivo en el mismo instante, pueden generar problemas adicionales para los programadores de la red, debido a que el programa debe ser proporcionado con bastante inteligencia para detectar esos problemas y encontrar una solución.

Por default, los archivos de bases de datos en un programa FORCE son abiertos en modo exclusivo. Este modo índica que la base de datos no puede ser abierta por otros procesos. El comando SET EXCLUSIVE designa cuando una base de datos o índice son abiertos en modo exclusivo o compartido; SET EXCLUSIVE ON determina que una base de datos con sus archivos y sus índices abiertos con USE o OPEN son exclusivos. Los archivos pueden ser cambiados a modo compartido usando SET EXCLUSIVE OFF antes de abrir o usando la palabra clave SHARED con el comando OPEN o USE. Pero USE no es recomendable ya que se tiene que enlazar con el comando SELECT y se pierde una gran característica de FORCE. Cuando un archivo es abierto en modo compartido, otros procesos pueden abrir el archivo y leerlo o escribirlo. Por ejemplo:

```
PROCEDURE FORCE_MAIN              Se tienen todas las bases
   SET EXCLUSIVE OFF              en modo compartido
   DO ABRE_BASES
--------------------
SET EXCLUSIVE OFF                 Abre un subconjunto de la
   DO ABRE_ARCHIVOS WITH c[]      base de datos en modo com-
SET EXCLUSIVE ON                  partido y otras en modo
   DO ABRE_ARCHIVOS WITH e[]      exclusivo
```

Una forma de abrir una base de datos que va hacer compartida por varios usuarios es por medio del comando OPEN el cual abre base de datos, índices y archivos memo por medio de su alias, por ejemplo:

```
DBFDEF  DATO1
{....}
ENDDEF

DBFDEF DATO2
{....}
ENDDEF

FUNTION LOGICAL red_abre
  PARAMETERS ALIAS red_base_datos
  VARDEF
        LOGICAL procesar
  ENDDEF
  REPEAT
      OPEN red_base_datos SHARED
      IF net_err() <> 0
         @1,1 SAY "Error: procesar?";
              GET procesar
         READ
      ELSE
         RETURN .t.
      ENDIF
  UNTIL .not. procesar
ENDPRO

PROCEDURE FORCE_MAIN

IF .not.  red_abre(dato1)
   { ....}
IF .not.  red_abre(dato2)
   {....}
ENDPRO
```

Es una rutina general para base de datos en un servidor de una red de área local, dentro de la función red_abre se abre el alias de la base de datos llamado red_base_datos para ser compartida, es decir con la palabra clave SHARED se especifica el derecho de acceso a la base de datos permitiendo que ésta pueda ser usada por varios usuarios en la red, si se quiere que la base de datos sea usada por una sola persona es sustituída la palabra SHARED por EXCLUSIVE.

En el comando OPEN varias bases de datos pueden estar abiertas simultáneamente, en el ejemplo, este comando regresa NOT si no puede abrir una base de datos compartida debido a un error de red. El programa utiliza una función net_err() para obtener el código de error de la red. Si no se encontró con ningún problema para abrir la base de datos la función regresa un 0.

La función net_err() regresa el último codigo de error de red y regularmente se utiliza para obtener información de cuando un archivo no puede ser abierto, los siguientes dos fragmentos de programas muestran la utilidad de net_err():

```
*ejemplo 1
* asegura que la base de datos sea abierta

 OPEN "f:\datos\acct.dbf" ALIAS accts

 IF net_err() > 0
    ? "error de red # :", net_err()
    ? "base de datos no esta abierta"
 ENDIF


*ejemplo 2
*una función genérica que trata de abrir una base de
datos


FUNCTION LOGICAL trata_de_abrir
  PARAMETERS CONST CHAR nombre, ALIAS a,;
  VALUE INT trytime

  DO WHILE trytime > 0
     OPEN nombre ALIAS a
     IF net_err() > 0
        trytime = trytime - 1
        delay( 1 )
        LOOP
     ENDIF
     RETURN .T.
ENDPRO
```

Una manera firme de mantener integra una base de datos en una red es usar archivos compartidos. El desarrollo de aplicaciones puede decidir cuando los archivos pueden ser restringidos al uso de más de un usuario. A menos que se especifique otra cosa, las aplicaciones FORCE son compiladas para un solo usuario con base de datos exclusivas.

EL diseño en una aplicación buscando integridad determina que elementos de la base de datos pueden tener candados. Un candado es opcional excepto cuando la información vaya ser modificada y escrita a una base de datos.

Los comandos para manejo de base de datos como: APPEND BLANK (Agrega un registro en blanco), DELETE(marca un registro para ser borrado), EDIT(edita y permite cambiar un registro), RECALL(restaura registros que han sido marcados para borrar) y REPLACE(remplaza campos en un registro), requieren un candado en el registro.

Si múltiples registros son modificados en una base de datos por comandos tales como APPEND FROM y DELETE, la base de datos puede ser usada exclusivamente a través del comando USE EXCLUSIVE. Antes de ejecutar comandos que lean información de cada registro en una base de datos, incluyendo los comandos COUNT(cuenta registros) y SUM(suma registros), la base de datos debe tener candado.

Algunos comandos, incluyendo INDEX(indexa), PACK(borra registros marcados), REINDEX(reindexa) y ZAP(borra todos los registros), funciones solo con uso exclusivo o archivos designados. Comandos de solo lectura tales como GOTO, LIST, REPORT y SEEK, no requieren base de datos con candado. APPEND BLANK puede ser usado con bases de datos compartidas.

Podemos concluir entonces, que laa orden SET EXCLUSIVE controla si los archivos están dsiponibles para uso compartido o para uso exclusivo del primer usuario que acceda al archivo. Una vez que un programa abre un archivo con SET EXCLUSIVE OFF, es responsabilidad del programa protegerse contra las posibles colisiones potenciales revisando el estado de los archivos y registros con las funciones de candados FLOCK y RLOCK, entre otras, que se veran más adelante.

Un candado en un registro asegura que dos usuarios no lo modifiquen simultáneamente, si esta situación ocurre ambos asumen que su cambio es salvado en la base de datos, aunque sólo el último registro del último usuario es el que ha sido cambiado, provocando esto una pérdida de integridad de alto riesgo.

Las funciones de candados pueden utilizarse para prevenir colisiones. Estas funciones permiten que un programa sepa si un archivo o registro está con candado puesto por otro usuario de la red. Las funciones de candados operan de forma algo diferente a otras funciones. Cuando otras funciones de FORCE normalmente devuelven un valor(tal como verdadero o falso), las funciones de candados pueden realizar una acción(el bloque de un registro o de un archivo) así como devolver un valor.(Se estudiará en control de acceso)

Cuando abrimos archivos compartidos el comando EXCLUSIVE OFF puede ser ejecutado antes que el comando USE sea ejecutado para asegurar que el archivo intentado sea abierto. La clausula INDEX no puede ser empleada en un comando USE. Los archivos indexados pueden abrirse con SET INDEX después de que la función net_err() haya sido evaluada. Como hemos visto en el ejemplo anterior, cuando los comandos OPEN o USE fallan por un uso concurrente en la red, la función net_err() es tomada como un evaluador para no continuar con el proceso y así evitar tener alguna degradación en la base de datos en uso.

No hay que olvidar, tener en cuenta que ocasionalmente existen fallas de hardware y diversos tipos de accidente que se encuentran fuera de los límites de los SMBD, pudiendo destruir los datos almacenados, de aquí que es necesario proteger los datos de este tipo de accidentes e incluir ciertas reglas y pruebas de cheque que aseguren la integridad y pérdida de la información.

5.2.1 CARACTERISTICAS

Para la creación de una estructura de base de datos, FORCE utiliza un comando BUILD como se muestra en el siguiente programa:

```
DBDEF empleado
  CHAR(20)  nombre
  DATE      fecha
  DBL(5:2)  salario
  CHAR(30)  puesto
ENDDEF

PROCEDURE FORCE_MAIN
   BUILD "EMPLEADOS.DBF"  FROM ALIAS empleado
   BUILD "TIPO.DBF"       FROM FIELDS;
         empleado->nombre, empleado->puesto
ENDPRO
```

Como se observa el comando BUILD construye dos estructuras bases de datos (EMPLEADO.DBF y TIPO.DBF), la primera llamada con el alias empleado, donde se puede ver que en el bloque DBFDEF-ENDDEF se describe la estructura de la base de datos, y en la segunda base sólo es tomada dentro del bloque DBDEF-ENDDEF los campos nombre y puesto. Hay que aclarar que en cualquier programa que utiliza una base de datos tenemos que trabajar con su alias. Para crear y modificar una base de datos FORCE trabajamos con una una utilidad llamada CREATE.EXE.

Sin embargo FORCE puede leer y escribir archivos con extensión .DBF y .DBT del SMBD DBASE, significando esto que también puede manipular los archivos creados por FOXBASE, dBXL o CLIPPER, por lo tanto podemos tomar bases de datos de cualquiera de estos SMBD, sólo nos resta agregar en los programas el bloque DBDEF-ENDDEF con su nombre del alias y sus respectivos campos. Pero FORCE utiliza un formato diferente para sus indices (.FDX) y un archivo de memoria (.MEM) que es diferente con los archivos de memoria (.MEM) de los SMBD mencionados. Si un archivo de base de datos (.dbf) existe, FORCE tiene integrado un archivo ejecutable DBFDMP.EXE que muestra la estructura de una base de datos y puede guardarla en un archivo de texto.

Ya que FORCE permite abrir bases de datos simultáneamente, si queremos trabajar con alguna de estas bases sólo hacemos referencia al alias especifico de la base, por ejemplo:

```
!empleado SEEK in_nombre
```

Busca en empleados.dbf el campo nombre sin necesidad de cerrar o abrir otras bases de datos, ya que hemos definido al alias de la base empleados.dbf como empleado.

Como hemos visto las características de FORCE incluyen tipos de variables, valores numéricos que pueden ser especificados como enteros o coma flotante, permitiendo emplear el más eficiente tipo de dato para la aplicación requerida, arreglos para cada tipo de dato, operadores booleanos y relacionales y matemáticos.

Comandos tales como INDEX, SEEK y REPLACE (indexar, buscar y remplazar), funciones tales como EOF() y RECNO() (fin de archivo y número de registro) estan diseñadas específicamente para aplicaciones de base de datos.

Ciertas palabras son reservadas como palabras claves, las cuales son reconocidas por el compilador como un medio predefinido. Un identificador es una serie de hasta 32 caracteres "A-Z", "a-z", "0-9" y "_", donde el primer caracter no debe ser un espacio o número, siendo utilizado para nombre de variables, procedimientos, funciones, macros, alias etc..

El caracter "*" al principio de la linea o precedido sólo por espacios denota un comentario. El "&&" denota un comentario entre una linea, cualquier caracter seguido de "*" o "&&" es ignorado por el compilador. La palabra NOTE puede ser usada en lugar del caracter "*".

FORCE contiene un robusto conjunto de funciones a nivel sistema, es decir regresan información sobre el hardware de la computadora en el cual un programa en FORCE está corriendo. También proveen información sobre el sistema operativo, la siguiente figura muestra un resumen de estas funciones :

```
CHDIR()       CAMBIA DIRECTORIO EN DOS
CHMOD()       CAMBIA O REGRESA ATRIBUTOS DE ARCHIVOS
CRITICAL      PERMITE A UN PROGRAMA INSTALADO SABER SUS ERRORES CRITICOS
CURDIR()      REGRESA EL DIRECTORIO DE TRABAJO
CURDRIVE()    REGRESA EL NUMERO DE UNIDAD DE DRIVE DE TRABAJO
DISKSPACE()   REGRESA EL NUMERO DE BYTES LIBRES DE UN DRIVE
EQUIPMENT     REGRESA UNA LISTA DEL EQUIPO DEL BIOS DE LA COMPUTADORA
EXIST()       REGRESA TRUE SI EL ARCHIVO ESPECIFICADO EXISTE
FILESIZE()    REGRESA EL TAMANO DEL ARCHIVO DEL DISCO ESPECIFICADO
FIND-FATTR()  REGRESA EL ATRIBUTO DEL ARCHIVO, DISCO O VOLUMEN
FIND-FDATE()  REGRESA LA FECHA DE UN ARCHIVO O DIRECTORIO
FIND-FEXT()   REGRESA LA EXTENSION DE TRES CARACTERES DEL ARCHIVO
FIND-FIRST()  LOCALIZA EL PRIMER ARCHIVO PATERNO UNIDO
FIND-FSIZE()  REGRESA EL TAMANO DEL ARCHIVO ENCONTRADO
FIND-FSTR()   REGRESA EL NOMBRE DEL ARCHIVO, DIRECTORIO O VOLUMEN DEL DISCO
FIND-FTIME()  REGRESA EL TIEMPO DE CREACION DE UN ARCHIVO O DIRECTORIO
FIND-NEXT()   ENCUENTRA EL SIGUIENTE ARCHIVO EN SECUENCIA
GETENV()      REGRESA EL VALOR DE UNA VARIABLE DE AMBIENTE
GET-EXEC()    REGRESA TODO EL NOMBRE DE LA RUTA DEL PROGRAMA QUE SE ESTA CORRIENDO
ISCOLOR()     REGRESA TRUE SI LA TARJETA DE VIDEO ES DE COLOR
MKDIR()       CREA UN SUBDIRECTORIO EN DOS
MORE-HANDLES HABILITA UN PROGRAMA PARA ABRIR 255 ARCHIVOS A LA VEZ
OS()          REGRESA AL SISTEMA OPERATIVO DOS
OS-VER        REGRESA LA VERSION DEL SISTEMA OPERATIVO
PATH-OF()     BUSCA A TRAVES DE TODOS LOS CAMINOS DESCRITOS DENTRO DEL PATH
RMDIR()       REMUEVE UN SUBDIRECTORIO DOS
SELECT-DRIVE CAMBIA EL DRIVE DE TRABAJO AL DRIVE SELECCIONADO
TIME()        REGRESA UNA REPRESENTACION DE 8 CARACTERES DE LA FECHA DEL SISTEMA
TODAY()       REGRESA LA FECHA DEL SISTEMA
```

FORCE usa el ciclo edita-compila-encadena. FORCE no contiene un editor, ni encadenador, cualquier editor de texto y cualquier encadenador que soporte el estandar INTEL/MICROSOFT (.OBJ) trabaja con FORCE.

Como el lenguaje "C", FORCE soporta dos tipos de funciones de archivo de bajo-nivel: Buffer y binario. Las funciones buffers son manejadas para manipular archivos de texto, mientras que las funciones de archivo binario pueden ser usadas para manejar cualquier tipo de archivos incluyendo .DBF y .DBT.

Para ciclos y alcances basados en ciertas condiciones FORCE maneja el IF, DO CASE, DO WHILE, REPEAT y FOR.

Otras características de FORCE son:

Creación de programas residentes (TSR) sin la necesidad de un conocimiento profundo del manejo de interrupciones.

Manejo de interrupciones al BIOS y DOS utilizando la función llamada INTERRUPT.

FORCE utiliza aproximádamente un 70% de las instrucciones del SMBD DBASE haciéndolo un lenguaje amigable, elabora reportes, etiquetas y formatos de pantalla con la misma lógica que DBASE, con los comandos como SET FORMAT TO, SET LABEL TO, y otro llamado SET REPORT TO, la diferencia es que estos tres tipos de salidas son generados en FORCE a través de código fuente.

FORCE se puede considerar un lenguaje de bajo nivel al estilo de "C" o PASCAL en donde se aprovechan conocimientos del DBASE, desarrollando complejos programas con gran facilidad. FORCE se puede agregar a C o a ensamblador dándole a estos dos una forma más simple de programar y con mejor comprensión de procedimientos y funciones.

En el ámbito de redes locales, sabemos que la entereza de la base de datos es amenazada siempre que dos usuarios intentan modificar el mismo registro de la base de datos a la vez. Si el software no está diseñado para operar en una red, pueden ocurrir varios problemas. Un usuario puede escribir encima de los cambios de otro o, en casos más extremos, el software operativo de la red puede fallar y venirse toda la red abajo. En el idioma de red, un desastre de este tipo se denomina colisión, o puede ocurrir que los programas ejecutan bucles infinitos intentando dar uso exclusivo al mismo archivo a más de un usuario de red. Para prevenir tales problemas, FORCE ofrece dos características el candado de archivo y el candado de registro. El candado de archivo hace que un archivo de base de datos que está utilizando un usuario no esté disponible para otro usuario de la red.

El candadado de registro realiza el mismo tipo de salvaguarda; pero lo hace sobre un registro individual del archivo. FORCE realizará el candado de archivo y de registro automáticamente cuando sea necesario mantener la integridad de los datos de cualquier base de datos que utilice. Además del candado de registro y archivo, puede utilizar órdenes específicas para conectar o desconectar un candado de registro o archivo.(Este punto se analizará en control de acceso).

Finalmente podemos decir que FORCE no es sólo aplicable al manejo de base de datos, puede ser utilizado para desarrollos matamáticos, manejo de pantallas y de opciones múltiples, manipulación de interrupciones, procedimientos residentes en memoria, manejo de archivos y registros para ambientes en redes locales como NOVELL y otros basados en el sistema operativo DOS y muchas otras aplicaciones.

5.2.1.1 CONTROL DE ACCESO

Hemos visto en el punto de integridad algunas técnicas utilizadas, las cuales también sirven para poder tener un control de acceso en la base de datos de una red de área local.

FORCE provee comandos para operar en una base de datos, en una área de trabajo remota. Por ejemplo:

```
!empleado SEEK empl_numero
```

Busca en empleado.dbf sin que haya ido a la primera área seleccionada, similarmente, muchas funciones de base de datos en FORCE operan en base de datos remotas simplemente especificando el alias deseado. Por ejemplo, si se quiere obtener un archivo con candado en empleado.dbf la cual está abierta en otra área, se puede llamar a la función a_flock(empleado).

FORCE incluye una gama de comandos y funciones para acceso y control en una red de área local como rlock(), flock(), unlock(), entre otras, pero sólo para trabajo en base de datos.

La función FLOCK() y RLOCK() son usadas para asegurar un proceso de acceso exclusivo a un archivo o registro en modo compartido. Cuando una función con candado es intentada, se reporta si el intento ha sido satisfecho. Una vez que al archivo o registro le ha sido puesto un candado, éste no puede ser abierto por cualquier otro usuario. Todos los usuarios pueden leer el archivo, pero sólo el usuario quien activó el candado del archivo o registro puede escribirlos.

El candado permanecerá hasta que el programa se encuentre con el comando UNLOCKS el cual quita un candado a una base o registro, se cierren los archivos ya sea con los comandos USE o CLOSE, que concluya el programa o resulte otro comando con candado para el archivo y si se intenta poner un candado en un archivo que previamente se le ha colocado un candado. FORCE intenta remover el primer candado antes de aplicar el segundo. Satisfactoriamente o insatisfactoriamente rlock() intenta remover los candados existentes. Sólo un candado puede ser activo por cada alias abierto, y en cualquier archivo abierto en el modo compartido. Por eso, sólo un registro puede tener un candado a la vez.

Si el dato es invaluable y tanto flock() y rlock() regresan FALSE, el programa puede alertar la falla del candado y crear otras opciones.

UNLOCKS puede ser invocado tan pronto como un candado no sea utilizado por el usuario. Esto es, para que el registro pueda ser accesado por otros procesos o usuarios.

Como hemos mencionado algunos comandos tales como PACK o ZAP, trabajan en una red en desarrollo sólo cuando los archivos o registros están designados para uso exclusivo.

Comandos o funciones con uno o varios registros son mostrados en la tabla siguiente:

COMANDO	REQUERIMIENTO: FUNCION o COMANDO
@...SAY...GET	RLOCK()
APPEND FROM	USE...EXCLUSIVE O FLOCK()
DELETE (UN REGISTRO)	RLOCK()
DELETE (VARIOS REGISTROS)	USE...EXCLUSIVE O FLOCK()
PACK	USE...EXCLUSIVE
RECALL (UN REGISTRO)	RLOCK()
RECALL (VARIOS REGISTROS)	USE...EXCLUSIVE O FLOCK()
REINDEX	USE EXCLUSIVE
REPLACE (UN REGISTRO)	RLOCK()
REPLACE (VARIOS REGISTROS)	USE...EXCLUSIVE O FLOCK()
ZAP	USE...EXCLUSIVE

TABLA DE COMANDOS DE RED

Se ha hablado de las funciones flock(), rlock() y el comando UNLOCK, toca en este momento definir su aplicación particular y algunos ejemplos:

Hay que recordar que FORCE administra sus bases de datos en base al alias definido en cada una de ellas, la función flock() intenta poner un candado al alias con el que se esta trabajando. Si se puede poner el candado a la base de datos satisfactoriamente flock() regresa TRUE (verdadero). Cualquier candado, registro o archivo asignado al alias de trabajo son borrados y si flock() regresa FALSE (falso) implica que los candados no se modificarán.

Todos los archivos con un candado en el alias de trabajo incluyen índices y archivos memo 3/ . Estos fragmentos de programas FORCE muestran como trabaja ésta función :

```
* ejemplo 1
* pone un candado a la base de datos antes de  borrarla

  DBDEF nombres
        char(20)  nombre
        char(20)  direcc
  ENDDEF


  PROCEDURE force_main
      VARDEF
          CHAR(1) s_n
      ENDDEF
```

3/LOS ARCHIVOS MEMO SON CAPACES DE ALMACENAR GRANDES BLOQUES DE INFORMACION DE LOS CAMPOS MEMO DE UNA BASE DE DATOS.

```
OPEN "f:\datos\nombres.dbf" ALIAS nombres SHARED
ACCEPT "Borra la base de datos?" TO s_n
IF "Ss" $ s_n
    IF .NOT. flock()
        PACK
    ELSE
        ? "no puede traer la base de datos"
        ? "intente otra vez por favor"
    ENDIF
ENDIF

* ejercicio 2
*pone un candado a la base de datos antes de usar SUM

 IF .NOT. flock()
     RETURN .F.
 ENDIF
 SUM ALL TO cuenta_total
```

La función rlock() intenta poner un candado al registro de trabajo en la base de datos de trabajo. Si es posible poner el candado rlock() regresa TRUE. Cualquier candado, registro o archivo asignado al alias de trabajo son borrados y si la función rlock() regresa FALSE, indica que no se pudo poner el candado.

Una vez que rlock() es aplicado a un proceso, ningún otro proceso puede escribir al registro o candado del registro. Sin embargo, rlock() no previene otros procesos de lectura al candado del registro. Estos fragmentos de programas ejemplifican la función rlock().

```
*ejemplo 1
*pone un candado al registro antes de  usar el comando
*EDIT

IF .NOT. rlock()
    ? "no puede accesar al registro"
ENDIF
EDIT RECORD recno()
UNLOCK

*ejemplo 2
*rlock() mueve cualquier candado previo.
*(NOTA: aunque rlock() es una función, se puede trabajar
*como un procedimiento, por ejemplo llamar  la función e
*ignorar el valor regresado. FORCE muestra una precaución
*cuando esto sucede, pero aun así no es ilegal)

GOTO 1
  rlock()          && pone candado al registro 1
GOTO 2
  rlock()          && borra candado en el registro 1
                   && y pone candado al registro 2
```

El comando UNLOCK borra un archivo o registro con un candado en el área de trabajo concurrente.

Es decir remueve todos los candados en el área de trabajo concurrente, sin importar que el candado activo sea un archivo o registro. Por ejemplo los siguientes dos fragmentos de programas muestran su aplicación:

```
*ejemplo 1
*Se pone UNLOCK a un registro después de que el usuario
*ha editado el registro.

PROCEDURE edita_regist
 PARAMETERS CONST CHAR(14) el_nombre, ALIAS z

    IF .NOT. a_rlock(z)
       IF .NOT. dispuesto_a_esperar()
          RETURN
       ENDIF
    ENDIF
    @3,4 GET el_nombre
    UNLOCK
ENDPRO

*ejemplo 2
*Se pone UNLOCK a la base de datos después de que ha sido
*indexada.

IF flock()
   INDEX index_alias
   UNLOCK
ENDIF
```

Como se puede observar el ejemplo 1 de UNLOCK muestra una nueva función a_rlock(), existe también la función a_flock(), estas funciones se describen a continuación:

a_flock() intenta poner un candado al alias pero pasado como un parametro. Si a la base de datos le ha sido satisfactorio poner el candado, la función a_flock() regresa TRUE. Cualquier candado, registro o archivo asignado al nombre del alias es borrado y si f_lock() regresa FALSE los candados no son restaurados. Todos los archivos asociados con el nombre del alias puede ponérseles candado, incluyendo archivos índices y memo. Estos dos fragmentos de programas muestran la función de a_flock():

```
*ejemplo 1
*pone un candado al alias antes de indexar el archivo

DBFDEF nombres
   CHAR(20)  nombre
   CHAR(20)  direcc
ENDDEF
```

```
INDEXDEF
   CHAR(20)  nind  nombres->nombre
ENDDEF

PROCEDURE force_main
    VARDEF
       CHAR(1) s_n
    ENDDEF

OPEN "f:\datos\nombres.dbf" ALIAS nombres SHARED
ACCEPT "indexas la base de datos?" TO s_n
IF "sS" $ s_n
    IF .NOT. a_flock(nombres)
       INDEX  nind
    ELSE
       ? "no se puede traer la base de datos"
       ? "intente otra vez"
    ENDIF
ENDIF

*ejemplo 2
* pone candado a la base de datos antes de usar COUNT o
* SUM para asegurar un buen reporte.

IF .NOT. a_flock(nombre_dato)
    RETURN .F.
ENDIF

!nombre_dato COUNT ALL TO cuenta_total
```

La función a_rlock() intenta poner un candado al registro de trabajo en el nombre del alias específico. Si es puesto el candado a_rlock() regresa TRUE. Cualquier candado, registro o archivo de trabajo asignado por el nombre del alias son borrados y si a_rlock() regresa FALSE los candados no son restaurados. Dos fragmentos de programas ejemplifican esta función:

```
*ejemplo 1
*pone un candado al registro antes del comando GET.
*Asegurar que el tiempo entre el GET y el READ sea
mínimo,
*ya que otros procesos pueden esperar para el registro.


IF .NOT. a_rlock() (nombre_alias)
   ?inhabilitado accesar al registro..."
ENDIF

@12,10 SAY "Nombre: ";
       GET nombre_alias->nombre
READ
UNLOCK
```

```
*ejemplo 2
*función que espera un registro. La espera de N segundos
*para un registro.

FUNCTION LOGICAL espera_un_registro
 PARAMETERS VALUE INT segundos, ALIAS a
   VARDEF
       LONG ts
   ENDDEF

   ts = time_segundos() + segundos
   REPEAT
       IF a_rlock(a)
           RETURN .T.
       ENDIF
   UNTIL time_segundos() > ts
   RETURN .F.
ENDPRO
```

Las funciones mencionadas en este punto flock(), rlock(), a_flock(), a_rlock() necesitan estar en el #INCLUDE database.hdr, por lo que siempre que se haga referencia a éstas se debe poner esta directiva.

Hay que remarcar la importancia de saber definir cuales son los usuarios que tienen acceso a ciertas bases de datos, como que base de datos van a fungir como compartidas, que usuarios deben tener permitido actualizar datos de una base, a quienes sólo se permita recuperar datos o tener denegado el acceso total de ciertas bases de datos. Esto con el fin de hacer programas que contengan un verdadero control al acceso de la base o base de datos en uso.

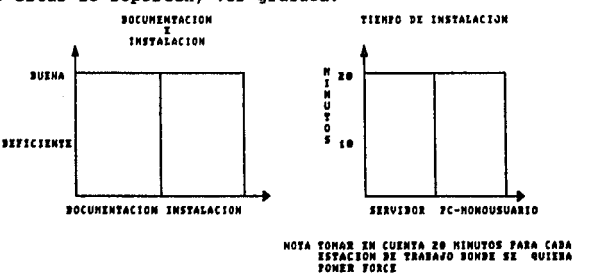
En estadísticas de utilización se mencionará otra técnica de control de acceso por medio de APIs del sistema operativo para red NOVELL-NETWARE y FORCE.

5.2.1.2 ESTADISTICAS DE UTILIZACION

La documentación de FORCE incluye un manual de referencia de lenguaje y un manual de funciones y comandos. El manual de referencia define a FORCE como lenguaje de programación y describe la sintaxis para cada comando y función. FORCE es un SMBD con su propio y extenso subconjunto de comandos, sintaxis y estructura. El manual de funciones y comandos describe la sintaxis para cada comando y función soportada por FORCE, también explica a FORCE como compilador, los archivos de encabezado, la sintaxis de los comandos propios de línea para el compilador FORCE y el uso del desarrollo de DOS, la compilación condicional, los errores de compilación, ligas de programas creados en FORCE con programas creados en otros lenguajes (en particular lenguaje C).

Ambos manuales contienen referencias a términos computacionales y conceptos que pueden confundir a programadores novatos. La documentación de FORCE no esta diseñada para enseñar computación y programación o los usos rudimentarios de programación en base de datos. Estos manuales están escritos para programadores que están familiarizados con los términos antes mencionados y otros.

La instalación del SMBD FORCE; es fácil y es desde el programa intall.exe. El programa de instalación crea un directorio de trabajo del compilador y sus librerías; en el caso de instalación en red de área local es necesario instalarlo en la máquina que fungirá como servidor de archivos y si se desea alguna aplicación monousuario es posible cargarlo en cualquier estación de trabajo siempre y cuando éstas lo soporten, ver gráfica.



FORCE tiene manejo de archivos para ambientes de red como el sistema operativo NOVELL-NETWARE y otros basados en el sistema operativo DOS. De ahí que los protocolos soportados por el sistema operativo para red son soportados por FORCE.

En el caso del sistema operativo NOVELL-NETWARE, éste soporta su propio protocolo IPX/SPX, en menor cantidad los protocolos NETBIOS y NAMED PIPES, pero éste último sólo en las estaciones de trabajo clientes bajo el sistema operativo OS/2.

FORCE directamente no produce interfaces gráficas, pero este SMBD se apoya en un paquete llamado DGE, que es una librería que facilita la creación e impresión de cualquier tipo de gráficas (gráficas de barra, de línea, de pastel, entre otras), mediante funciones para una aplicación, permitiendo desarrollar y explotar gráficas a un entorno de alto nivel.

FORCE no soporta ninguna interacción con hojas electrónicas y como no trabaja en la arquitectura cliente-servidor, tampoco soporta frontales ya que no funge como servidor de base de datos.

FORCE se puede agregar o relacionar al lenguaje C o a ensamblador dándole a estos dos últimos una forma más simple de programar y con mejor comprensión de procedimientos y funciones, para aplicaciones no necesariamente de tipo científico, sino de manejo de base de datos, que hace su relación mucho muy fuerte.

Se han desarrollado un gran número de utilidades de C y ensamblador para la red NOVELL usando APIs de NOVELL. Para codificar un programa en lenguaje C usando funciones API, se necesitan los archivos de encabezado de NOVELL (nit.h y niterror.h) y la librería API tal como Lnit.lib. Esta gran facilidad de codificar una aplicación de base de datos en FORCE cubre los servicios de red a través de los APIs.

Para evitar tener que reescribir los voluminosos archivos de encabezado de NOVELL, primero se compila la utilidad en lenguaje C, segundo se enlaza el resultado del código objeto con nuestro programa FORCE.

La figura 1 y 2 muestran un ejemplo práctico de como se puede ir tomando información del cuaderno de NOVELL dentro de la aplicación FORCE. Se puede usar el programa para ayudar a producir un único nombre de archivo por un archivo temporal en una red, o dejar un rastro de quién usó la aplicación de base de datos.

GetConnectionNumber() y GetConnection Information() son funciones del API NOVELL. Esta última regresa un útil dato sobre la estación de trabajo, incluyendo usuario, tiempo de login y número de estación

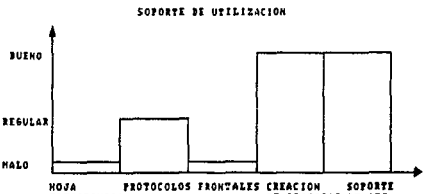
```
/* PEQUEÑO PROGRAMA EN TURBO QUE REGRESA EL NOMBRE DEL USUARIO
/* Y EL NUMERO DE CONECCION. COMPILADO CON
/* TCC -C -ml NOMBRE-PROGRAMA

#INCLUDE <STDIO.H>
#INCLUDE <DIT.H >
#INCLUDE <NITERROR.H >

CHAR *USERID;CHAR *OBJECTNAME; WORD *CONNECTNUMBER;
{

/*    PALABRA CONNECTNUMBER */
     LONG OBJECTID;
     WORD OBJECTTYPE;
     BYTE LOGINTIME[7];

/* TOMA EL NOMBRE DEL USUARIO */

    *CONNECTNUMBER =  GETCONNECTIONNUMBER();
    GETCONNECTIONINFORMATION(*CONNECTNUMBER, OBJECTNAME,
            &OBJECTTYPE, &OBJECTID,LOGINTIME);

}
```

FIGURA 4. PROGRAMA EN C QUE REGRESA EL USUARIO Y NUMERO DE CONNECION DE UNA ESTACION DE TRABAJO DE NOVELL.

```
/* PROGRAMA QUE MUESTRA LA MEZCLA FORCE/TURBO/NETWARE
/* TOMANDO EL PROGRAMA DE C ANTERIOR, PARA UN PROGRAMA FORCE

#INCLUDE SYSTEM.HDR
#INCLUDE IO.HDR
#INCLUDE MATH.HDR
#INCLUDE STRING.HDR

/*- PROCEDIMIENTOS Y FUNCIONES EXTERNOS

   PROCEDURE SETUP-TURBOC  PROTOTYPE
   PROCEDURE USERID        PROTOTYPE        && PROCEDIMIENTO EXTERNO DE TURBO C
    PARAMETERS CHAR(48) YOURID, UINT STANO && VISTO ARRIBA

/*- PROGRAMA PRINCIPAL

   PROCEDURE FORCE_MAIN
        VARDEF
           CHAR(48) YOURID
           UINT STANO
        ENDDEF

   DO SETUP_TURBOC            && INICIA SIMULACION DE TURBOC
                              && EMPIEZA TURBOC
   YOURID =SPACE(48)
   STANO =0
   DO USERID WITH YOURID, STANO
      ?" EL USUARIO ES" , YOURID
      ?"EL NUMERO DE ESTACION ES" , STANO
   ENDPRO
```

FIGURA 5. MUESTRA COMO TOMAR UN USUARIO Y EL NUMERO DE CONECCION DE UNA ESTACION DE TRABAJO DE LA RED NOVELL, DE UN PROGRAMA COMPILADO EN C Y EL USO DE ESTE EN UNA APLICACION DE FORCE.

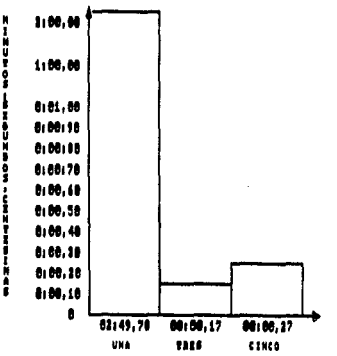
La siguiente gráfica muestra algunas estadísticas de soporte de utilización de FORCE:



Las mismas pruebas efectuadas en el capítulo 4 para SQL SERVER, sobre la red local con cinco estaciones de trabajo, son efectuadas para FORCE, solo que la máquina que sirvió como servidor de base de datos es eliminada para estas pruebas y el sistema operativo fue NOVELL-NETWARE 311. Las gráficas siguientes muestran los resultados arrojados. (refiérase a la página 129 y 130 para ver las pruebas)

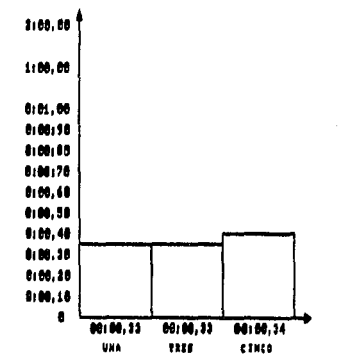
PRUEBA 1

BUSQUEDA POR INDICE SATISFACTORIA



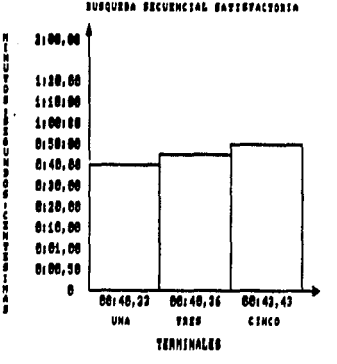
PRUEBA 2

BUSQUEDA SECUENCIAL INSATISFACTORIA



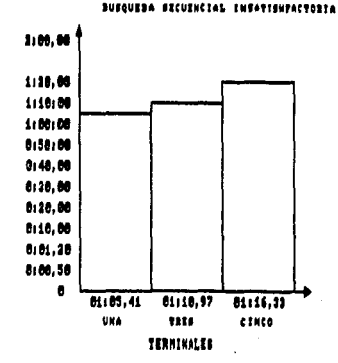
PRUEBA 3

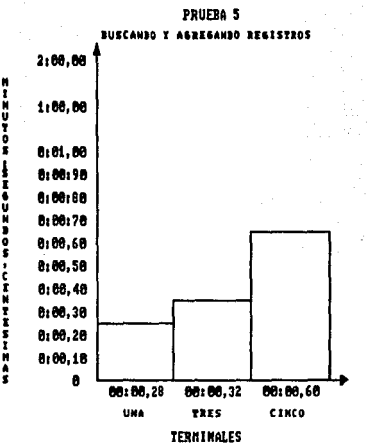
BUSQUEDA SECUENCIAL SATISFACTORIA



PRUEBA 4

BUSQUEDA SECUENCIAL INSATISFACTORIA





Los resultados de estas pruebas son analizados junto con los arrojados en el capítulo 4, en el siguiente capítulo de esta tesis.

5.2.1.3 VENTAJAS.

Indudablemente se ha visto a lo largo de este capítulo, algunas de las ventajas que ofrece FORCE como SMBD, lenguaje y compilador. FORCE es un lenguaje profesional de programación tal como el C o PASCAL. FORCE usa sintaxis compatible con el SMBD DBASE con numerosas extensiones para hacer un potente compilador. Desarrolladores de una amplia diversidad de programación pueden capitalizar fácilmente el uso de DBASE, CLIPPER y el poder y organización de C.

FORCE utiliza aproximadamente un 70% de las instrucciones de DBASE III+ haciéndolo un lenguaje amigable considerando que la mayoría de los programadores conocen a DBASE.

FORCE convierte los programas en pequeños y rápidos archivos ejecutables (.exe) usando un conjunto de reglas que permiten compilar tal programa y no hacerlo intérprete. Es decir, quita mucho la ambiguedad asociada con el lenguaje DBASE. La primera diferencia operacional entre FORCE y DBASE es que la estructura de base de datos y variables son declaradas como programas que estan escritos en FORCE. Ya que quita el diseño interpretativo el runtime 4/ . FORCE crea un código mas pequeño y rápido que cualquier compilador de base de datos.

A favor de esto, los programas pueden ser fácilmente manejados por un grupo de desarrolladores o por un programador individual. El compilador FORCE produce pequeños archivos ejecutables conteniendo solo las rutinas de la librería necesaria para ejecutar el programa. Si el programa no usa un comando que requiera de un archivo de entrada-salida, entonces en el programa de FORCE no incorpora esta librería.

FORCE permite enlazarse con otros lenguajes tales como C o ensamblador. Debido a su estructura dinámica de programación que lo hace un digno lenguaje de programación a competir, con un conjunto de herramientas disponibles para la manipulación de bases de datos.

Los programas en FORCE pueden ser puestos dentro de memoria residente(TSR) con un pequeño código, teniendo varias aplicaciones prácticas, sin la necesidad de un conocimiento profundo del manejo de interrupciones. Una de éstas puede ser un setup de una impresora en una red de área local, un programa TSR que los usuarios puedan llamar mientras están en la mitad de sus procesos o en sus procesadores de palabras. Este programa TSR de FORCE típicamente puede guardar los nombres de las colas de la red y los códigos de control para todo el soporte de impresoras.

4/RUNTIME INDICA QUE SOLO SE PUEDE TRABAJAR EN EL AREA QUE ABARCA EL SMBD.

Dos funciones que se usan para poner un programa dentro de la memoria residente son PROC_ENTRY() y TSR(). TSR() acepta un argumento el cual es agregado a la memoria RAM y guarda después el tamaño requerido de la imagen del programa. La función UNLOAD_TSR mueve el programa de la memoria, y la TSR_INSTALL puede ser usada para checar si el programa esta listo en memoria. La función TIMER_ENTRY(), establece un procedimiento para ser llamado de una manera periódica. Un programa residente en memoria es como el que se muestra a continuación:

```
* ejemplo 1
* instala un reloj TSR. Imprime el tiempo en la parte
* superior izquierda de la pantalla
#INCLUDE tsr.hdr
PROCEDURE reloj
    VARDEF
       INT r,c
    ENDDEF
    r = row()
    c = col()
    @ 0,68 ?? time()
    @ r,c
ENDPRO

PROCEDURE force_main
    IF .NOT. timer_entry( reloj, 1, &TSR_TIME_SEC,;
      &TSR_CALL_ANY)
      ? "No puede instalar reloj."
    ENDIF
    tsr(0)
ENDPRO
```

FORCE contiene una manipulación de archivos de diferentes maneras pudiendo utilizar los .DBF tradicionales, agregando archivos en forma binaria y texto. Y generación de archivos objetos (.OBJ) nativos.

FORCE permite el manejo de las interrupciones al BIOS y DOS utilizando la función llamada INTERRUPT al estilo Borland. Por ejemplo, para manipular los puertos serial o paralelo en el lenguaje COBOL sería una obra de arte, pero para FORCE sería posible debido a este manejo de interrupciones.Pero una desventaja es que en el manual de FORCE no se encontrará esta función.

Para FORCE, elaborar reportes, etiquetas y formatos de pantalla es realmente fácil ya que utiliza la misma lógica que DBASE III+, donde lo único que se debe saber en FORCE es que existe un comando llamado "SET FORMAT TO.." otro que se llama "SET LABEL TO.." y "SET REPORT TO", la diferencia es que estos tres tipos de salidas son generadas en FORCE a través de código fuente y no así en DBASE III+.

Y finalmente las ventajas de poder aplicar todas estas ventajas en conjunto de programas o sistemas que puedan correr en una red de área local de una forma transparente (específicamente redes NOVELL).

5.2.1.4 LIMITACIONES

Este SMBD no trabaja en una arquitectura cliente/servidor, ya que no se basa en el estándar SQL, este SMBD sólo corre en PCs con sistema operativo DOS,tipo AT compatible con IBM, requiriendo como mínimo tanto para PC, como para el servidor de archivos, 256 kilobytes de memoria RAM, sin embargo para un mejor alcance en disco duro, 640 kilobytes es recomendable, un procesador 80286, 20 megabytes en disco duro o mayor de preferencia. Para PCs FORCE trabaja con la versión del sistema operativo DOS 2.0 en adelante, si una aplicación requiere soporte de red de área local, se necesita la versión 3.1 de DOS en adelante.Trabaja con cualquier red de área local que corra con DOS (NOVELL-NETWARE). FORCE soporta APIs de NOVELL y soporte para gráficas como se vió en estadísticas de utilización, capacidad para acceso múltiple y simultáneo, con protección de archivos y registros, no permite frontales.

FORCE contiene un número máximo de campos por registro de 255, un número máximo de caracteres por registro de 4000, un número máximo de registros por base de datos de 2 mil millones, un número máximo de caracteres por campo tipo caracter de 255, 15 dígitos máximos de presición en campos numéricos, no tiene limitación para manejo de variables de memoria, así como para arreglos en memoria, FORCE contiene un número máximo de elementos por arreglo de 65536, un máximo de archivos abiertos a un tiempo de 250, así como 250 base de datos abiertas a un tiempo y 212 índices abiertos a un tiempo. Podemos decir que algunos valores varían de acuerdo a la computadora y capacidad del disco duro.

Los nombres de funciones y procedimientos pueden tener una longitud de hasta 32 caracteres para producir archivos fuentes leíbles, pero por compatibilidad con DBASE sobre todo, FORCE limita la longitud de los nombres de campos de la base de datos a 10. Los campos caracter no pueden exceder de 255 caracteres y los campos numéricos no pueden exceder de 19 caracteres, la máxima longitud de cadena es de 254 caracteres. Un bloque de procedimiento, función o formato puede contener hasta un máximo de 64 kilobytes de código.

FORCE usa el ciclo edita-compila-enlaza. El paquete no contiene un editor, ni ligador o debugeador. Cualquier editor de textos trabaja con FORCE y con cualquier enlazador que soporte el estándar INTEL/MICROSOFT (.OBJ), se puede trabajar.

Una declaración de arreglo especifica su tipo y el número de elementos en él. Si éste es declarado caracter, entonces un tamaño opcional puede ser especificado, pero si el tamaño no es especificado, el arreglo tipo caracter asume por default 255 caracteres por elemento.

FORCE crea un espacio de almacenamiento basado en el número y tamaño de elementos. El espacio de almacenamiento no puede exceder de 65,535 kbytes. El producto del número de elementos y el tamaño de cada elemento no puede exceder los 64 kilobytes.

FORCE no incluye funciones para ventanas. Sin embargo esta limitación es solucionada con una librería del lenguaje C, llamada CODEBASE, la cual contiene un grupo de funciones para manejo de base de datos de XBASE y manejo de rutinas de pantallas, incluyendo entrada de datos, menús y funciones para ventanas.

6. ESTUDIO COMPARATIVO DE LOS MANEJADORES DE BASE DE DATOS SQL SERVER Y FORCE EN RED LOCAL

En este último capítulo se busca que el lector en base a las siguientes pruebas y estudios se encargue de tomar una decisión de cual es la mejor opción para la manipulación y administración de los datos en su red de área local. Se establecerán dos tipos de comparaciones : Las externas y las internas.

Las externas son básicamente, las actividades que influyen en el trabajo de un sistema manejador de base de datos indirectamente, en este caso como SQL-SERVER y FORCE son manejadores que trabajan en una red local de computadoras, es importante comparar los sistemas operativos para red (LAN-MANAGER y NOVELL-NETWARE) respectivamente, los cuales se encargan de establecer la comunicación de bases de datos con estaciones de trabajo, entre otras cuestiones (primer punto). Como segundo punto a comparar, son los sistemas operativos que permiten ejecutar aplicaciones, tanto a los sistemas manejadores como a los sistemas operativos para red, estos son OS/2 para SQL-SERVER y DOS para FORCE. El tercer punto es establecer una comparación de los dos tipos de arquitectura utilizada por los sistemas manejadores estudiados, la arquitectura cliente-servidor y la arquitectura servidora de archivos.

Las internas es la conjunción de la información arrojada de los capítulos cuatro y cinco, con el propósito de establecer puntos de comparación en cuanto a: características, capacidades, limitaciones, entre otros, para poder finalmente elegir la mejor opción en la manipulación de datos en una red local.

La importancia de dividir en comparaciones externas e internas radica en que cualquier sistema manejador de base de datos para redes de computadoras, forma parte de un conjunto de elementos, sin los cuales no puede desarrollar su trabajo, de aquí que para una evaluación más extricta, es necesario comparar su fuerza exterior de trabajo como son los sistemas operativos y las diferentes tecnologías existentes. Las internas indudablemente son un punto primordial para esclarecer los puntos cuestionables surgidos en los dos anteriores capítulos.

Una vez mostradas estas dos comparaciones podemos coadyuvar que manejador de base de datos en su ambiente de trabajo normal, resulta ser el más optimo en el trabajo diario de una empresa, para que así el lector pueda escoger el sistema manejador más conveniente en su red de área local o inclusive si aún no cuenta con una poder elegir todo el ambiente de trabajo (red, sistema operativo, manejador etc..) para su empresa.

COMPARACIONES DEL AMBIENTE EXTERNO DE SQL SERVER Y FORCE

PRIMER PUNTO :

Tomando en cuenta que el ambiente de estos dos SMBDs es diferente, SQL SERVER trabaja como base principal en el sistema operativo para red LAN MANAGER, el cual opera bajo el sistema operativo OS/2 y FORCE trabaja como base principal en cualquier versión del sistema operativo para red NOVELL-NETWARE, (ver punto 1.5 de capítulo 1) el cual opera bajo el sistema operativo DOS.

Estudiaremos cierta diferencias que existen al trabajar con estos dos sistemas operativos para red, las pruebas son hechas en la versión de LAN MANAGER 2.0 Y NETWARE 386 3.1.

En el NETWARE 386 3.1 las aplicaciones basadas en el servidor se pueden ejecutar en la PC que actúa como servidor, pero ese servidor nunca puede actuar como una estación de trabajo, es decir solamente trabaja con servidor dedicado, como LAN MANAGER es una aplicación que se ejecuta bajo OS/2, puede efectuar otros procesos, mientras transmite o mueve información a otras computadoras en la red. Esto significa que cada cliente en la red ejecutando OS/2 puede actuar como un servidor LAN MANAGER de impresoras, archivos o comunicaciones, y como una estación de trabajo al mismo tiempo, es decir trabaja con servidor no dedicado. También LAN MANAGER permite aplicaciones basadas en el servidor de base de datos tipo SQL

El precio de NETWARE es de $7,995 dólares, incluyendo todo excepto una copia del DOS para el servidor y el precio sigue siendo igual aunque opere en una pequeña red o si quiere cientos de personas usando el mismo servidor, pero no hay que olvidar que NETWARE maneja los ELSis para organizaciones que necesiten menos potencia en el servidor (ver punto 1.5 de capítulo 1).

Sin embargo LAN MANAGER permite ir incrementando la potencia conforme se vaya requiriendo. Una versión de LAN MANAGER para cinco usuarios cuesta $995 dólares. Añadir otros diez usuarios cuesta otros $995 dolares y si se quiere saltar a una versión de un número ilimitado de usuarios para un servidor, pagará $5,495 por el producto.

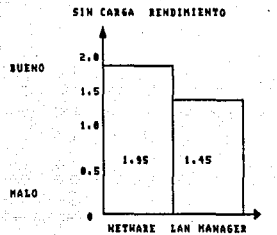
Pruebas efectuadas nos arrojarón, que la instalación en LAN-MANAGER en la máquina servidor es sencilla y rápida, unos menús fáciles de seguir lo guian a través del programa. Sin contar el tiempo de dar formato al disco duro, el proceso total tomo 20 minutos. En NETWARE, para configurar el servidor, simplemente se va contestando preguntas acerca del tipo de los adaptadores de redes y los discos que se están usando en un tiempo de 15 minutos, sin contar el tiempo necesario para preparar el disco duro.

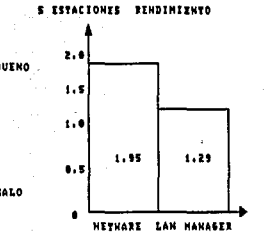
En la siguiente gráfica se muestra un resumen general de las características de estos dos sistemas operativos para red.

	LAN MANAGER	NETWARE
INFORMACION GENERAL		
COMPARTE RECURSOS DE USUARIO A USUARIO	SI	NO
OPCION DE SERVIDOR NO DEDICADO	SI	NO
OPERACIONES DEL SERVIDOR		
TOLERANCIA CONTRA FALLOS	SI	SI
MAXIMO NUMERO DE CONEXIONES SIMULTANEAS	1000	250
SOFTWARE DEL SERVIDOR		
REQUIERE OTRO SISTEMA OPERATIVO	OS/2	DOS
PERMITE CACHING DEL DISCO	SI	SI
RAM REQUERIDO	5MB	3MB
RAM RECOMENDADO	9MB	6MB
SOFTWARE DE LA ESTACION		
RAM DISPONIBLE PARA APLICACIONES	549K	536K
RAM DISPONIBLE CON NETBIOS	549K	516K
RAM RECOMENDADO PARA ESTACIONES OS/2	3.5MB	3.5MB
SE AUTOTORRECONECTA	SI	NO
SISTEMA DE ARCHIVOS		
ARCHIVOS ABIERTOS CONCURRENTEMENTE POR SERVIDOR	16000	100000
VOLUMENES POR SERVIDOR	26	64
DISCOS FISICOS POR VOLUMEN	24	32
DISCOS FISICOS POR SERVIDOR	24	1024
TAMANO MAXIMO DE ARCHIVO	2GB	4GB
TAMANO MAXIMO DE VOLUMEN	2GB	32TB
CANTIDAD FISICA MAXIMA DE RAM	16MB	4GB
CAPACIDAD MAXIMA DE DISCO	4BGB	32TB
ADMINISTRACION		
MANTIENE REGISTRO HISTORICO DE ERRORES/ESTADO	SI	SI
MANTIENEN BASE DE DATOS DE CONTABILIDAD POR USUARIO O RECURSO	SI	SI
REPORTA ERRORES EN LA RED	SI	SI
MUESTRA NOMBRES DE USUARIOS CONECTADOS	SI	SI
MUESTRA PORCENTAJE DEL SERVIDOR EN USO	NO	SI
SEGURIDAD		
PUEDE CONCEDER ACCESO POR NIVEL	SI	SI
PERMITE CONTROL DE ACCESO POR FECHA Y HORA	SI	SI
HERRAMIENTAS DE PROGRAMACION DISPONIBLES		
APIS	SI	SI
NAMED PIPES	SI	SI
IMPRESION		
NUMERO DE IMPRESORAS SOPORTADAS	ILIMITADAS	ILIMITADAS
IMPRESORAS EN LAS ESTACIONES DE TRABAJO	SI	SI
USUARIOS PUEDEN MODIFICAR LAS COLAS DE IMPRESION	SI	SI

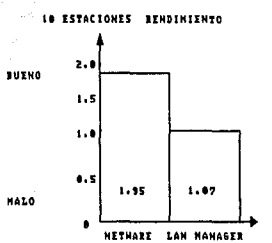
RESUMEN DE CARACTERISTICAS DE LAN MANAGER Y NETWARE

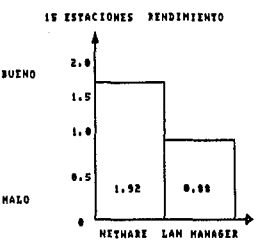
La siguientes gráficas sacadas de la revista PC-MAGAZINE número 16, año 1989, muestran unas pruebas de carga de acceso al disco duro de un servidor COMPAQ SYSTEMPRO, con dos procesadores 80386 de 330 Mhz, con 15 estaciones, de las cuales siete con procesadores 80286 operando a 12 Mhz, siete con procesadores 80386 operando a 16 Mhz y un máquina 486 operando a 33 Mhz.





EL RENDIMIENTO SE DA EN MEGABITS POR SEGUNDO





Resultando que hay pequeñas variaciones entre el rendimiento de LAN MANAGER 2.0 y NETWARE 3.1; sin embargo manejan la información de forma diferente. LAN MANAGER mueve los datos de forma más eficiente en segmentos grandes, haciéndolo más útil para las comunicaciones de LAN a LAN. NETWARE 386 3.1 sobresale moviendo paquetes pequeños en la red, de manera que la mayoría de las aplicaciones todavía están diseñadas para accesar a un disco duro en una sola red.

COMPARACION FINAL DEL PRIMER PUNTO.

Finalmente podemos concluir en base a este pequeño estudio que a favor de LAN MANAGER, este producto ofrece una tecnología robusta con un excelente precio de entrada y una buena ruta de mejora. Uno puede comenzar por un sistema de cinco usuarios por $995 dólares y mejorar a una red que abarque toda la compañia sin tener que volver a cargar el sistema operativo ni los archivos de datos. A favor de NETWARE, aún con su desventaja en precio, NETWARE es más comercial y reconocido en el medio de computación, probado y respaldado por un amplio sistema de apoyo y muchos productos de terceros fabricantes que no hay para LAN MANAGER.

Se puede decir que si se necesita una red que inmediatamente ejecute aplicaciones críticas en la organización, que cuente más con PCs que con otro tipo de dispositivos como mainframes, minis etc.; que maneje relativamente el mismo procesamiento de texto, correo electrónico, y otras aplicaciones típicas, que tenga poca o ninguna conectividad entre redes en múltiples localidades, y si se espera extenderse a más de cien usuarios, considerar a NETWARE. Si se quiere probar el valor de las redes y gradualmente introducir la tecnología a un grupo de trabajo, si se busca aplicaciones basadas en el cliente/servidor y enlaces de redes de área local esparcidas, entonces el poder de un servidor SQL le dan opción a LAN MANAGER.

SEGUNDO PUNTO :

Un segundo punto a estudiar es el sistema operativo que permite ejecutar las aplicaciones tanto a los SMBDs como a los sistemas operativos para red. Ya que SQL SERVER se desarrolla en el ambiente LAN MANAGER, trabaja en el sistema operativo OS/2 y FORCE como se desarrolla en el ambiente NETWARE, trabaja con el sistema operativo DOS.

Toca efectuar un estudio comparativo de ambos sistemas OS/2 y DOS. Sin duda una ventaja de OS/2 es su sistema de multitareas, es decir puede ejecutar varios procesos simultáneamente, a diferencia de DOS que sólo puede ejecutar un proceso a la vez, tanto como DOS, OS/2 aprovecha las características del procesador 80286 y tiene dos modos de operación: el real que es similar a DOS y el protegido, el cual es el modo multitarea, el archivo config.sys de OS/2 es notablemente más grande que el de DOS, ya que en este archivo es donde se determina si se podrá trabajar en uno o ambos modos. Así como DOS tiene un archivo ejecutable, OS/2 tiene tres startup, os2init y autoexec, el primero viene a ser el equivalente al autoexec.bat de DOS y se ejecuta cuando se entra por primera vez al modo protegido.

os2init.cmd se ejecuta cada vez que se inicia una nueva sesión en modo protegido y autoexec.bat se ejecuta cuando se pasa por primera vez al modo real. El intérprete de comandos de OS/2 es similar a DOS, aunque existen algunas diferencias por ejemplo: En OS/2 se llama cmd.exe mientras que en DOS es command.com, las extensiones de los archivos batch en OS/2 es .cmd en lugar de .bat como DOS, en OS/2 si se interrumpe la ejecución de un archivo batch, no se tiene opción de continuar como la hay en DOS.

DOS por su parte ofrece una administración avanzada de memoria para aumentar al máximo la cantidad de memoria convencional libre, de forma que el usuario tenga más memoria para ejecutar sus programas de aplicación.

La amigabilidad de DOS implica su facilidad de uso, añadiendo utilidades básicas que se necesitan para trabajar cómodamente, incluyendo editores de pantalla completa, programas de recuperación de archivos borrados, pantallas de ayuda para todos los comandos, una interface gráfica (shell), utilidad de búsqueda de archivos, etc. Que OS/2 no presenta en su manejo.

OS/2 como DOS tienen comandos internos y comandos externos, pero son los internos los que residen en el intérprete de comandos y realizan sus actividades por medio de éste. OS/2 maneja un sistema de archivos jerárquico como el de DOS, los programas diseñados para trabajar bajo OS/2 requieren pocas modificaciones para convertirse en aplicaciones listas para LANs.

Sin embargo, las direcciones de 16 bits de OS/2 y su limitación de memoria de 16Mb, restringen el tamaño y la potencia de los programas de LAN MANAGER (LAN MANAGER opera en el modo protegido de OS/2).

La siguiente tabla muestra algunas características únicas de OS/2.

DESPACHO PREFERENCIAL	EN UN MOMENTO DADO QUITA EL USO DEL UCP A UN PROCESO Y SE LO DA A OTRO PROCESO DE MAYOR PRIORIDAD.
BLOQUE DE ARCHIVOS	EVITA LA CONCURRENCIA DE DOS PROGRAMAS EN UN ARCHIVO.
POOLING	SI AL INICIAR UN NUEVO PROCESO, NO TERMINA EL ANTERIOR, DIVIDE SU TIEMPO ATENDIENDO A TODOS LOS PROCESOS ACTIVOS EN ESE MOMENTO.
BIBLIOTECA DE LIGA DINAMICA (DDL)	LOS DDL SON CONJUNTOS DE ARCHIVOS CON EXTENSION .DDL QUE OS/2 PUEDE LLAMAR EN CUALQUIER MOMENTO.
LIGA DINAMICA	PERMITE CORREGIR UNA APLICACION E INCORPORARLA EN LAS .DDL.
API	LLAMADAS DE FUNCION QUE PERMITEN USAR UN DISPOSITIVO O COMUNICACION.

CARACTERISTICAS ESPECIALES DE OS/2

COMPARACION FINAL PARA SEGUNDO PUNTO :

Aportando la siguiente comparación, OS/2 es un sistema operativo complejo, necesita mucho espacio en disco y mucha memoria RAM, para los usuario nuevos es difícil de entenderlo, es difícil de instalarlo, así como manejarlo y su precio es de 160 dólares. Sin embargo, ha mostrado valor en el ambiente de un servidor de red y sin duda el modo multitarea es un punto muy a su favor. A diferencia DOS con un valor de 100 dólares y una gran demanda de usuarios por su flexibilidad o facilidad de uso del sistema, permite una amigabilidad usuario-máquina en su manejo, sus requerimientos mínimos de espacio en disco y RAM, su administración de memoria y su amplio apoyo por parte de software, entre otras características, sigue siendo un sistema operativo digno de competir.

En términos generales la pregunta no tiene que ser ¿qué ofrecen los sistemas operativos?, sino qué se les puede exigir?. Y realmente DOS satisface más del 80% de las necesidades inmediatas que tienen los usuarios para su manejo, mientras que los usuarios de OS/2 deben de tener más conocimiento y mucha paciencia para poder satisfacer estas mismas necesidades.

TERCER PUNTO :

Un tercer punto a evaluar es la tecnología cliente-servidor de SQL SERVER contra la arquitectura servidora de archivos de FORCE. A lo largo de esta tesis hemos podido diferenciar las características de cada una de estas arquitecturas.

Se habla del término cliente-servidor de manera muy general para designar aplicaciones de software basadas en dos o más programas que corren cooperativamente en diferentes computadoras conectadas por medio de una red, esta arquitectura es una forma muy especial de procesamiento distribuído. El procesamiento puede dividirse totalmente en su parte cliente y su parte de servidor.

El surgimiento de esta arquitectura permitió la fabricación de servidores de bases de datos como SQL SERVER. El servidor de base de datos se basa en el estándar SQL, controla el acceso a los datos, se comunica con todas las estaciones a través de la red y automáticamente controla la seguridad e integridad de los datos. Previene la corrupción de los archivos cuando un programa termina de manera anormal y procesa rutinas de recuperación en el caso de desconexión de la estación de trabajo o fallo de algún elemento del hardware.

De aquí, que podemos decir que un servidor de base de datos SQL proporciona una solución basada en el cliente-servidor, así como todos los servicios de SQL. En un medio ambiente de red heterogéneo, el servidor SQL no solamente trata con las solicitudes SQL de los usuarios de redes locales, sino que también actúa como un coordinador de datos con otras bases de datos SQL de la red. Las solicitudes de las estaciones de trabajo basadas en cliente, pueden extenderse para tener acceso a los datos de SQL, almacenados bajo otros ambientes operativos.

Como se sabe la arquitectura servidor de archivo, es una arquitectura tradicional de las redes locales, donde las estaciones de trabajo tienen acceso a los datos de forma transparente en el servidor de archivos, el cual es el que se encarga de almacenar y administrar todos los archivos de la red, cuando la estación de trabajo solicita un archivo compartido el sistema operativo de la red lo recupera de forma transparente en el servidor, pero como esta

arquitectura regularmente existe sólo un servidor, cuando se solicitan y envían muchos archivos a través de la red, se produce un tráfico en la misma y su rendimiento baja considerablemente. Sin embargo, hablando de costos, el manejar un servidor de archivos resulta ser buena opción debido a que sólo necesitamos una PC con una capacidad de memoria y almacenamiento muy grande.

Dos de las ventajas que puede ofrecer la arquitectura cliente-servidor sobre la arquitectura de archivos son:

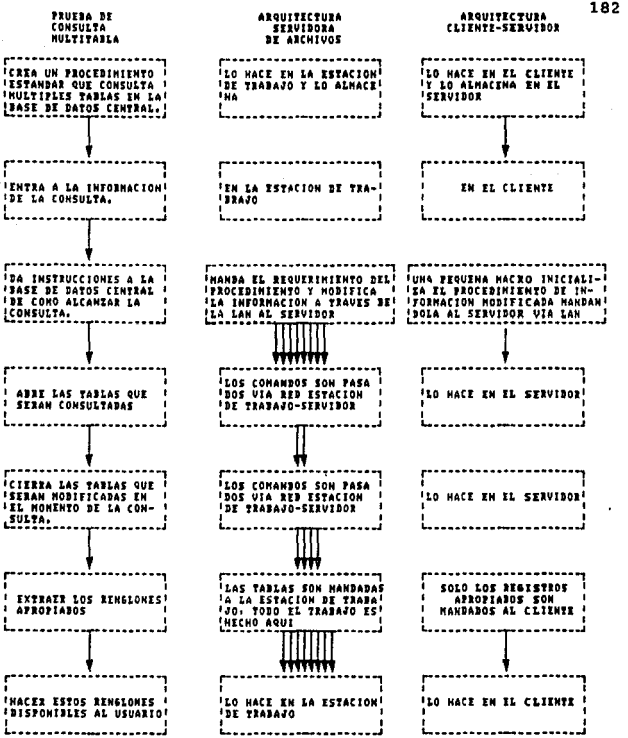
1. Las partes de la aplicación intensiva en manejo de información y cálculo pueden ser llevadas con procesadores rápidos y poderosos, mientras que la interfase de usuario puede hacerse en una PC. Sin embargo, en la arquitectura de archivos todo tipo de aplicación se lleva en la PC que funge como servidor de archivos

2. La reducción de tráfico en la red, ya que el proceso se hace en el lado del servidor y los datos no tienen que viajar a la estación de trabajo como la arquitectura de archivos. Es decir, en una arquitectura servidora de archivo tradicional, en cada requerimiento de datos por un usuario al servidor se envía toda la base de datos a través de la red, de aquí que si muchos usuarios efectuan repetidos requerimientos se crea un continuo tráfico que causa un bajo rendimiento en la misma.

Sin embargo la arquitectura cliente-servidor nos lleva, en ocasiones a tener redes de varios servidores dedicados (de impresora, de comunicación, de correo, etc.), lo cual eleva el costo de la red, y como es relativamente nueva el desarrollo e implantación de sistemas es escaso, por lo que los pocos especialistas en esta arquitectura elevan los precios y se elevan más los costos de la red.

En la gráfica de la página siguiente se muestra un proceso cliente-servidor contra un proceso servidor de archivos, en el lado izquierdo se encuentra cada uno de los pasos del proceso.

El camino de estos pasos es alcanzado por una estación de trabajo y un servidor de archivos contra un cliente basado en SQL y un servidor de base de datos, que son comparados en las dos columnas de la derecha. Las flechas indican la cantidad relativa de tráfico generado en la LAN entre las estaciones y los servidores. Es claro que la primer ventaja de SQL es su habilidad para aislar pesados procesos al servidor dedicado, librando a la estación cliente y sumamente reduciendo en conjunto el tráfico en la red.



COMPARACION FINAL PARA TERCER PUNTO :

Para terminar, podemos decir que la arquitectura cliente-servidor con sus servidores de bases de datos basados en el estándar SQL resuelven muchas de las ventajas y limitaciones de los servidores de archivo, pero hay que tomar en cuenta los costos asociados que implica trabajar con este tipo de arquitectura. Si somos una empresa con suficiente capital para cambiar a esta tecnología y poder satisfacer los requerimientos, tanto económicos, como de mantenimiento y capacitación de personal en una red de área local con tecnología cliente-servidor, sin duda ésta arquitectura es la ideal.

De lo contrario si estamos sujetos a cierto techo financiero o queremos ajustarnos a la tecnología común en redes locales es posible pensar en la tecnología servidora de archivos como una solución a nuestras necesidades.

COMPARACION INTERNA DE SQL SERVER Y FORCE

Una vez hechas las comparaciones externas que influyen en estos SMBDs, entraremos de lleno a mostrar las capacidades de ambos sistemas manejadores para poder establecer ciertas comparaciones que intentarán manifestar que sistema es el más conveniente dada sus capacidades como : aplicaciones, desempeño, costos, instalación entre otras.

La siguiente tabla muestra las diferencias encontradas al estudiar SQL SERVER y FORCE.

PRODUCTO	SQL SERVER	FORCE
PLATAFORMAS	OS/2	DOS
PRECIO	$2995 Y $7995 DOLARES	$600 DOLARES
TIPOS DE CAMPO		
ENTEROS	SI	SI
COMA FLOTANTE	SI	SI
FECHA	SI	SI
CARACTERES	SI	SI
TEXTO	SI	SI
ALIAS	NO	SI
BOOLEANOS	SI	SI
OTROS	SI	NO
LENGUAJES DE AYUDA		
DBASE	NO	SI
SQL	SI	NO
C	SI	SI
ENSAMBLADOR	NO	SI
PROPIETARIO	NO	NO
OTROS	SI	SI
CAPACIDAD MINIMA		
RAM	640KB	256KB
DISCO	30MB	20MB
LIMITES		
NUMERO MAXIMO DE BASES ABIERTAS	ILIMITADO	10
NUMERO MAXIMO DE INDEXACIONES POR BASE	ILIMITADO	7
MAXIMO DE RENGLONES POR COLUMNA	ILIMITADO	100 MILLONES
MAXIMOS DE CAMPO POR RENGLON	ILIMITADO	255
SOPORTE MULTIUSUARIO		
PROTECCION DE ARCHIVOS	SI	SI
HERRAMIENTAS		
GENERADOR DE FORMAS	SI	SI
GENERADOR DE REPORTES	SI	SI

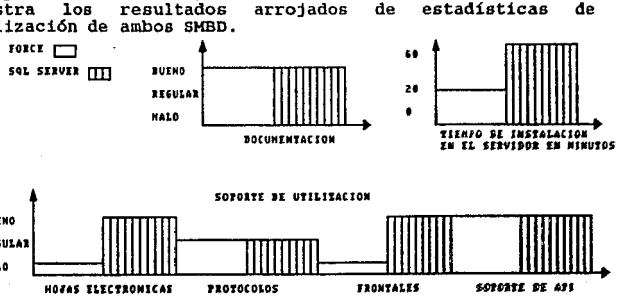
SQL SERVER es sin duda un digno servidor de base de datos que trabaja en la arquitectura cliente-servidor, apoyado por el estándar SQL lo cual lo hace muy poderoso en este tipo de ambiente, a diferencia de FORCE que circula en una tecnología servidora de archivos y no maneja el estándar SQL. Sin embargo el impedimento principal para implantar SQL, es la variación que existe en los estándares SQL.

Las diferencias más claras que se pueden observar en la tabla de la página anterior son que ambos SMBD, como se sabe corren en diferentes plataformas.

El precio de SQL SERVER es mucho mayor al de FORCE (hay que recordar que SQL SERVER también trabaja como servidor de base de datos en una arquitectura cliente-servidor por lo que se incrementa el precio considerablemente). FORCE requiere menor cantidad tanto en memoria, como en disco duro y si se quiere sólo se puede cargar en el servidor de archivos a diferencia de SQL SERVER que es necesario cargar cierta parte en la estación de trabajo. Hay que aclarar que para trabajar idealmente en una arquitectura cliente-servidor, el servidor de base de datos necesita una configuración costosa en cuanto a hardware y software, si se carga SQL SERVER en una red NOVELL se requiere una computadora AT como mínimo, con procesador 80486 y 600 mb en disco duro, para un mejor rendimiento.

SQL SERVER muestra un ilimitado uso, en el manejo de bases de datos o índices entre otros, que lo hace muy potente en el manejo masivo de base de datos, a diferencia de FORCE que sí esta limitado a trabajar con un grado menor de base de datos. Pero de aquí, la flexibilidad de ser un lenguaje con manejo de base de datos, permitiendo que, con un análisis de los requerimientos de la empresa o institución, se puedan hacer programas con pocas bases de datos y manejo de índices pudiendo manejar también grandes masas de datos (hay que recordar que FORCE deja los programas ejecutables pequeños y muy rápidos).

En las siguientes gráficas se muestran los resultados arrojados de los capítulos anteriores con respecto a la documentación y tiempo de instalación de SQL SERVER y FORCE. Notando en ambos puntos una igualdad importante, pero en tiempo de instalación, FORCE es más rápido. Y también se muestra los resultados arrojados de estadísticas de utilización de ambos SMBD.



SQL SERVER por trabajar también como servidor de base de datos soporta interactuar directamente con hojas electrónicas, a diferencia de FORCE que es un compilador que no tiene relación directa con éstas.

Ambos SMBD soportan casi los mismos protocolos por lo que en este punto es difícil establecer un punto a comparar. Hay que recordar que SQL SERVER trabaja primordialmente con el sistema operativo LAN MANAGER y FORCE primordialmente con cualquier versión NOVELL-NETWARE, y cada uno de estos sistemas operativos está buscando la manera de poder comunicarse con otras redes, minis, unix, etc. Por lo que tratan de trabajar o de ser compatibles con cada uno de los protocolos existentes con el fin de crear lo que se entiende como sistemas abiertos. Es decir, donde no existan problemas para comunicarse de un sistema o otro.

SQL SERVER por trabajar en una arquitectura cliente-servidor y fungir como servidor de base de datos sí soporta frontales a diferencia de FORCE que no trabaja en esta arquitectura pero, sin embargo, también puede relacionar sus archivos .dbf y .dbt con otros SMBD para dar apoyo o recibirlo. Algunos de ellos son el DBASE, FOXBASE, FOXPRO y CLIPPER.

SQL SERVER tiene su propio API y un procedimiento almacenado 1/ que ofrecen una ventaja adicional pudiendo incluir sentencias de Transact-SQL que interactuen con otro software en el servidor aparte del propio SMBD, FORCE no tiene su propio API, sin embargo, su fuerte relación con el sistema operativo NOVELL-NETWARE, se mezcla con su API de este sistema y como hemos visto en el capítulo 5, ofrece una mayor ventaja en la red.

FORCE tiene una relación muy formal con el lenguaje de programación C, como pudimos observar en el capítulo 5, FORCE y C interactuando con el sistema operativo NOVELL-NETWARE ofrecen un desarrollo amplio de aplicaciones para la red por medio de sus APIs, sin embargo, SQL SERVER aunque tiene relación con el lenguaje C resultó ser de mayor grado de complejidad efectuar la interacción para desarrollo de aplicaciones en la red.

Un factor interesante es el apoyo del paquete DGE para FORCE, donde los programas pueden importar archivos en formato de gráficas y almacenar estos datos para tener acceso a ellos a través de herramientas de base de datos. SQL SERVER no permite este tipo de ventaja. Pero SQL SERVER soporta interface gráfica a SQL WINDOWS que FORCE no soporta.

1/SON PROCEDIMIENTOS QUE SON EJECUTADOS AUTOMATICAMENTE CUANDO SE HACE UN INTENTO DE MODIFICACION A LOS DATOS QUE ELLOS PROTEGEN

SQL SERVER maneja las consultas multitablas de una manera muy flexible debido a su manejo de base de datos relacionales, a diferencia de FORCE que requiere conocimientos más amplios de indexación y programación para poder establecer este tipo de consultas.

FORCE aparte de enlazarse con lenguaje C, permite enlazarse también con ensamblador, con lo que puede desarrollar programas a un bajo nivel, manejar las interrupciones del BIOS y DOS, crear programas residentes en memoria (TSR) como se vio en el capítulo 5, que SQL SERVER no soporta directamente.

SQL SERVER a diferencia de FORCE, incorpora una ventaja que incluye procedimientos almacenados y disparadores. Los procedimientos almacenados tienen la habilidad para almacenar y ejecutar macros en el servidor, pueden reducir el tráfico en la red local y una rapidez sobre las comunicaciones cliente-servidor.

Todos lo servidores de base de datos promueven elevar la seguridad para permitir poner una base de datos a si misma en una localización segura. Los servidores de base de datos como SQL SERVER con disparadores pueden reforzar la integridad de la base de datos. Los disparadores son procedimientos almacenados invocados principalmente para eventos internos o externos, más que para comandos directos o de programación. Así de esta manera, se puede prevenir automáticamente a un operador de borrar clientes con ciertos archivos o tablas abiertos en sus cuentas. Los disparadores no pueden ser sustituídos para construir provisiones de integridad dentro de cada aplicación, ya que estos disparadores no toman el lugar de validación del dato en la entrada. Pero estos proveen una segunda línea de defensa.

En el capítulo 4 y 5 de esta tesis se efectuaron 5 pruebas, graficando en cada prueba los resultados de una, tres y cinco terminales de trabajo. Estas pruebas fueron :

1. Prueba de búsqueda satisfactoria.

2. Prueba por índice insatisfactoria.

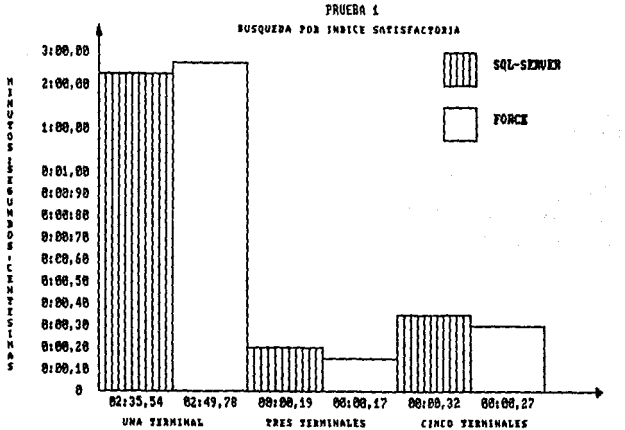
3. Prueba de búsqueda secuencial satisfactoria.

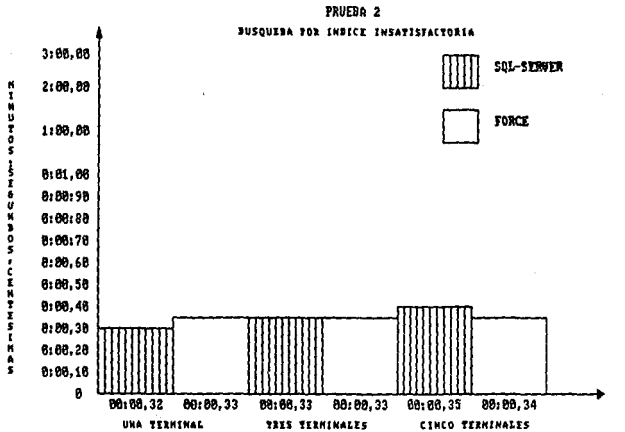
4. Prueba de búsqueda secuencial insatisfactoria.

5. Prueba buscando y agregando registros.

Las cuales son explicadas a detalle en el capítulo 4 de esta tesis.

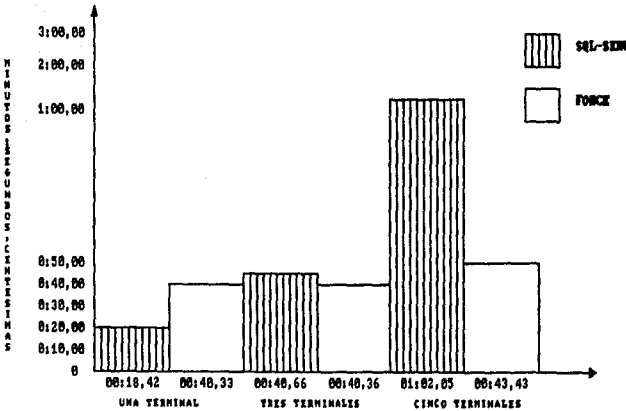
En las gráficas de las páginas siguientes son mostrados los tiempos arrojados en cada prueba, para posteriormente establecer puntos comparativos de estos estudios.





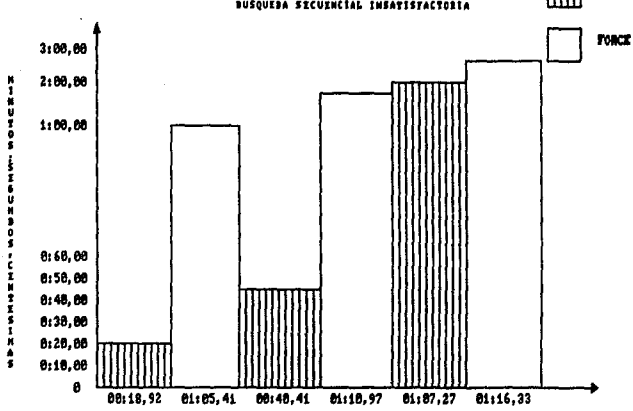
PRUEBA 3

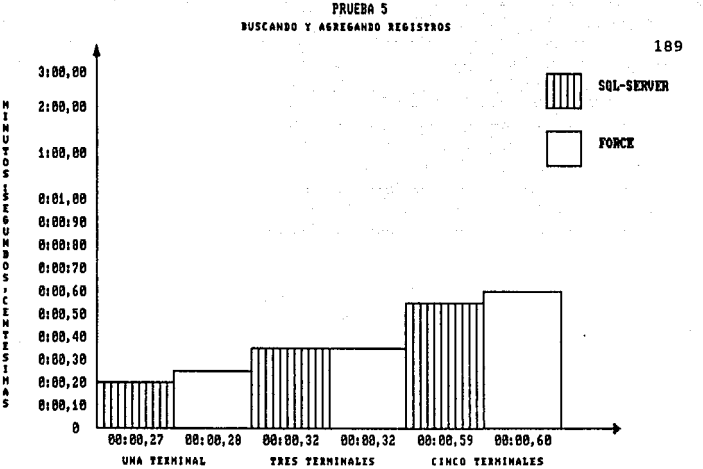
BUSQUEDA SECUENCIAL SATISFACTORIA



PRUEBA 4

BUSQUEDA SECUENCIAL INSATISFACTORIA





En la prueba 1, podemos observar que SQL-SERVER creá más rápido sus archivos de índice, pero para una búsqueda en la base indexada resulta ser más lento mientras más usuarios efectuán la búsqueda.

En la prueba 2, se observa que el tiempo de ambos manejadores en barrer toda la base de datos es mínimo, pero sin embargo, FORCE efectuó tiempos más estables que SQL-SERVER.

En la prueba 3, se encuentra que cuando SQL-SERVER trabaja con un sólo usuario es mucho muy rápida la búsqueda en una base secuencial, pero conforme van aumentando los usuarios SQL-SERVER degrada de una forma considerable. Sin embargo, FORCE mantiene tiempos cortos de diferencia entre más se van agregando usuarios.

En la prueba 4, SQL-SERVER obtuvo en menos tiempo la barrida de toda la base de datos, tanto en una, como en tres y cinco terminales, pero se vuelve a observar que mientras más terminales ocupen la base de datos el tiempo va aumentando considerablemente a diferencia de FORCE que sigue manteniendo tiempos relativamente cortos entre más terminales se agregan.

En la prueba 5, la diferencia entre SQL-SERVER y FORCE es muy mínimo en rangos de centesimas de segundo por lo que para este tipo de problemas, cualquiera de los dos SMBDs es aceptable.

Como ventaja adicional FORCE ofrece un diseño de bases de datos para múltiples usuarios sin tener que invertir más dinero a la red. Se puede decir que SQL SERVER ofrece mayor ventajas para personas que no tienen amplia experiencia en la programación, con su catálogo de sistema estudiado en el capítulo 4, no es necesario tener conocimientos de programación para accesar a SQL SERVER, diferente a esto FORCE por su requerimiento de un buen nivel de programación permite elaborar programas de un contenido más apegado a las necesidades de los usuarios y elaboración de sistemas en un ambiente más amigable inclusive que lo que ofrece un catálogo de sistema como el de SQL SERVER. Por otro lado en SQL SERVER como servidor de base de datos existe una ilimitada asistencia de instalación y configuración pero no hay soporte para diseño de aplicaciones, que FORCE si lo tiene.

COMPARACION INTERNA FINAL :

Finalmente, los manejadores aquí comentados son realmente buenos, ambos tienen sus pros y sus contras pero realmente cumplen con lo necesario para ser calificados de "Excelentes". Pero siempre hay alguno mejor que otro y en este caso para mi punto de vista como manejador de base de datos es FORCE, ya que es barato, no requiere de mucho hardware para su instalación y solamente requiere de buenos programadores inmersos en las necesidades de la empresa y un administrador de red, para que juntos encuentren la solución en la manipulación de información de la manera más óptima.

COMPARACION GLOBAL :

Partiendo de los tres puntos de las comparaciones externas, por el lado de los sistemas operativos podemos observar que éstos van desde las soluciones rápidas para una pequeña red hasta las soluciones estructuradas de transporte con posibilidades de interconexión a otras redes, lo cual nos lleva a pensar que la parte fundamental de todo esto, es que la selección de un sistema operativo es una decisión estratégica muy importante a la vez que compleja, las características que hacen a un sistema operativo la elección perfecta para una red determinada, pueden ser de mínima importancia en otra. Pero lo que no debe de perderse de vista, es la necesidad tarde o temprano de operar con el sistema operativo que ofrezca la mejor amigabilidad usuario-máquina, por lo que para este efecto NOVELL-NETWARE y DOS son la solución.

Por el lado de las arquitecturas en redes locales, la inclinación para mi punto de vista, es la servidora de archivos, sin hechar de menos la alta capacidad en el manejo de información, tanto en velocidad de proceso como en volúmen de datos de la arquitectura cliente-servidor.

Como se puede comprobar, en la arquitectura cliente-servidor, el desarrollo e implantación de sistemas de infromación es alto debido a que los especialistas en esta rama son escasos y con gran demanda, en su instalación el precio sube considerablemente por la necesidad de plataformas de hardware y software más robustas que los servidores de archivos. Y un hecho más a favor de la arquitectura servidora, es que un solo usuario no es suficiente para lograr el desarrollo e implantación de las aplicaciones , ni tampoco puede vigilar el crecimiento en cuanto a hardware y software y realizar el diseño de la base de datos, hacer los ajustes de afinación en los protocolos de comunicación, ni crear las interfaces con otras aplicaciones en servicio, que al fin y al cabo es lo que a futuro pretende la arquitectura cliente-servidor. Recordemos que los analistas y desarrolladores profesionales siguen y seguirán siendo una necesidad en el uso de una red local, y no que nuevas tecnologías lleven a los usuarios a la necesidad de hacerse desarrolladores profesionales.

En términos generales, podemos concluir que cuando se requiere una red, no sólo hay que considerar el tipo o sistema operativo que se va a encargar de administrarla, se debe tener en cuenta las ventajas que puede manejar ciertos SMBD para red ya que al fin y al cabo son los elementos necesarios para poder manipular los datos de la empresa, los cuales requieren todo un proceso de control para que estos sean reales y aplicables a las necesidades de las organizaciones, si tenemos el capital suficiente para comprar equipo que avale un red con arquitectura cliente-servidor, esperamos extendernos en un tiempo no muy lejano y no queremos desarrollar sistemas o aplicaciones, sino más bien ajustarnos a un catálogo de sistemas, y queramos aventurar aprender el ambiente que ofrece OS/2 aunado con LAN MANAGER, el servidor de base de datos y manejador SQL SERVER, es la opción.

Si queremos ajustarnos a lo tradicional y lo más conocido, contamos con un equipo de desarrolladores, programadores y analistas que conozcan las necesidades de la red y de los usuarios de la misma, que no tengamos el capital suficiente para poder comprar servidores o equipo especial como se necesita para trabajar con SQL SERVER, que no esperemos extendernos en un buen tiempo, FORCE como compilador, manejador de base de datos y fácil relación con la red será la opción.

Agregado a esto, si contamos con todos los requerimientos mencionados anteriormente, podemos encontrar la manera de poder trabajar con ambos SMBD en un ambiente de red local con sistema operativo NOVELL-NETWARE, en una arquitectura cliente-servidor y poder encontrar y combinar todas las diferencias, que ofrece el manejo de SQL SERVER y FORCE, sin duda sería una buena opción.

CONCLUSIONES.

Como hemos podido observar a lo largo de este trabajo, el servicio de los Sistemas Manejadores de Base de datos en la industria de la informática y sobre todo en las ya de moda Redes de Area Local, comprende un amplio campo de aplicación de la información dentro de cualquier tipo de industria o institución.

Nos dimos cuenta que a la gente inmersa en la informática nos corresponde por un impulso necesario conocer las diferentes técnicas que ofrecen los manejadores de base de datos en las Redes Locales que hoy en día vienen tomando un gran impulso en el manejo de información, sin importar el giro o actividad de la empresa a la que se pertenezca, ya que somos los profesionales que contamos con los conocimientos y la capacidad suficiente para llevarlo a cabo, tomando en consideración que en la actualidad es un campo que no es aprovechado en su totalidad, por la inovación de este tipo de software.

En la investigación se intento exponer de la manera más flexible el ambiente en como se desarrolla un Sistema manejador de base de datos en redes locales, buscando que el lector encuentre técnicas y estructuras para poder diferenciar los diferentes tipos de sistemas que se presentan en la actualidad y así pueda en determinado momento saber que tipo de manejador es el más adecuado a las necesidades tanto económicas como de información en la red con la que se esta trabajando, o en su defecto en la compra de una Red de Area Local ¿Que tipo de software para manejo de información es el más conveniente?.

Así mismo, se logro establecer una clara comparación de los manejadores estudiados, que en mi opinión son útiles para lograr beneficios adicionales en cuanto a calidad de trabajo se refiere, tales como conocer la mejor alternativa para manejo de datos, de comunicación y para implantar mejores sistemas de información para los usuarios de las Redes de Area Local.

Espero que este trabajo de investigación sea de gran utilidad no solo aquellas personas relacionadas con la Informática, sino también a nuevos egresados para que en base a estos temas, puedan desarrollar tesis futuras más avanzadas y de mayor provecho.

BIBLIOGRAFIA

AUTOR	BURCH,JOHN G. y Grudnitski Gary
TITULO	Diseño de Sistemas de Información.
EDITORIAL	Limusa
EDICION	1era edición
PAIS	México, D.F.
AÑO	1992
PAGINAS	985

AUTOR	CURRID C. CHERYL, Gillet A. Craig
TITULO	Domine el Novell Netware.
EDITORIAL	Macrobit
EDICION	1era edición
PAIS	España
AÑO	1990
PAGINAS	492

AUTOR	DIONYSIOS C. TSICHRITZIS, Frederic H. Lochovsky
TITULO	Data base management systems.
EDITORIAL	Werner Rheinbrdt
EDICION	1era edición
PAIS	U.S.A.
AÑO	1977
PAGINAS	387

AUTOR	GIMENO CARLOS
TITULO	Introducción al Novell Netware.
EDITORIAL	Macrobit
EDICION	1era edición
PAIS	México.
AÑO	1991
PAGINAS	132

AUTOR	GRANILLO ROBERT
TITULO	FORCE 2 Complete command reference with tutorial.
EDITORIAL	Wordware Publishing, Inc.
EDICION	1era edición
PAIS	U.S.A.
AÑO	1991
PAGINAS	420

AUTOR	JAMES MARTIN
TITULO	Principles of data-base management.
EDITORIAL	Prentice Hall, Inc.
EDICION	1era edición
PAIS	U.S.A.
AÑO	1976
PAGINAS	352

AUTOR	JONES EDWARDS
TITULO	Dbase III Plus Guia para usuarios expertos.
EDITORIAL	Mc Graw Hill
EDICION	1era edición
PAIS	España
AÑO	1989
PAGINAS	415

AUTOR	J. VERZELLO ROBERT y Reutter John
TITULO	Procesamiento de datos, conceptos y sistemas.
EDITORIAL	Mc Graw Hill
EDICION	1era edición
PAIS	U.S.A.
AÑO	1983
PAGINAS	579

AUTOR	KURTH F. HENRY, Abrahm Silberschatz
TITULO	Fundamentos de base de datos.
EDITORIAL	Mc Graw Hill
EDICION	1era edición
PAIS	U.S.A.
AÑO	1988
PAGINAS	525

AUTOR	NOVELL INC.
TITULO	Supervisor`s Guide.
EDITORIAL	Novell, Inc.
EDICION	1era edición
PAIS	U.S.A.
AÑO	1988
PAGINAS	110

AUTOR	NOVELL, INC
TITULO	User`s Guide.
EDITORIAL	Novell Inc.
EDICION	1era edición
PAIS	U.S.A.
AÑO	1988
PAGINAS	150

AUTOR	NOVELL, INC
TITULO	Netware getting started.
EDITORIAL	Novell, Inc
EDICION	1era edición
PAIS	U.S.A.
AÑO	1988
PAGINAS	15

AUTOR NOVELL, INC
TITULO Supervisor Reference.
EDITORIAL Novell, Inc
EDICION 1era edición
PAIS U.S.A.
AÑO 1988
PAGINAS 120

AUTOR NOVELL, INC
TITULO Console Reference.
EDITORIAL Novell, Inc
EDICION 1era edición
PAIS U.S.A.
AÑO 1988
PAGINAS 60

AUTOR NOVELL, INC
TITULO Concepts.
EDITORIAL Novell Inc
EDICION Sin edición
PAIS U.S.A.
AÑO 1991
PAGINAS 280

AUTOR RULLO A. THOMAS
TITULO Advances in data comunications management.
EDITORIAL Heyden & son, Inc
EDICION 1era edición
PAIS U.S.A.
AÑO 1980
VOLUMEN Primero
PAGINAS 241

AUTOR R. GROFF JAMES, N. Weinberg Paul
TITULO Aplique SQL.
EDITORIAL Osborne Mc Graw Hill
EDICION 1era edición
PAIS U.S.A.
AÑO 1990
PAGINAS 619

AUTOR SOPHCO, INC
TITULO FORCE ultimate data base compiler
EDITORIAL SOPHCO, INC
EDICION 1era edición
PAIS U.S.A.
AÑO 1990
PAGINAS 1500

AUTOR	SQUIRE ENID
TITULO	Introducción al diseño de sistemas.
EDITORIAL	Fondo Educativo Interamericano
EDICION	1era edición
PAIS	México
AÑO	1984
PAGINAS	345

AUTOR	STALLING WILLIAMS
TITULO	Tutorial Network Technology.
EDITORIAL	Ed. IEEE Computer Society Press
PAIS	U.S.A.
AÑO	1985
PAGINAS	429

AUTOR	WELDON, JAY-LOUISE
TITULO	Data base administration.
EDITORIAL	Plenum Press
EDICION	1era edición
PAIS	U.S.A.
AÑO	1981
PAGINAS	250

AUTOR	WOODWARD JEFF
TITULO	El ABC del Novell Netware.
EDITORIAL	Ventura ediciones, S.A. de C.V.
EDICION	1era edición
PAIS	E.E.U.U
AÑO	1989
PAGINAS	244

AUTOR	W. CHU WESLEY
TITULO	Advances in computer comunications y networking.
EDITORIAL	A Wiley-Interscience Publications
EDICION	cuarta
PAIS	U.S.A.
AÑO	1979
PAGINAS	653

REVISTAS

NOMBRE	PC/TIPS
AÑO	1990
NUMERO	32,35
NOMBRE	PC/TIPS
AÑO	1992
NUMERO	49,51,52
NOMBRE	RED
AÑO	1990
NUMERO	2,5
NOMBRE	RED
AÑO	1991
NUMERO	4,7,9,10,11,12
NOMBRE	RED
AÑO	1992
NUMERO	15,16,20,24
NOMBRE	DBMS
AÑO	1991
NUMERO	7,8
NOMBRE	PC MAGAZINE EN ESPAÑOL
AÑO	1991
NUMERO	2,3
NOMBRE	PC MAGAZINE
AÑO	1989
NUMERO	16