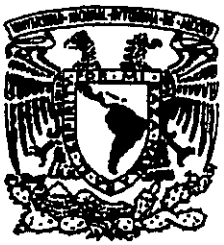
12



UNIVERSIDAD NACIONAL
AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES
CUAUTITLAN

"BASES DE DATOS. AFINACION EN INFORMIX"

TRABAJO DE SEMINARIO

QUE PARA OBTENER EL TITULO DE

LICENCIADO EN INFORMATICA

P R E S E N T A :

JOSE ALFREDO ISIDRO LUNA

ASESOR: MCC. ARACELI NIVON ZAGHI

CUAUTITLAN IZCALLI, EDO. DE MEXICO 200

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN
UNIDAD DE LA ADMINISTRACION ESCOLAR
DEPARTAMENTO DE EXAMENES PROFESIONALES

UNIVERSIDAD NACIONAL AVENIDA DE MEXICO

DR. JUAN ANTONIO MONTARAZ CRESPO
DIRECTOR DE LA FES CUAUTITLAN
P R E S E N T E

ATN: Q. Ma. del Carmen García Mijares
Jefe del Departamento de Exámenes
Profesionales de la FES Cuautitlán

Con base en el art. 51 del Reglamento de Exámenes Profesionales de la FES-Cuautitlán, nos permitimos comunicar a usted que revisamos el Trabajo de Seminario:

Bases de Datos. Afinación en Informix

que presenta el pasante: José Alfredo Isidro Luna

con número de cuenta: 8928896-6 para obtener el título de

Licenciado en Informática

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el EXÁMEN PROFESIONAL correspondiente, otorgamos nuestro VISTO BUENO.

A T E N T A M E N T E
"POR MI RAZA HABLARA EL ESPIRITU"
Cuautitlán Izcalli, Méx. a 8 de Noviembre de 2000

MODULO	PROFESOR	FIRMA
I	Ing. Victor Hugo Arroyo Hernández	
II	MCC. Araceli Nivón Zaghi	
IV	L.C. Carlos Pineda Muñoz	

A mis padres:

Con mucho amor. Porque esta aventura que juntos inicamos hoy la vemos culminada gracias al esfuerzo compartido.

A mis hermanos:

Por brindarme la posibilidad de ser alguien dentro y fuera de la familia gracias al ejemplo mostrado.

A mi hermana:

Por su alegria y paciencia, pero sobre todo por sus apapachos y cuidados.

A Salvador Isidro:

Por su forma de vivir, por el apoyo,
por sus consejos simpre valiosos y por
formar parte nuestra familia.

A mis compañeros de generación

Por todos los momentos agradables
y desagradables que pasamos juntos.

A la Universidad Nacional Autónoma de México

y en especial a la Fes Cuautitlán por
haberme dado una formación como
profesional.

INTRODUCCIÓN

El elemento básico de cualquier sistema de información lo constituye la misma información, la cual desempeña un papel vital en la forma de tomar decisiones y en general en la forma de percibir el mundo. Tal vez sea por esta razón que la información se ha convertido en la nueva fuente de poder, el motor que mueve al mundo.

El factor que le ha dado esta dimensión a la información es su facilidad de acceso. En todo momento se envían, alrededor del mundo, mensajes, noticias e imágenes. El afán por la información de la sociedad actual es incontenible, revistas, libros, bases de datos, reportes especiales, y algunos otros que son ejemplos de una lista en continuo crecimiento.

La forma en como el hombre convierte datos aislados en información ha ido evolucionando a través del tiempo, a la par del avance tecnológico, económico y social. Sin embargo, el gran parteaguas en la tecnología informática fue la invención de la computadora. La forma de producir e interpretar información sufrió una revolución a partir de esta invención, permitiéndonos manejar información cada vez en mayor cantidad y complejidad. Adicionalmente a la invención de la computadora, su fácil disponibilidad ha creado una explosión de información a través de la sociedad.

Con la aparición de la computadora surgieron múltiples productos entorno al nuevo invento. La combinación de hardware/software, cada vez más especializada y mejorada, han tratado de resolver algunos de los mayores problemas del hombre. Actualmente nuestra sociedad se apoya en la tecnología de sistemas de información para trabajar con mayor inteligencia. Las computadoras y los sistemas de información ocupan ahora un sitio especial en las empresas donde facilitan la operación diaria en forma eficiente.

Los sistemas de información proporcionan información tanto de problemas como de oportunidades. Los sistemas de información deben ser capaces, utilizables y confiables. Sin embargo, no es fácil cumplir con todos estos objetivos, en ocasiones se sacrifican unas características por otras dependiendo de los requerimientos de información. Para lograr sus objetivos, los sistemas de información se apoyan en herramientas que les permitan un uso ágil y correcto de la información con la que cuentan. Actualmente el uso de una Base de Datos permite un manejo eficiente de la información.

Otro factor importante dentro de los sistemas de información es el factor humano. Los sistemas de información se desarrollan a través de la gente y para la gente. Sin importar el uso, un sistema de información basado en computadora debe funcionar de manera apropiada, ser fácil de utilizar y adecuarse a la organización para la que fue diseñado. Si un sistema de información ayuda a las personas a trabajar con mayor eficiencia entonces lo utilizarán, de lo contrario lo evitarán.

Es necesario lograr entonces un punto de equilibrio entre las necesidades de información y los esfuerzos necesarios para lograrla. La calidad de la información depende de sus usuarios y es lo que le da valor a los sistemas, sin embargo información de mayor calidad implicará mayores inversiones y esfuerzos. Para saber si la información es de calidad o no, es necesario definir los objetivos y fines que se persiguen con esa información. Lo anterior se logra a través de la definición de estándares que establezcan los parámetros o elementos de control a tomar en cuenta, al momento de evaluar un sistema de información.

En la actualidad, encontramos gran variedad de productos comerciales que administran y garantizan estándares altos en los niveles de la información. Estos productos son conocidos como sistemas manejadores de bases de datos (DBMS, sus siglas en inglés). La forma en cómo un DBMS maneja y da servicios de información dependerá principalmente de las características del propio manejador y del esquema o diseño de la base de datos. Sin embargo, algunos aspectos importantes que afectan el desempeño de una base de datos los podemos resumir en elementos básicos de hardware y software.

Dentro de los elementos de hardware podemos distinguir aspectos como CPU, memoria y disco. Con respecto al software encontramos sistema operativo, software servidor de base de datos y cliente, así como herramientas de apoyo y utilerías.

Por lo que se refiere al hardware, el componente de disco normalmente es el más lento en el proceso de la base de datos, la mayoría del tiempo que ocupa la base de datos en procesar la información ocurre en operaciones de lectura y escritura desde y hacia el disco (I/O). Resulta de suma importancia la forma en como los datos están dispuestos a lo largo de todos los dispositivos de disco con que se cuente, para evitar esperas al tratar de leer o escribir la información. Si los datos están distribuidos de forma desproporcionada sobre los diferentes discos, habrá discos con mayor carga de trabajo mientras que otros discos tendrán actividad mínima.

De acuerdo con esta problemática presentada tan comúnmente en los DBMS que manejan volúmenes considerables de información, en el presente trabajo de tesis se propone un proceso de afinación de bases de datos, cuyo propósito es identificar los recursos críticos tanto de hardware y software y, para que basándose en los objetivos y estándares de rendimiento, se establezca una estrategia a seguir en la administración de los recursos de la base de datos.

Un proceso de afinación de base de datos se hace necesario por alguna de las siguientes razones:

1. Los datos incrementan su cantidad y complejidad.
2. El sistema sufre cambios debido a la naturaleza de su medio ambiente.
3. La base de datos deja de cumplir con algunos de sus objetivos.

OBJETIVOS

Objetivo general

Describir el proceso de afinación y mejoramiento de desempeño en una base de datos.

Objetivos específicos

- Identificar recursos críticos en el desempeño de una base de datos.
- Identificar actividades o eventos que repercutirán en el desempeño de la base de datos.
- Mejorar el desempeño de la base de datos por medio de una utilización óptima de sus recursos.

1. SISTEMAS DE INFORMACIÓN

1.1 Naturaleza de la información

Podemos definir como información al "factor cualitativo que designa la posición de un sistema, y que eventualmente es transmitido por este sistema a otro"[1], es decir todo aquello que pueda ser representado, que se pueda leer o escribir, que sea capaz de ser manipulado y que posea un significado. La información esta representada por símbolos, sin embargo, los símbolos por sí solos no representan información. También tenemos que saber que la información para que pueda ser utilizada en diferentes procesos, la tendremos que tener almacenada en soportes físicos de almacenamiento, tales como papel, cintas o discos.

A pesar de que los datos se encuentran dispersos a nuestro alrededor y forman parte de nuestra atmósfera, en forma de eventos o sucesos, el proceso de transformarlos en información no es fácil y generalmente implica grandes esfuerzos económicos, tecnológicos y humanos.

Sin embargo, un manejo inadecuado de la información puede conducir a una polución informativa. Fenómeno análogo a la contaminación del aire en el que la información al perder sus cualidades, no puede cumplir sus objetivos, llegando incluso a ser más nociva que beneficiosa para sus destinatarios, pues corre el riesgo de basar sus decisiones en algo irreal. Para evitar el peligro de la polución informativa se debe exigir a la información un conjunto de cualidades que mantengan su valor comunicativo, es decir proporcionar un conocimiento.

Las cualidades que debe poseer la información y que hacen de ella un recurso fundamental de las organizaciones y de los individuos, son básicamente: la precisión, oportunidad, plenitud, significado y coherencia. Todas ellas en el grado que exija cada sistema concreto.

- **Precisión**. Es el porcentaje de información correcta sobre la información total del sistema. Una precisión baja lleva a una falta de credibilidad del usuario hacia la información que se le proporciona.
- **Oportunidad**. Se refiere al tiempo transcurrido desde el momento en que se produjo el hecho que originó el dato hasta el momento en que la información se pone a disposición del usuario. Otras veces la oportunidad se mide en función del momento en que el dato

[1] El Pequeño Larousse Ilustrado 1999. Diccionario Enciclopédico. Ed. Larousse. Página 556

tendría que estar disponible, o bien respecto al defase que produjo el proceso por computadora. La oportunidad depende de cada aplicación. En general, el valor de la información va disminuyendo con el transcurso del tiempo e incluso, después de cierto momento, puede llegar a perder totalmente la relevancia que pudiera tener, a excepción de la información histórica.

- **Plenitud**. La información debe de ser completa para poder cumplir sus fines. La plenitud absoluta es imposible de conseguir, y lo que se pretende en los sistemas de información es alcanzar un nivel que se considere suficiente, el cual dependerá de dos factores: de los datos existentes en los sistemas de información y de los que el sistema sea capaz de localizar durante una consulta concreta.
- **Significado**. Tener el máximo contenido semántico posible, es decir debe de ser comprensible e interesante, lo que supone no proporcionar a los usuarios grandes masas de información que por su volumen no pueden ser asimiladas. Un volumen de información justo es condición indispensable para que ésta sea significativa. Cuando se realiza el diseño de un sistema es preciso tener en cuenta que la información suministrada por éste ha de ser sólo la necesaria y suficiente para que se cumplan los fines propuestos.
- **Coherencia**. La información ha de representar lo más fielmente posible el mundo real debido que su origen esta en la vida real.

Todos estos requisitos de la información son necesarios para tenerlos presentes cuando se están haciendo los estudios que llevarán a la implantación de un sistema de información. Hay que buscar el punto de equilibrio necesario para alcanzar los objetivos del sistema a un costo aceptable, ya que cuantas más cualidades reúna la información más se incrementará su precio. Por otro lado, unas cualidades pueden resultar incompatibles con otras; por ejemplo, pretender una gran precisión lleva consigo generalmente una pérdida de oportunidad.

Las exigencias en materia de información que se detectan en los directivos, políticos, científicos para tomar decisiones racionales y para realizar una investigación documentada y coherente se concretan en la necesidad de acceso a datos dispersos. Esta información no puede obtenerse sólo de sistemas internos propios de las organizaciones, sino mediante sistemas externos de tipo público. La atención de estas necesidades resulta factible ante la posibilidad de utilización de los nuevos instrumentos que los avances tecnológicos ponen a nuestro alcance, como son las redes de computadoras y el web.

La industria o mercado de la información esta relacionada con la información considerada como recurso. La información constituye un recurso fundamental que ha de ser utilizado en todos los sectores, convirtiéndose en el elemento esencial para la investigación y la producción, susceptible de ser adquirida en el mercado y creando en torno a ella una importante actividad industrial.

Cuando se analiza la cadena producción-consumo de la información es preciso distinguir un conjunto de actores que participan en el establecimiento de mecanismos de mercado donde esta actividad se desenvuelve:

- Creadores de bancos de datos (productores): Organizaciones públicas o privadas que han creado archivos o bases de datos, y que deciden ofrecer esta información o otras personas o instituciones.
- Distribuidores de bancos de datos (mayoristas): Instituciones que ofrecen la posibilidad de acceder a conjuntos de datos que se consideran de interés para los usuarios, para lo que habrá de disponer del equipo informático necesario.
- Redes de transporte: Redes de telecomunicaciones públicas o privadas que efectúan la transmisión de los datos entre la computadora donde está almacenado el banco de datos y el centro de acceso.
- Centros de acceso a bancos de datos (minoristas): Organizaciones que disponen de equipos conectados con la red de transmisión, mediante la cual pueden tener acceso a uno o varios centros distribuidores de bancos de datos situados en su propio país o en el extranjero.
- Fondos documentales: Organizaciones que se ocupan de suministrar a los usuarios los documentos cuyas referencias bibliográficas se han obtenido en la consulta.

Además de estos actores, han aparecido otros como los agentes o intermediarios de la información, que están teniendo un papel cada día más importante como profesión liberal a la que acuden los usuarios para que les resuelvan globalmente su problema informativo, ayudándolos a definir sus necesidades de información y a obtener esta, por ejemplo las firmas de consultoría que se encargan de analizar cada situación específica y en base a cada uno de los actores listados ofrecen una solución integral.

1.2 *Definición de sistema de información*

La aplicación de las computadoras en las empresas e instituciones comenzó con el tratamiento administrativo de sus datos operacionales; es decir, los que son necesarios para llevar a cabo las tareas de rutina (nómina, contabilidad, etcétera). Sin embargo, la potencia de la computadora va en aumento como para limitarla al simple uso en el trabajo masivo y repetitivo, por lo que la computadora comenzó a intervenir en otros niveles de la empresa, ayudando a la sistematización de las funciones de dirección y constituyendo un elemento activo en el proceso de toma de decisiones.

Toda organización necesita para su funcionamiento que la información se transmita entre sus distintos elementos, y generalmente también desde y hacia el exterior del sistema. Una parte de esta comunicación se realiza por medio de contactos interpersonales entre los empleados, *sistema de información informal*. Pero este tipo de flujo de información, cuando se trata de organismos complejos, se muestra insuficiente y costoso, siendo preciso disponer de un *sistema de información formal*, también llamado organizacional, que integrado en el sistema de orden superior que es el organismo, aporte a esté la información necesaria de forma eficaz y eficiente.

Todo sistema de información formal se diseña a fin de satisfacer las necesidades de información de una organización y está inmerso en ella. El sistema de información ha de tomar los datos de la propia organización y de fuentes externas, y sus resultados han de ser la información que dicha organización necesita para su gestión y toma de decisiones; por otra parte, los directivos de la organización tendrán que marcar los objetivos y directrices para regular el sistema de información.

"Un sistema de información puede ser definido como una colección de personas, procedimientos y equipos diseñados, construidos, operados y mantenidos para recoger, registrar, procesar, almacenar, recuperar y visualizar información"[2]. Las características de un sistema de información pueden agruparse en:

- **Tecnológicas**, que afectan al rendimiento y seguridad del sistema desde el punto de vista del equipo.
- **Funcionales y semánticas**, que se refieren a sí el sistema hace lo que debe de una forma correcta y si es capaz de adaptarse a requisitos cambiantes.

---

[2] Adoración, de Miguel y Mario Piattin. Conceptos y diseño de bases de datos del modelo E/R al modelo relacional. Ed. Addison-Wesley Iberoamericana, S.A. 1993. p. 17

- **Económicas**, que ponen el énfasis en el costo del sistema y en la eficiencia con que responde a los objetivos.
- **Sociales**, son las que tienen un impacto sobre el entorno social en que se desenvuelve el sistema.

El sistema de información es un motor que impulsa la información, haciéndola circular por el organismo, distribuyéndola y aportándola a aquellas áreas donde es necesaria. Para realizar esta función es preciso que el sistema recoja previamente los datos donde son generados y los procese para convertirlos en información útil.

Entre el sistema de información y el organismo donde esta insertado, existe una mutua y estrecha interrelación. La falta de aceptación entre el sistema de información y el organismo es causa del fracaso de muchos sistemas que prometían ser eficientes.

Un sistema de información está constituido por una serie de componentes:

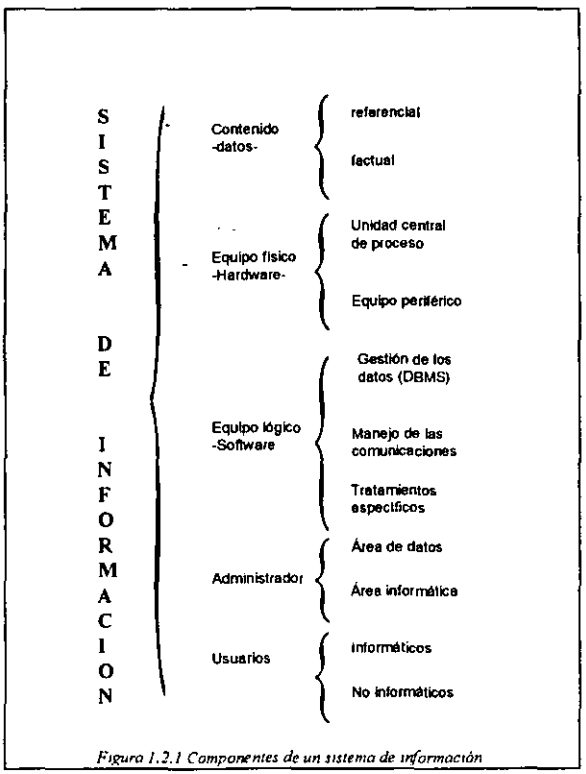


Figura 1.2.1 Componentes de un sistema de información

El contenido del sistema de información es el conjunto de datos, con su correspondiente descripción, estructurados y almacenados en un soporte físico. Los datos habrán de adecuarse a los objetivos que se pretende alcanzar con el sistema.

También existen sistemas de información referencial que contienen referencias bibliográficas de los documentos donde se puede encontrar la información, pero no la información en sí misma, de modo que una vez recuperado el dato es preciso conseguir el documento fuente. En cambio, los sistemas de tipo factual devuelven la información buscada, la cual puede ser directamente utilizada sin necesidad de acudir a nuevos circuitos informativos.

La computadora, que ha de soportar la función de tratamiento o proceso, está integrada por dos subsistemas: equipo físico (*hardware*) y equipo lógico (*software*).

El conjunto de programas, documentación y lenguajes es el equipo lógico que debe manejar los datos (creación, recuperación y actualización), controlar las comunicaciones y dar respuesta a necesidades de tratamientos específicos.

Otro componente fundamental del sistema es el *administrador*, o más bien la unidad de administración, cuya misión es asegurar la calidad y permitir el uso correcto y permanente de los datos almacenados y, en caso de ser necesario, realizar los ajustes necesarios definidos en el componente de control. El administrador no es el propietario de los datos, sino el administrador y custodio de los mismos, cuya responsabilidad se extiende tanto al contenido del sistema como al área informática.

Finalmente, consideramos como otro componente del sistema a los *usuarios*. La persona o grupo de personas que utilizan el sistema de información. Estos usuarios pueden ser tanto informáticos como usuarios finales. También pueden existir usuarios que no acceden directamente al sistema, pero que obtienen información del mismo.

En toda organización se distinguen tres niveles distintos de gestión (operacional, táctico y estratégico), por lo que el sistema de información estará compuesto por tres subsistemas estructurados jerárquicamente y que se corresponden con cada uno de estos tres niveles (figura 1.2.2).

En el plano operacional, los usuarios necesitan datos puntuales, elementales que describan los sucesos que caracterizan las actividades de la organización, por lo que este subsistema de información será muy voluminoso. A partir de este subsistema, mediante un proceso de elaboración adecuado se podrán obtener los datos necesarios para el funcionamiento de los otros dos subsistemas, cuyos usuarios tienen unas exigencias

totalmente distintas, y para los que tal volumen de información no solamente sería inadecuado, sino también inoperante y contraproducente.

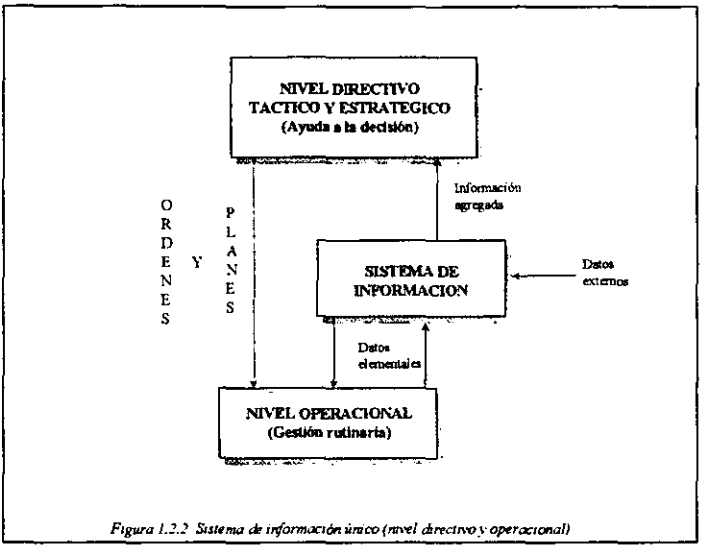


Figura 1.2.2 Sistema de información único (nivel directivo y operacional)

Dentro de los tres niveles de gestión, se puede observar que mientras la información se transmite en sentido ascendente, las órdenes y planes se mueven en sentido descendente (Figura 1.2.3).

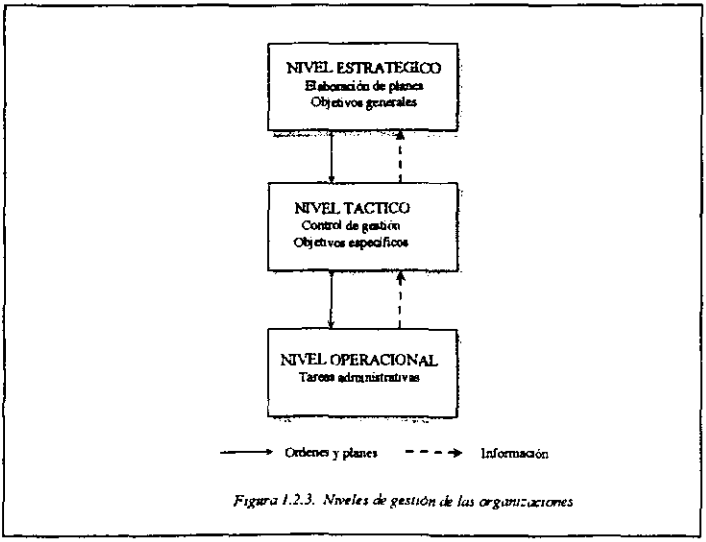


Figura 1.2.3. Niveles de gestión de las organizaciones

En un principio se atendieron las necesidades de información propias del nivel administrativo, desarrollándose aplicaciones distintas y específicas que facilitan las tareas de rutina. La información para la ayuda a la decisión en esta primera etapa se elaboraba manualmente y, a veces, por programas diseñados específicamente para resolver necesidades concretas y puntuales. Posteriormente, y ante los graves problemas a que daba lugar este planteamiento, se vio la necesidad de buscar nuevas soluciones, surgiendo la idea de utilizar una base común de datos que incorporan sin redundancias indeseables la información necesaria para las distintas funciones. Con este enfoque se trata de disponer de un sistema de información integrado capaz de dar respuesta tanto a las necesidades de gestión como de decisión.

En los sistemas diseñados para servir de soporte a la toma de decisiones dirigidos a los directivos de la empresa, uno de los componentes principales es una *base de datos*.

Los sistemas de información son desarrollados con propósitos diferentes dependiendo de las necesidades de información, entre ellos:

- *Sistemas para el procesamiento de transacciones (TPS)*

  Son sistemas orientados hacia operaciones. Tienen como finalidad mejorar las actividades rutinarias de una empresa y de las que depende toda la organización. Una transacción es cualquier suceso o actividad que afecta a toda la organización. Las transacciones más comunes incluyen facturación, pago a empleados, y depósito de cheques. Aunque los tipos de transacciones cambian en cada organización, la mayor parte de las compañías procesan transacciones como una mayor parte de sus actividades cotidianas.

  El procesamiento de transacciones es el conjunto de procedimientos para el manejo de éstas. Todas las actividades de transacciones forman parte del nivel operacional de cualquier organización. El estudio de un grupo de organizaciones muestra la existencia de características similares entre ellas:

  - ✓ Gran volumen de transacciones.
  - ✓ Gran similitud entre las transacciones.
  - ✓ Los procedimientos para el procesamiento de transacciones están bien comprendidos y se pueden describir a detalle.
  - ✓ Existen muy pocas excepciones a los procedimientos normales.

  Estas características permiten establecer rutinas para el manejo de transacciones. Las rutinas describen que buscar en cada transacción, los pasos y procedimientos a seguir y

lo que debe hacerse en caso de que se presente un error. Los procedimientos para el proceso de transacciones se denominan procedimientos de operación estándar.

Las rutinas asociadas con transacciones bancarias caracterizan el empleo de procedimientos de operación estándar para el manejo de depósitos y retiros, pago de cheques y otros procesos. El gran volumen de transacciones precisas asociado con el nivel operativo de una organización junto con la capacidad de los administradores para desarrollar procedimientos específicos para manejarlos, conduce con bastante frecuencia a la implantación de ayuda asistida por computadora. Los sistemas de procesamiento de transacciones brindan velocidad y exactitud; además se pueden programar para seguir rutinas sin ninguna variación.

- *Sistemas de información administrativa (MIS)*

  Los sistemas de información administrativa ayudan a los directivos a tomar decisiones y resolver problemas de tipo recurrente o periódico. Los directivos recurren a los datos almacenados como consecuencia del procesamiento de las transacciones, pero también emplean otra información externa.

  En cualquier organización se deben tomar decisiones sobre muchos asuntos que se presentan con regularidad y para hacerlo se requiere de cierta información. Dado que los procesos de decisión están claramente definidos, entonces se puede identificar la información necesaria para formular las decisiones. Se pueden desarrollar sistemas de información para que, en forma periódica, preparen reportes para el soporte a decisiones. Cada vez que se necesita la información, ésta se prepara y presenta en una forma y formato diseñado con anterioridad.

  Con frecuencia, los especialistas en sistemas de información describen las decisiones apoyadas por estos sistemas como decisiones estructuradas. El aspecto estructurado se refiere al hecho de que los administradores conozcan de antemano los factores que deben tenerse en cuenta para la toma de decisiones así como las variables con influencia más significativa sobre el resultado de una decisión. A su vez, los analistas de sistemas desarrollan reportes bien estructurados que contienen la información necesaria para las decisiones o que indican el estado de las variables más importantes. Estos sistemas presentan reportes basados en las actividades de nivel de transacción. En muchas ocasiones la información proporcionada se combina con otra naturaleza externa, tal como los detalles relacionados con tendencias económicas, demanda y costo de préstamos, y también la tasa de gastos de los consumidores.

- *Sistemas para el soporte de decisiones (DSS)*

  Los sistemas para el soporte de decisiones ayudan a los directivos que deben tomar decisiones no muy estructuradas, también denominadas no estructuradas o decisiones semiestructuradas. Una decisión se considera no estructurada si no existen procedimientos claros para tomarla y tampoco es posible identificar, con anticipación, todos los factores que deben considerarse en la decisión.

  Un factor clave en el uso de estos sistemas es determinar la información necesaria. Conforme se adquiere la información, puede ocurrir que el gerente sé de cuenta de que se necesita más información; es decir, tener información puede conducir a otros requerimientos. En estos casos es imposible diseñar de antemano tanto el formato como el contenido de los reportes del sistema. En consecuencia, los sistemas para el soporte de decisiones deben tener una flexibilidad mayor que la de los demás sistemas de información. El usuario debe ser capaz de solicitar informes definiendo su contenido y especificando la forma para producir la información. El criterio de los directivos tiene un papel importante en la toma de decisiones donde el problema no es estructurado. Los sistemas para el soporte de decisiones ayudan pero no reemplazan el criterio del directivo.

Proceso de Control.

Un sistema es dinámico cuando controla su actuación en función de cómo las salidas cumplen los objetivos marcados; de esta forma, el sistema se va adecuando dinámicamente a unas condiciones de entorno que, en el caso más general, son variables en el tiempo.

El *control* del sistema puede realizarse por medio de mecanismos internos, por mecanismos situados en el entorno o por ambos, aunque esta distinción tiene un alto grado de subjetividad, ya que siempre se podrán ampliar los límites del sistema haciendo que los elementos que llevan a cabo la función reguladora estén comprendidos en el mismo, dependiendo de cuál haya sido la definición y delimitación del sistema. Los sistemas dinámicos están en interacción con el entorno de forma que las entradas y los procesos se van adaptando constantemente para obtener determinadas salidas.

El controlador del sistema, que ejerce funciones de planificación y de gobierno, actúa de acuerdo con la información que recoge de la salida, enviando estímulos a la unidad de entrada y al procesador para conseguir que las salidas respondan a los objetivos del sistema. Debe ser capaz de recibir la información, interpretarla, compararla con los objetivos previstos y emitir los impulsos de control que exija la regulación del sistema.

Las *entradas* del sistema son los elementos que se consumen o transforman en el proceso. Corresponde a la materia prima en los procesos de fabricación, y en el caso de un sistema de información serán los datos.

Las *salidas* son los elementos que se crean en el proceso. Constituyen el producto terminado, la información en forma de reportes, informes o gráficas.

El *procesador* es el lugar donde se efectúa el tratamiento, y comprende todos los elementos que participan en él sin transformarse ni crearse, es decir, excepción de las entradas y las salidas.

Para alcanzar sus objetivos, los sistemas interaccionan con su medio ambiente, el cual esta formado por todos los objetos que se encuentran fuera de las fronteras de los sistemas. El elemento de *control* está relacionado con la naturaleza de los sistemas. Los sistemas trabajan mejor cuando operan dentro de niveles de desempeño tolerables. Todos los sistemas tienen niveles aceptables de desempeño, denominados *estándares* y contra los que se comparan los niveles de desempeño actuales. Siempre deben anotarse las actividades que se encuentran muy por encima o por debajo de los estándares para poder efectuar los ajustes necesarios. La información proporcionada al comparar los resultados con los estándares junto con el proceso de reportar las diferencias a los elementos de control recibe el nombre de retroalimentación.

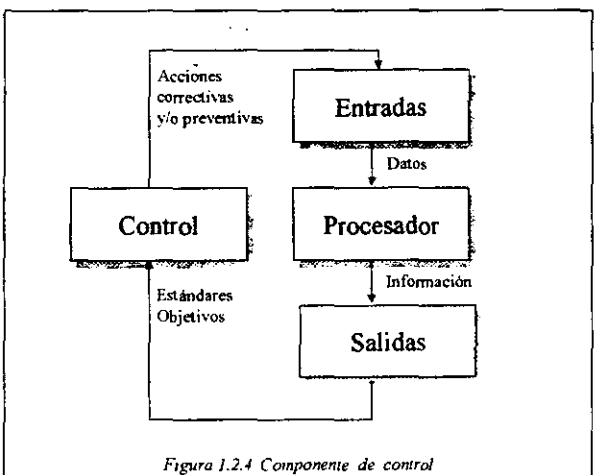


Figura 1.2.4 Componente de control

De acuerdo al esquema anterior, los sistemas de información emplean un componente de control básico consistente en:

1. Un estándar para lograr un desempeño aceptable.
2. Un método para medir el desempeño actual.
3. Un medio para comparar el desempeño actual contra el estándar.
4. Un método de retroalimentación.

Los sistemas deben ajustar sus actividades para mantener niveles aceptables, de lo contrario, tarde o temprano dejan de trabajar. Recibir y evaluar la retroalimentación, permiten al sistema determinar qué tan bien está operando. Para mantener su funcionamiento, deben estar bajo control, es decir, satisfacer ciertos estándares de desempeño.

El presente trabajo se desarrolla dentro de la fase de control ya que es la que determina, en base a los estándares de operación, objetivos y resultados; cuando un proceso de afinación se hace necesario.

Los sistemas de información están formados por subsistemas que incluyen hardware, software, medios de almacenamiento de datos para archivos y bases de datos. El conjunto particular de subsistemas utilizados - equipo específico, programas, archivos y procedimientos - es los que se denomina una aplicación de sistemas de información.

2. MARCO TEÓRICO

2.1 Bases de datos y sistemas manejadores de bases de datos(DBMS)

Si analizáramos la situación de algunos sistemas de información que operan actualmente, encontraríamos una proliferación de archivos para cada uno de los elementos de una determinada aplicación. Los datos se recogen varias veces y se encuentran repetidos en los distintos archivos. Esta redundancia además de malgastar recursos, origina a menudo inconsistencias en los resultados.

Los sistemas de información tradicionales han sido llamados por algunos autores sistemas *orientados hacia el proceso*, debido a que en ellos se pone énfasis en los tratamientos que reciben los datos, los cuales se almacenan en *archivos* diseñados para una determinada aplicación. Las aplicaciones se analizan e implantan con entera independencia unas de otras, y los datos no se transfieren entre ellas, sino que se duplican siempre que los correspondientes trabajos los necesitan.

Este planteamiento produce además de una ocupación inútil de memoria secundaria (discos), un aumento de los tiempos de proceso, al repetirse los mismos controles y operaciones en los distintos archivos. Pero más graves son todavía las inconsistencias que a menudo se presentan en los sistemas, debido a que la actualización de los mismos datos, cuando se encuentran en más de un archivo, no se realiza en forma simultánea en todos los archivos.

Por otra parte, la dependencia de los datos respecto al soporte físico y a los programas, da lugar a una falta de flexibilidad y de adaptabilidad frente a los cambios que repercuten muy negativamente en el rendimiento conjunto del sistema de información.

Los problemas son aún más graves cuando se presentan demandas inesperadas de información o cuando los directivos pretenden tener un verdadero sistema de información orientado a la toma de decisiones, lo que es inalcanzable con estas aplicaciones totalmente inoperantes fuera del contexto para el que fueron concebidas.

El típico sistema orientado hacia el proceso está apoyado por un sistema operativo convencional. Los registros se almacenan en archivos permanentes y se escribe un número de diferentes programas aplicativos para extraer registros y/o añadir registros a los archivos apropiados. Este tipo de sistema tiene un número de desventajas importante.

- **Redundancia e inconsistencia de los datos**. Puesto que los archivos son creados por distintos programas, es probable que los archivos tengan diferentes formatos y los

programas pueden estar duplicados en varios archivos. Esta redundancia aumenta los costos de almacenamiento y acceso. Además, puede llevar a inconsistencia de los datos si las diversas copias de los mismos datos no concuerdan entre sí.

- **Dificultad de acceso a los datos**. Se depende totalmente de la forma en como el archivo esta organizado, por lo que el programador debe saber la estructura de almacenamiento para poder leerlo. Lo que implica un mayor esfuerzo para los programadores.
- **Aislamiento de los datos**. Puesto que los datos están repartidos en varios archivos, y estos pueden tener diferentes formatos, es difícil escribir nuevos programas de aplicación para obtener los datos apropiados.
- **Anomalías del acceso concurrente**. La interacción de actualizaciones concurrentes puede dar por resultado datos inconsistentes. Para prevenir esta posibilidad, debe mantenerse alguna forma de supervisión en el sistema. Puesto que se puede acceder a los datos por medio de diversos programas de aplicación que no han sido previamente coordinados, esta supervisión es muy difícil de proporcionar.
- **Problemas de seguridad**. Puesto que los programas de aplicación se añaden al sistema de una forma precisa, es difícil implantar restricciones de seguridad que controlen el acceso a la información.
- **Problemas de integridad**. Los valores de datos almacenados en la base de datos deben satisfacer ciertos tipos de restricciones de consistencia. Estas restricciones se hacen cumplir en el sistema añadiendo códigos apropiados en los diversos programas de aplicación. Sin embargo, cuando se añaden restricciones nuevas, es difícil cambiar los programas para hacerlos cumplir.

De este análisis se dedujo claramente la necesidad de una administración más racional del conjunto de datos, surgiendo así un nuevo enfoque que se apoya sobre una *base de datos* en la cual los datos son recogidos y almacenados una sola vez, con independencia de los procesos. Los datos se organizan y mantienen en un conjunto estructurado que no esta diseñado para una aplicación concreta, sino que por el contrario, tiende a satisfacer las necesidades de información de toda una organización.

Estos sistema son *orientados hacia los datos* y van sustituyendo a los sistemas orientados hacia el proceso, que por su poca fiabilidad, falta de adecuación a la realidad y mal asegurada confidencialidad han ido perdiendo de forma progresiva la confianza de los usuarios. Las ventajas de los sistemas de bases de datos son las siguientes:

- **Independencia de los datos respecto a los tratamientos**. La mutua independencia de datos y tratamientos lleva a que un cambio de estos últimos no imponga un nuevo diseño lógico y/o físico de la base de datos. Por otra parte, la inclusión de nuevas informaciones, desaparición de otras, cambios en la estructura física o en los caminos de acceso, no deben obligar a alterar los programas.
- **Flexibilidad**. La flexibilidad que proporciona la independencia de los datos y programas es muy importante para conseguir sin costos excesivos la continúa adaptación del sistema de información a la evolución de las organizaciones.
- **Coherencia de los resultados**. Debido a que la información de la base de datos se recoge y almacena una sola vez, en todos los procesos se utilizan los mismos datos por lo que los resultados son siempre los mismos.
- **Los datos pueden compartirse**. No sólo significa que las aplicaciones existentes pueden compartir los datos de la base de datos, sino que también es factible desarrollar nuevas aplicaciones que operen con los mismos datos almacenados. En otras palabras, las necesidades de datos de las nuevas aplicaciones pueden atenderse sin tener que crear nuevos archivos almacenados.
- **Seguridad**. No todos los usuarios pueden acceder a todos los datos.
- **Sincronización**. Permite a más de un usuario acceder a la misma información.

La introducción de la expresión "*base de datos*" se genera a comienzos de los años setenta, dando como resultado diversas definiciones que en la actualidad y debido a su análisis conceptual se ha llegado a la siguiente:

"Una base de datos es una colección o depósito de datos integrados, con redundancia controlada y con una estructura que refleje las interrelaciones y restricciones existentes en el mundo real; los datos, que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de éstas, y su definición y descripción, únicas para cada tipo de datos, han de estar almacenadas junto con los mismos. Los procedimientos de actualización y recuperación, comunes y bien determinados, habrán de ser capaces de conservar la integridad, seguridad y confidencialidad del conjunto de los datos".[3]

---

[3] Adoración, de Miguel y Mario Piattin. Conceptos y diseño de bases de datos del modelo E/R al modelo relacional. Ed. Addison-Wesley Iberoamericana, S.A. 1993. p. 24

"Un sistema de administración de bases de datos (DBMS database management system) es un sistema cuyo objetivo es proporcionar un entorno que sea a la vez conveniente y eficiente para ser utilizado al extraer y almacenar información de la base de datos. Se compone de la base de datos y de un conjunto de programas para manejarlos".[4]

Los sistemas de bases de datos están diseñados para gestionar grandes bloques de información. La administración de datos implica tanto la definición de las estructuras para el almacenamiento de información como la provisión de mecanismos para la operación de la información. Además, los sistemas de bases de datos deben mantener la seguridad de la información almacenada, pese a caídas del sistema o intentos de accesos no autorizados. Si los datos van a ser compartidos por varios usuarios, el sistema debe evitar posibles resultados anómalos.

La importancia de la información en la mayoría de las organizaciones, y por tanto el valor de la base de datos, ha llevado al desarrollo de una gran cantidad de conceptos y técnicas para la gestión eficiente de los datos, que a continuación se describen:

Un *esquema* de base de datos se especifica por medio de un conjunto de definiciones que se expresan mediante un lenguaje llamado lenguaje de definición de datos (*DDL data definition language*). El resultado de la compilación de las sentencias de DDL es un conjunto de tablas las cuales se almacenan en un archivo especial llamado *diccionario de datos*.

El *diccionario de datos* es un archivo que contiene *metadatos*, es decir datos sobre datos. Este archivo se consulta antes de leer o modificar los datos reales en el sistema de base de datos.

La estructura de almacenamiento y los métodos de acceso usados por los sistemas de bases de datos se especifican por medio de un conjunto de definiciones en un tipo especial de DDL llamado lenguaje de *almacenamiento y definición de datos*. El resultado de la compilación de estas definiciones es un conjunto de instrucciones que especifican los detalles de implementación de los esquemas de bases de datos que normalmente se esconden a los usuarios.

Los niveles de abstracción se aplican no sólo a la definición o estructuración de datos, también se aplican a la manipulación de datos. Por manipulación de datos queremos decir:

- La recuperación de información almacenada en la base de datos.

---

[4] Henry F. Korth. Fundamentos de Bases de datos. Segunda Edición. Ed. McGraw Hill. 1993. p. 20

- La inserción de información nueva en la base de datos.
- La supresión de información de la base de datos.
- La modificación de datos almacenados en la base de datos.

A nivel físico, deben existir algoritmos que permitan acceso eficiente a los datos. En los niveles de abstracción más altos, se pone énfasis en la facilidad de uso. El objetivo es proporcionar una interacción eficiente entre las personas y el sistema.

Un lenguaje de manipulación de datos (*DML data manipulation language*) es un lenguaje que capacita a los usuarios a acceder o manipular datos según estén organizados por el modelo de datos adecuado.

Una *consulta* es una sentencia que solicita la recuperación de información. La porción de un DML que implica recuperación de información se llama *lenguaje de consultas*.

Un *gestor de base de datos* es un módulo de programa que proporciona la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación y consultas hechos al sistema. El gestor de base de datos es responsable de las siguientes tareas:

- **Interacción con el gestor de archivos.** Los datos sin procesar se almacenan en el disco usando el sistema de archivos que normalmente es proporcionado por un sistema operativo convencional. El gestor de base de datos traduce las distintas sentencias DML a comandos del sistema de archivos de bajo nivel. Así, el gestor de base de datos es responsable del almacenamiento, recuperación y actualización de los datos en la base de datos.
- **Implantación de la integridad.** Los valores de los datos que se almacenan en la base de datos deben satisfacer ciertos tipos de restricciones de consistencia. La persona encargada de la administración de la base de datos debe especificar explícitamente estas restricciones, entonces puede determinar si las actualizaciones a la base de datos dan como resultado la violación de la restricción; si así es, se debe tomar la acción apropiada.
- **Implantación de la seguridad.** No todos los usuarios de la base de datos necesitan tener acceso a todo su contenido. Es trabajo del administrador de la base de datos hacer que se cumplan estos requisitos de seguridad.
- **Copia de seguridad y recuperación.** Un sistema de información esta sujeto a fallos, en los cuales se pierde información referente a la base de datos. Es responsabilidad del gestor de la base de datos detectar tales fallos y restaurar la base de datos al estado

que existía antes de ocurrir el fallo. Esto se lleva a cabo normalmente a través de la iniciación de varios procedimientos de copias de seguridad y recuperación.

- **Control de concurrencia.** Cuando varios usuarios actualizan la base de datos concurrentemente, es posible que no se conserve la consistencia de los datos. Controlar la interacción entre los usuarios concurrentes es otra responsabilidad del gestor de la base de datos.

La información contenida en la base de datos se puede consultar en cualquier momento, modificar o borrar por los distintos usuarios de la misma. Hay cuatro tipos de usuarios de sistema de base de datos, diferenciados por la forma en que esperan interaccionar con el sistema.

- **Programadores de aplicaciones.** Interactúan con el sistema por medio de llamadas DML, las cuales están incorporadas en un programa escrito en un lenguaje principal (COBOL, C o Pascal). Estos programas se denominan comúnmente programas de aplicación.
- **Usuarios sofisticados.** Interactúan con el sistema sin escribir programas. En cambio escriben sus preguntas en un lenguaje de consulta de base de datos.
- **Usuarios especializados.** Algunos usuarios sofisticados escriben aplicaciones de base de datos especializadas que no encajan en el marco tradicional de procesamiento de datos; entre estas aplicaciones están los sistemas de diseño ayudados por computadora, sistemas expertos y basados en conocimiento, sistemas que almacenan datos con tipos complejos de datos y sistemas de modelación de entorno.
- **Usuarios ingenuos.** Interactúan con el sistema invocando a uno de los programas de aplicación que se han escrito anteriormente.

Un sistema de base de datos se divide en módulos que tratan cada una de las responsabilidades del sistema general. En la mayoría de los casos, el sistema operativo de la computadora proporciona únicamente los servicios más básicos, y el sistema de la base de datos debe partir de esa base. Así, el diseño de un sistema de base de datos debe incluir la consideración de la interface entre el sistema de base de datos y el sistema operativo.

Los componentes funcionales de un sistema de base de datos que desempeñan un papel muy importante en el rendimiento total incluyen:

- **Gestor de archivos**, el cual gestiona la asignación de espacio en la memoria del disco y de las estructuras de datos usadas para representar información almacenada en disco.

- **Gestor de base de datos**, el cual proporciona la interface entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación y las consultas que se hacen al sistema.
- **Procesador de consultas**, el cual traduce sentencias en un lenguaje de consultas a instrucciones de bajo nivel que entiende el gestor de la base de datos. Además, el procesador de consultas intenta transformar una pregunta del usuario en una forma equivalente pero más eficiente, encontrando así una buena estrategia para ejecutar la consulta.
- **Precompilador de DML**, el cual convierte las sentencias DML incorporadas en un programa de aplicación en llamadas normales a procedimientos en el lenguaje principal. El precompilador debe interactuar con el procesador de consultas para generar el código apropiado.
- **Compilador de DDL**, el cual convierte sentencias en DDL en un conjunto de tablas que contienen *metadatos*, o datos sobre datos.

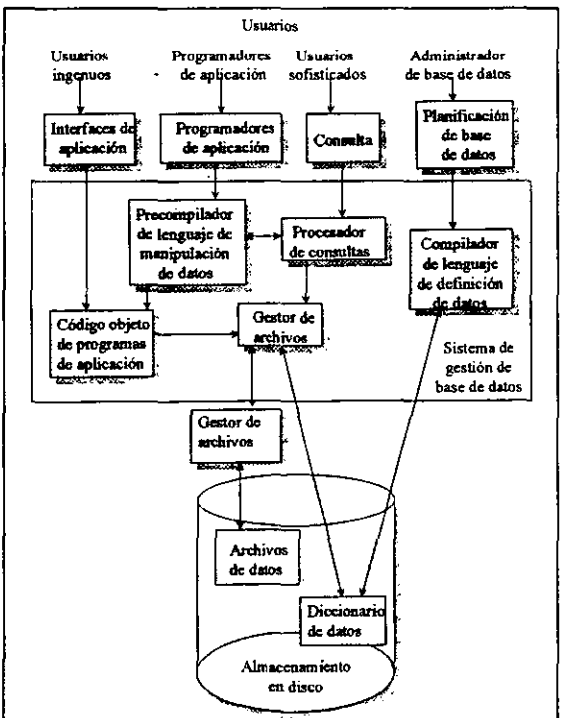


Figura 2.1.1. Estructura del sistema de Base de Datos

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

- **Archivos de datos**, que almacenan la base de datos.
- **Diccionario de datos**, que almacena metadatos sobre la estructura de la base de datos.
- **Indices**, que proporcionan acceso rápido a los elementos de datos que contienen valores determinados..

Un objetivo importante de un sistema de base de datos es proporcionar a los usuarios una visión abstracta de los datos. Es decir, el sistema esconde ciertos detalles de cómo se almacenan y mantienen los datos. Sin embargo, para que el sistema sea manejable, los datos se deben extraer eficientemente. Este requerimiento ha llevado al diseño de estructuras de datos complejas para la representación de datos. Puesto que muchos usuarios de sistemas de bases de datos no tienen experiencia en computadoras, se les esconde la complejidad a través de diversos niveles de abstracción para simplificar su interacción con el sistema.

Las bases de datos cambian a lo largo del tiempo según se añade y se suprime información. La colección de información almacenada en la base de datos, en un determinado momento en el tiempo, se llama una *instancia* de la base de datos. El diseño global de la base de datos se llama *esquema* de la base de datos.

En una base de datos se pueden observar tres niveles de abstracción tales como la estructura lógica o *esquema externo*, estructura lógica global o *esquema conceptual* (o simplemente esquema) y la estructura física o *esquema interno* (Figura 2.1.2).

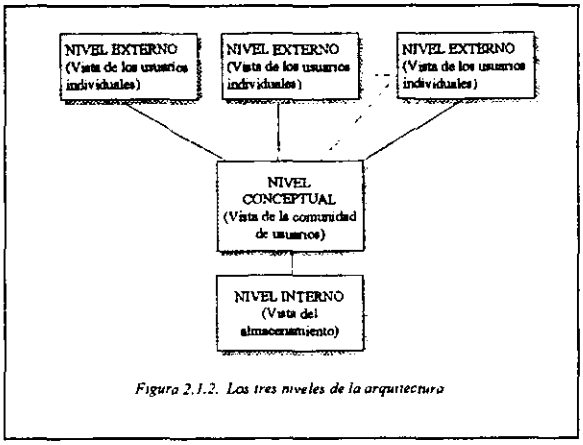


Figura 2.1.2. Los tres niveles de la arquitectura

El *esquema externo* es la visión que de la base de datos tiene cada usuario, en el cuál deberán reflejarse las restricciones de uso y los datos e interrelaciones que necesite el usuario, por lo que habrá tantos esquemas externos como exijan las diferentes aplicaciones y un mismo esquema externo podrá ser utilizado por varias aplicaciones.

El *esquema conceptual* es la visión global de los datos por lo que debe de incluir la descripción de todos los datos así como sus interrelaciones, restricciones de integridad y de confidencialidad. La estabilidad de estos conceptos disminuye en el orden en que son citados. Describe qué datos son realmente almacenados y las relaciones entre ellos. Aquí se describe la base de datos completa en términos de un número pequeño de estructuras relativamente sencillas.

El *esquema interno* es la forma en que se organizan los datos en el almacenamiento físico y aunque éste es muy dependiente de cada DBMS se le deben especificar tres clases de aspectos: *Estrategia de almacenamiento, caminos de acceso y miscelánea,* que incluyen la asignación de espacios de almacenamiento para el conjunto de datos, relaciones que existan entre éstos, estrategia de emplazamiento, tratamiento de los desbordamientos, especificación de claves primaria y secundaria en los caminos de acceso, técnicas de compresión de datos, la translación o correspondencia del esquema interno al esquema conceptual, etcétera.

Finalmente el término *"Afinación de bases de datos"* surge de la necesidad identificada por los desarrolladores de software para bases de datos. Realmente el término se obtiene de la traducción que se hace de la palabra "Tuning" que denota ajuste, afinación o poner a punto. En base a esta definición podemos entender al proceso de afinación de una base de datos como poner a punto la base de datos, es decir corregir, pulir o ajustar esos pequeños o grandes detalles que pudieron haber escapado a la fase de diseño o que han surgido durante la operación del sistema.

2.2 *Modelos básicos de datos*

Un modelo de datos se define como "una colección de herramientas conceptuales para describir datos, relaciones entre ellos, semántica asociada a los datos y restricciones de consistencia"[5]. En general, un modelo de datos se compone de dos elementos:

1. Una notación matemática para expresar datos y relaciones.

[5] Korth F. Henry. Fundamentos de Bases de Datos. Segunda edición. Ed. McGrawHill 1993. p. 6

2. Operaciones sobre los datos que sirven para expresar consultas y otras manipulaciones de datos.

Los diversos modelos de datos que se han propuesto se dividen en dos grupos: modelos lógicos basados en objetos y modelos lógicos basados en registros.

- **Modelos lógicos basados en objetos.** Se usan para describir datos en los niveles conceptual y de visión o externo. Se caracterizan por el hecho de que proporcionan capacidad de estructuración bastante flexible y permiten especificar restricciones de datos explícitamente. Hay muchos modelos existentes y es probable que aparezcan más. Algunos de los más conocidos son:
  - ✓ Modelo entidad-relación.
  - ✓ Modelo orientado a objetos.
- **Modelos basados en registros.** Los modelos lógicos basados en registros se utilizan para describir datos en los niveles conceptual y físico. A diferencia de los modelos de datos basados en objetos, se usan para especificar la estructura lógica global de la base de datos y para proporcionar una descripción a nivel más alto de la implementación.
  Se llaman así porque la base de datos esta estructurada en registros de formato fijo de varios tipos. Cada tipo de registro define un número fijo de campos, o atributos, y cada campo normalmente es de longitud fija. Esto contrasta con muchos de los modelos orientados a objetos en los que los objetos pueden contener otros objetos a un nivel de anidamiento de profundidad arbitraria.
  Los modelos de datos basados en registros no incluyen un mecanismo para la representación directa de código en la base de datos. En cambio, hay lenguajes separados que se asocian con el modelo para expresar consultas y actualizaciones de la base de datos.
  Existen tres modelos de datos que tienen uso común. El conocimiento de qué modelos de datos utilizará un DBMS determinará como debe estructurarse un diseño y las formas en como se representaran las relaciones entre los datos.
  - ✓ Modelo relacional.
  - ✓ Modelo jerárquico.
  - ✓ Modelo de red.

El modelo de datos Entidad-Relación (E-R) se basa en una percepción de un mundo real que consiste en un conjunto de objetos básicos llamados *entidades* y *relaciones* entre

estos objetos. Se desarrollo para facilitar el diseño de bases de datos permitiendo la especificación de esquema empresarial. Este esquema representa la estructura lógica global de las bases de datos.

El modelo orientado a objetos combina los conceptos orientados a objetos con capacidades de base de datos, para así tener el potencial para proveer aplicaciones de bases de datos avanzadas tales como oficinas inteligentes. Los conceptos fundamentales del diseño orientado a objetos son tipos de datos abstractos, herencia e identidad de los objetos. Los conceptos fundamentales de bases de datos son persistencia, transacciones, recuperación, seguridad, consultas y construcción de desempeño.

El modelo jerárquico consiste en una colección de registros que se conectan entre sí por medio de enlaces. Cada registro es un conjunto de campos (atributos), cada uno de los cuales contiene un sólo valor. Un enlace es una asociación entre dos registros exclusivamente. La organización de los registros es en forma de árbol con raíz donde la raíz del árbol es un nodo ficticio. Por lo tanto, una base de datos jerárquica es una colección de árboles formando un bosque.

El modelo de red consta de una colección de registros, los cuales están conectados entre sí por medio de enlaces. Un registro en muchos aspectos es similar a una entidad en el modelo entidad-relación. Cada registro es una colección de campos (atributos), cada uno de los cuales contiene solamente el valor de un dato. Un enlace es una asociación entre dos registros exclusivamente. Así pues, un enlace puede verse como una forma restringida (binaria) de relación en el sentido del modelo E-R.

A continuación se explica más a detalle el modelo relacional debido a su importancia en el desarrollo de las bases de datos actuales por su fundamento y por sus ventajas operacionales, además de que es el modelo sobre el cual realizaremos el proceso de afinación.

2.3 Modelo relacional

El concepto fundamental en el modelo relacional es el conjunto de relaciones, el cual es un subconjunto del producto cartesiano de una lista de dominios. Un *dominio* es simplemente un conjunto de valores, por ejemplo el conjunto de los números enteros representa un dominio.

Una base de datos relacional consiste en una colección de *tablas*, a cada una de las cuales se le asigna un nombre único. Una tabla esta formada por *filas* y *columnas*. Una fila de una tabla representa una relación entre un conjunto de valores. Puesto que una tabla es

una colección de dichas relaciones, hay una estrecha correspondencia entre el concepto de tabla y el concepto matemático de *relación*, del cual toma su nombre el modelo de datos relacional.

Los matemáticos definen una relación como un subconjunto de un producto cartesiano de una lista de dominios. Esto corresponde casi a la definición de tabla. La única diferencia es la asignación de nombres de atributos en las tablas, mientras que los matemáticos se basan en nombres numéricos. Puesto que las tablas son esencialmente relaciones, usaremos los términos matemáticos *relación* y *tupla* en lugar de los términos *tabla* y *fila*. Puesto que una relación es un conjunto de tuplas, usamos la notación matemática $t \in r$ para indicar que la tupla t está en la relación r.

Necesitamos que, para todas las relaciones r, los dominios de todos los atributos r sean atómicos. Un dominio es atómico si los elementos del dominio se consideran unidades indivisibles. Las relaciones tienen las siguientes propiedades:

1. No existen tuplas repetidas.
2. Las tuplas no están ordenadas.
3. Los atributos no están ordenados.
4. Todos los valores de los atributos son atómicos.

Cuando hablamos de una base de datos debemos diferenciar entre el esquema de la base de datos o el diseño lógico de la base de datos y una instancia de la base de datos, que son los datos en la base de datos en un instante de tiempo dado.

El concepto de esquema de una relación corresponde a la noción de definición de tipo en los lenguajes de programación. Una variable de un tipo dado tiene un valor determinado en un instante de tiempo dado.

En general, el esquema de una relación es una lista de atributos y sus correspondientes dominios: *(nombre: cadena, direccion: cadena, edad: entero)*

No es conveniente que todos los usuarios vean el modelo conceptual completo. Las consideraciones de seguridad pueden requerir que se oculten ciertos datos al usuario. Aparte de las cuestiones de seguridad podemos desear crear una colección personalizada de relaciones que se corresponden mejor con una cierta intuición de usuario que el modelo conceptual.

Cualquier relación que no es parte del modelo conceptual, pero se hace visible al usuario como una "relación virtual", se llama *vista*. Es posible tener un gran número de vistas sobre cualquier conjunto de relaciones dado.

El uso de atributos comunes en esquemas de relaciones es una forma de relacionar tuplas de distintas relaciones.

El atributo o conjunto de atributos cuyos valores únicos identifican un conjunto de entidades se le llama *clave.* Cada conjunto entidad tiene una clave, desde que definimos que cada entidad es distinguible de las otras. Conceptualmente, las entidades y relaciones individuales son distintas, pero, desde una perspectiva de base de datos, la diferencia debe expresarse en términos de atributos. El concepto de *superclave* nos permite hacer tales distinciones. Una *superclave* es un conjunto de uno o más atributos que, considerados conjuntamente, nos permiten identificar de forma única a una entidad en el conjunto de entidades.

Sin embargo, el concepto de superclave no es suficiente debido a que una superclave puede contener atributos ajenos. Si K es una superclave, entonces también lo será cualquier subconjunto de K. Normalmente estamos interesados en superclaves para las cuales ningún subconjunto propio es superclave. Dichas superclaves se llaman *claves candidatas.*

El término *clave primaria* o *llave primaria* denota una clave candidata que elige el diseñador de la base de datos como el medio principal de identificar entidades dentro de un conjunto de identidades. Un conjunto de entidades que no tiene atributos suficientes para formar una clave primaria se denomina *conjunto de entidades débil.* Un conjunto de entidades que tiene una clave primaria se denomina *conjunto de entidades fuerte.* Un miembro de un conjunto de entidades fuerte es, por definición, una entidad dominante, mientras que un miembro de un conjunto de entidades débil es una entidad subordinada.

Aunque un conjunto de entidades débil no tiene una clave primaria, debe haber un medio de distinguir entre todas aquellas entidades en el conjunto de entidades que dependen de una entidad fuerte determinada. El *discriminador* de un conjunto de entidades débil es un conjunto de atributos que permite que se haga esta distinción. La clave primaria de un conjunto de entidades débil está formada por la clave primaria del conjunto de entidades fuerte de la que depende su existencia y su discriminador

La mayor parte de los sistemas comerciales de bases de datos relacionales ofrecen un lenguaje de consulta que incluye elementos de los dos enfoques: procedimental y no procedimental. El álgebra relacional es un lenguaje procedimental, mientras que el cálculo relacional de tuplas y el cálculo relacional de dominios son no procedimentales.

Un lenguaje de manipulación de datos completo incluye no sólo un lenguaje de consulta, sino también un lenguaje para la modificación de la base de datos.

2.3.1 ALGEBRA RELACIONAL

Algebra relacional se define como "un lenguaje de consulta procedimental. Consta de un conjunto de operaciones que toman una o dos relaciones como entrada y producen una nueva relación como resultado"[6]. Es decir, un conjunto de operaciones sobre las relaciones.

El álgebra se compone de dos grupos de operadores: los operadores tradicionales de la teoría de conjuntos (unión, intersección, diferencia y producto cartesiano) todos un poco modificados para operar con relaciones; y los operadores especiales de selección, proyección, reunión y división. Las operaciones seleccionar, proyectar y renombrar se llaman operaciones unitarias, ya que operan sobre una relación.

Selección. Selecciona tuplas que satisfagan un predicado dado. Usamos la letra minúscula sigma (σ) para indicar la selección. El predicado aparece como subíndice de σ. Así, para seleccionar aquellas tuplas en una relación, escribimos:

$$\sigma_{Nombre= "Juan"}(empleado)$$

Donde nombre="Juan" forma el predicado que debe cumplirse para presentar las tuplas de la relación empleado.

Proyección. Devuelve su relación argumento con ciertas columnas omitidas y sólo los atributos listados. Puesto que una relación es un conjunto, se eliminan todas las tuplas duplicadas. La proyección se indica por la letra griega pi (∏). Listamos los atributos que queremos que aparezcan en el resultado como subíndice de pi. La relación argumento se escribe a continuación entre paréntesis.

$$\Pi_{nombre,puesto}(empleado)$$

Produto cartesiano. Es una operación binaria, es decir entre dos relaciones, representada por una cruz (X). El producto cartesiano de dos relaciones *A* y *B*, es el conjunto de todas las tuplas *t* tales que *t* es la concatenación de una tupla *a* que pertenece a *A* y una tupla *b* que pertenece a *B*.

Renombrar. Está representado por el símbolo ρ. La expresión ρx(r) devuelve la realción r con el nombre x.

$$\rho_{emp}(empleado)$$

Unión. Es el conjunto de todas las tuplas *t* que pertenecen a la vez a ambas relaciones. Para que una operación de unión sea valida se exigen dos condiciones:

---

[6] Korth F. Henry. Fundamentos de Bases de Datos. Segunda edición. Ed. McGrawHill. 1993. p. 64.

1. Las relaciones r y s deben tener el mismo número de atributos.
2. Los dominios del atributo iésimo de *r* y del atributo iésimo de *s* deben ser los mismos.

Intersección. La intersección de dos relaciones *A* y *B*, es el conjunto de todas las tuplas *t* que pertenecen a la vez a *A* y a *B*.

Diferencia. La diferencia entre dos relaciones *A* y *B*, es el conjunto de todas las tuplas *t* que pertenecen a *A* y no a *B*. Se representa por el símbolo menos (-). La expresión *A-B* da como resultado una relación que contiene aquellas tuplas que están en *A* pero no en *B*.

Las operaciones fundamentales del álgebra relacional son suficientes para expresar cualquier consulta en álgebra relacional, sin embargo, si utilizamos sólo las operaciones fundamentales, algunas consultas comunes son largas de expresar. Por tanto, tenemos operaciones adicionales que no añaden ninguna potencia al álgebra, pero que simplifican consultas comunes.

- **Producto natural.** Es una operación binaria que nos permite combinar ciertas selecciones y un producto cartesiano en una operación. Se representa por el símbolo de producto |X|. La operación producto natural forma un producto cartesiano de sus dos argumentos, realiza una selección forzando la igualdad en aquellos atributos que aparezcan en ambas planificaciones de relaciones y, finalmente, quita las columnas duplicadas.Aunque la definición del producto natural es un poco complicada, la operación se aplica fácilmente
- **División**. La operación división, representada por el símbolo entre (÷), se establece para aquellas consultas que incluyen la frase "para todos". Formalmente, sean *r(R)* y *s(S)* relaciones, y $S \subseteq R$. la relación $r \div s$ es una relación de esquema $R - S$. Una tupla *t* está en $r \div s$ si para cada tupla t_s en *s* existe una tupla t_r en *r* que satisface las dos condiciones siguientes:

  $t_r[S] = t_s[S]$

  $t_r[R-S] = t[R-S]$

  De hecho la operación puede definirse en términos de las operaciones fundamentales. Sea *r(R)* y *s(S)*, con $S \subseteq R$.

  $$r \div s = \prod_{R-S}(S) - \prod_{R-S}((\prod_{R-S}(r) \times S) - r)$$

- **Asignación.** A veces es conveniente escribir una expresión del álgebra por partes usando la asignación a una variable de relación temporal. La operación asignación,

representada por el símbolo ← funciona de forma parecida a la asignación en un lenguaje de programación.

$$temp \leftarrow \Pi_{nombre}(empleado)$$

2.3.2 CÁLCULO RELACIONAL DE TUPLAS

Cuando escribimos una expresión en álgebra relacional, damos una secuencia de procedimientos que genera la respuesta a nuestra consulta. El cálculo relacional de tuplas, en cambio, es un lenguaje de consultas no procedimental. Describe la información deseada sin dar un procedimiento específico para obtener esa información.

Una consulta en el cálculo relacional de tuplas se expresa como $\{t|P(t)\}$

Es decir, el conjunto de todas las tuplas t, tal que el predicado P, es verdadero para t. Siguiendo la notación anterior, usamos $t[A]$ para representar el valor de la tupla t en el atributo A, y usamos $t \in r$ para indicar que la tuplas t está en la relación r.

Las expresiones del cálculo de tuplas se construyen a partir de los elementos siguientes:

- **Variables de tupla**. Cada variable se restringe a variar sobre alguna relación con nombre. Si la variable de tupla t representa a la tupla t, entonces la expresión $t[A]$ representa al componente A de t (en ese instante), donde A es un atributo de la relación sobre la cual varía t.
- **Condiciones de la forma x * y**, donde * es cualquiera de los símbolos =, <, <=, >, o >=, y al menos una de entre x e y es una expresión de la forma $t[A]$ y la otra es una expresión semejante o una constante.
- **Fórmulas bien formadas (FBFs)**. Una FBF se construye a partir de condiciones, operadores booleanos (AND, OR, NOT) y cuantificadores ($\forall$,$\exists$).

El cálculo relacional de tuplas restringido a expresiones seguras es equivalente en poder expresivo al álgebra relacional. Esto quiere decir que para cada expresión en el álgebra relacional existe una expresión equivalente en el cálculo relacional de tuplas, y que cada expresión en el cálculo relacional existe una expresión equivalente en el álgebra relacional.

2.3.3 CÁLCULO RELACIONAL DE DOMINIOS

Existe una segunda forma del cálculo relacional llamada cálculo relacional de dominios. Esta forma usa variables de dominio que toman valores del dominio de un atributo más que valores de una tupla completa. El cálculo relacional de dominios, sin embargo, está íntimamente relacionado con el cálculo relacional de tuplas.

Una expresión en el cálculo relacional de dominios es de la forma

$$\{< x_1, x_2, \ldots, x_n > | P(x_1, x_2, \ldots, x_n)\}$$

Donde x_1, x_2, ..., x_n representan variables de dominio. P representa una fórmula compuesta por átomos, como en el caso del cálculo relacional de tuplas. Un átomo en el cálculo relacional de dominios tiene una de las siguientes formas:

- $<x_1, x_2, \ldots, x_n> \in r$, donde r es una relación en n atributos y x_1, x_2, ..., x_n son variables de dominio o constantes de dominio.
- $x * y$, donde x e y son variables de dominio y * es un operador de comparación (<, <=, =, ≠, >, >=). Es requisito que los atributos x e y tengan dominios que puedan compararse por medio de *.
- $x * c$, donde x es una variable de dominio, * es un operador de comparación y c es una constante en el dominio del atributo para el cual x es una variable de dominio.

2.3.4 MODIFICACIÓN DE LA BASE DE DATOS

Hasta ahora hemos limitado nuestra atención a la extracción de información de la base de datos. No hemos mostrado como añadir, quitar o cambiar información. Las modificaciones a la base de datos se expresan utilizando el operador asignación.

- **Eliminación.** Una solicitud de eliminación se expresa de forma muy parecida a una consulta. Sin embargo, en vez de presentar tuplas al usuario, quitamos las tuplas seleccionadas de la base de datos. Sólo podemos eliminar tuplas completas; no podemos eliminar únicamente valores de determinados atributos. En álgebra relacional, una eliminación se expresa así:

  $$r \leftarrow r - E$$

  Donde r es una relación y E es una consulta del álgebra relacional.
- **Inserción.** Para insertar datos en una relación, bien especificamos la tupla que se va insertar o escribimos una consulta cuyo resultado sea un conjunto de tuplas que se va a insertar. Obviamente, en este último caso, los valores de los atributos para las tuplas insertadas deben ser miembros del dominio de los atributos. Análogamente, las tuplas

insertadas deben ser del mismo orden. En álgebra relacional una inserción se expresa así:

$$r \leftarrow r \cup E$$

Donde r es una relación y E es una expresión del álgebra relacional. La inserción de una sola tupla se expresa haciendo que E sea una relación constante que contiene una tupla.

- **Actualización.** En ciertas ocasiones podemos cambiar un valor en una tupla sin cambiar todos los valores de la tupla. Si hacemos estos cambios usando eliminación e inserción, es posible que no podamos conservar los valores que no queremos cambiar. En lugar de ello, usamos el operador *actualización* δ, que tiene la forma:

$$\delta_{A \leftarrow E}(R)$$

Donde R es el nombre de una relación con atributo A, al cual se le asigna el valor de la expresión E. La expresión E es cualquier expresión aritmética que implica constantes y atributos en la planificación de R.

2.3.5 NORMALIZACIÓN

Es el proceso de simplificar la relación entre los campos de un registro. Por medio de la normalización un conjunto de datos en un registro se reemplazan por varios registros que son más simples y predecibles y, por lo tanto, más manejables. La normalización se lleva a cabo por cuatro razones:

- Estructurar los datos de forma que se puedan representar las relaciones pertinentes entre los datos.
- Permitir la recuperación sencilla de los datos en respuesta a las solicitudes de consultas y reportes.
- Simplificar el mantenimiento de los datos actualizándolos, insertándolos y borrándolos.
- Reducir la necesidad de reestructurar o reorganizar los datos cuando surjan nuevas aplicaciones.

La teoría de la normalización está basada en el concepto de *formas normales*. Se dice que una relación está en una forma normal particular si satisface cierto conjunto específico de restricciones.

Dada una relación R, el atributo Y de R es funcionalmente dependiente del atributo X de R si y sólo si, siempre que dos tuplas de R coincidan en sus valores de X, también coincidan en sus valores de Y. Por ejemplo, una relación *sucursal*, el atributo *ciudad* tiene

una dependencia con el atributo sucursal, ya que ambos valores de atributos coinciden siempre:

Sucursal	Préstamo	Ciudad	Cantidad
200	0001	México	$20,000
123	0002	Querétaro	$123,000
200	0003	México	$235,000
123	0004	Querétaro	$40,000

Relación Sucursal

- **Primera forma normal**. Una relación R esta en primera forma normal (1FN) si y sólo si todos los dominios subyacentes sólo contienen valores atómicos. Es decir, descomponer todos los grupos de datos en registros bidimensionales, un solo valor para cada intersección de renglón y columna.
- **Segunda forma normal**. Una relación R está en segunda forma normal (2FN) si y sólo si está en 1FN y cada atributo no es primo completamente dependiente de la llave primaria. Un atributo es no primo si no participa en la llave primaria.
- **Tercera forma normal**. Una relación R está en tercera forma normal (3FN) si y sólo si está en 2FN, y todo atributo no primo es dependiente no transitivamente de la llave primaria.
- **Forma normal de Boyce/Codd**. Una relación R está en forma normal de Boyce/Codd (FNBC) si y sólo si cada determinante es una llave candidata. Se le llama determinante (funcional) a un atributo, tal vez compuesto, del cual depende funcionalmente en forma completa algún otro atributo.

  Ahora se habla en términos de llaves candidatas, no sólo de llaves primarias. La motivación para introducir la FNBC estriba en que la definición original de la 3FN no maneja satisfactoriamente el caso de una relación que posea dos o más llaves candidatas compuestas y traslapadas.
- **Cuarta forma normal.** Una relación *R* esta en cuarta forma normal (4FN) si y sólo si siempre que exista una dependencia multivaluada (DMV) en *R*, por ejemplo $A \rightarrow\rightarrow B$, todos los atributos de *R* también son funcionalmente dependientes de *A*, es decir, $A \rightarrow X$ para todos los atributos *X* de *R*.

Las dependencias multivaluadas son una generalización de las dependencias funcionales. La declaración de DMV *R[A]→→R[B]* se lee como el atributo *R[B]* es multidependiente del atributo *R[A]* o, en forma equivalente, el atributo *R[A]* multidetermina al atributo *R[B]*.

Dada una relación R con atributos *A*, *B* y *C*, la dependencia multivaluada *R[A]→→R[B]*, se cumple en *R* si y sólo si el conjunto de valores de *B* que corresponden a un par (valor de *A*, valor de *C*) dado en *R* depende tan sólo del valor de *A* y es independiente del valor de *C*. Las DMVs pueden existir sólo si la relación *R* tiene al menos tres atributos.

- **Quinta forma normal.** Una relación R está en quinta forma normal (5FN) si y sólo si toda dependencia de reunión en R está implicada por las llaves candidatas de R.
  La dependencia de reunión generalizada de una lista ordenada de relaciones se forma reemplazando repetidamente las relaciones primera y segunda de la lista por su reunión natural, hasta que la lista entera quede reducida a una sola relación.

2.3.6 LENGUAJE SQL

SQL (Structured Query Language) es el lenguaje de bases de datos relacional estándar. Se compone de varias partes:

- **Lenguaje de definición de datos (DLL).** Proporciona ordenes para definir esquemas de relación, eliminar relaciones, crear índices y modificar esquemas de relación.
- **Lenguaje de manipulación de datos interactivo**. Lenguaje de consulta basado en álgebra relacional y el cálculo relacional de tuplas. También incluye ordenes para insertar, suprimir y modificar tuplas de la base de datos.
- **Lenguaje de manipulación de datos inmerso**. Esta diseñada para utilizar dentro de los lenguajes de programación de propósito general, tales como COBOL, C.
- **Definición de vista.** Ordenes DLL para definir vistas.
- **Autorización**. Especifica derechos de acceso a relaciones y vistas.
- **Integridad**. Ordenes para especificar restricciones de integridad complejas.
- **Control de transacciones**. Especifica el comienzo y final de las transacciones.

La estructura básica de una expresión en SQL consta de tres cláusulas: *select*, *from* y *where*.

- *Select* corresponde a la operación de proyección del álgebra relacional. Lista los atributos que se desean en el resultado de una consulta.

- *From* corresponde a la operación de producto cartesiano. Lista las relaciones que se van a examinar en la evaluación de la expresión.
- *Where* corresponde al predicado de selección del álgebra relacional. Consta de un predicado que implica atributos de las relaciones que aparecen en la cláusula *from*.

  Select $A_1, A_2, \ldots, A_n$

  From $r_1, r_2, \ldots, r_m$

  Where P

SQL forma el producto cartesiano de las relaciones formadas en la cláusula *from*, realiza una selección del álgebra relacional usando el predicado de la cláusula *where* y después proyecta el resultado a los atributos de la cláusula *select*. El resultado de una consulta SQL es una relación.

En SQL se utilizan los conectores lógicos *and, or* y *not*, expresiones aritméticas como operandos de los operadores de comparación. También incluye un operador de asignación *between* para simplificar cláusulas *where* que especifican que un valor que sea menor o igual que un valor dado y mayor o igual que otro valor dado. Análogamente podemos utilizar el operador *not between*.

Una variable de tupla en SQL debe estar asociada con una relación determinada. Las variables de tupla se definen en la cláusula *from* colocándola después del nombre de la relación con la cuál esta asociada, separada por uno o más espacios.

```
SELECT distinct T.nombre, S.saldo
FROM clientes T, cuentas S
```

SQL ofrece la posibilidad de calcular funciones en grupos de tuplas usando la cláusula *group by*. El atributo o atributos dados en la cláusula *group by* se usan para formar grupos. Las tuplas con el mismo valor en todos los atributos en la cláusula group by se colocan en un grupo. El resultado de una función de agregación es un valor único. SQL incluye las siguientes funciones:

- Promedio (avg).
- Minimo (min).
- Máximo (max).
- Total (sum).
- Contar (count).

Hasta ahora solo se ha contemplado operaciones que extraen información de la base de datos sin modificarla, crearla o eliminarla. A continuación se describen las operaciones de modificación de la base de datos:

- **Eliminación**. Una solicitud de eliminación se expresa casi de la misma forma que una consulta. Podemos suprimir solamente tuplas completas. La eliminación se expresa por medio del enunciado *delete*:

  delete r where p,

  donde P representa un predicado y r representa una relación. Las tuplas t en r, para las cuales P(t) es verdadero, son eliminadas de r.

  La operación de eliminación opera sobre una relación. Si se quiere eliminar tuplas de más de una relación, debemos utilizar una orden delete para cada relación.
- **Inserción**. Para insertar datos en una relación especificamos la tupla que se va a insertar o escribimos una consulta cuyo resultado es un conjunto de tuplas que se van a insertar. Los valores de atributos para las tuplas insertadas deben ser miembros del dominio de los atributos. Las tuplas insertadas deben tener el número correcto de atributos. La inserción se expresa por medio del enunciado *insert*:

  insert into r values (atributos)
- **Actualizaciones**. Se lista la relación y las columnas con los nuevos valores para un predicado específico, si no se especifica el predicado se actualiza toda la relación

  Update r

  Set $atributo_1 = valor_1$, $atributo_2 = valor_2$, ... , $atributo_n = valor_n$

  Where P
- **Definición de datos**. El SQL DDL permite la especificación no sólo de un conjunto de relaciones, sino también información sobre cada relación, incluyendo:
  - ✓ El esquema para cada relación.
  - ✓ El dominio de valores asociado con cada atributo.
  - ✓ El conjunto de índices que se va a mantener para cada relación.
  - ✓ La información de seguridad y autorización para cada relación.
  - ✓ Limites de integridad.
  - ✓ La estructura física de almacenamiento de cada relación de disco.

  Una relación en SQL se define usando la orden CREATE TABLE:

  CREATE TABLE $r(A_1D_1, A_2D_2, ..., A_nD_n)$

Donde r es el nombre de la relación, cada A_i es el nombre de un atributo del esquema de la relación r y D_i es el tipo de datos de los valores en el dominio del atributo A_i.

Para eliminar una relación de una base de datos en SQL, usamos la orden DROP TABLE con la cual eliminamos toda la información sobre la relación obtenida de la base de datos:

DROP TABLE r

Donde r es el nombre de la relación

La orden ALTER TABLE se usa para añadir atributos a una relación existente. A todas las tuplas en la relación se les asigna un valor nulo para el nuevo atributo.

ALTER TABLE r ADD A D

Donde r es el nombre de la relación, A es el nombre del atributo que se va a añadir y D es el dominio del atributo añadido.

2.4 Estructura funcional de los archivos

Para realizar la afinación de una base de datos es necesario conocer también los componentes funcionales de las bases de datos. Estos son integrados por diferentes elementos de gestión, a partir de los cuales se determina la forma de accesar la información, de acuerdo con la forma es como se almacenan físicamente los datos, entre estos se encuentran:

- **Gestor de archivos**, que se encarga de asignar el espacio en la memoria del disco y la estructura de datos que se usa para representar la información almacenada en el disco.
- **Gestor de registros intermedio buffer**, que es el responsable de transferir la información entre la memoria del disco y la memoria principal.
- **Analizador sintáctico (parser) de consultas**, que traduce sentencias de un lenguaje de consulta a un lenguaje de nivel más bajo.
- **Selector de estrategias**, que intenta transformar la solicitud del usuario en una forma equivalente pero más eficiente.
- **Gestor de autorización e integridad**, que prueba la satisfacción de las restricciones de integridad y comprueba que el usuario está autorizado para acceder a los datos.
- **Gestor de recuperaciones**, que asegura que la base de datos permanezca en un estado consistente a pesar de que ocurran fallos en el sistema.
- **Controlador de concurrencia**, que asegura que todos los usuarios trabajen en la base de datos sin conflictos entre ellos.

En los sistemas computacionales existen varios tipos de almacenamiento de datos. Estos medios de almacenamiento se clasifican según la velocidad con que pueda tenerse acceso a los datos, el costo y la confiabilidad que tienen:

- **Próximo (caché)**. Es la forma de almacenamiento más rápida y costosa. Su tamaño es limitado y el sistema operativo maneja su utilización.
- **Memoria principal**. Se emplea para los datos disponibles sobre los que se opera. Todos los programas que se ejecutan requieren estar en la memoria principal. Este tipo de memoria es volátil ya que se pierde cuando se corta la energía eléctrica o se apaga el sistema.
- **Disco**. Es el medio principal para el almacenamiento a largo plazo. Los datos deben trasladarse del disco a la memoria principal para poder operar sobre ellos. Comúnmente la base de datos está almacenada en disco. El almacenamiento en disco se denomina de acceso directo porque es posible leer los datos en cualquier orden. Este tipo de almacenamiento es permanente ya que continua aún cuando se apaga el sistema.
- **Cinta**. Se usa principalmente para copias de seguridad y datos de archivo. Aunque la cinta es más barata que el disco, el acceso a los datos es mucho más lento, debido a que la cinta se lee de forma secuencial.

Un disco se compone de un plato que está organizado en pistas concéntricas de datos. La cabeza es un dispositivo que se coloca muy cerca de la superficie del plato. El brazo puede colocarse sobre cualquiera de las pistas. El plato gira a gran velocidad. Para leer o escribir información, el brazo se coloca sobre la pista correcta, y cuando los datos se van a acceder pasan por debajo de la cabeza, realizando la operación de lectura o escritura (I/O).

Puesto que el plato gira a gran velocidad, no se requiere mucho tiempo para que todo el contenido de una pista pase por debajo de la cabeza. Esta cantidad de tiempo se conoce como *tiempo de latencia del disco.* Comparado con el tiempo de latencia, la reposición del brazo tarda mucho tiempo. El tiempo de volver a colocar el brazo, *tiempo de búsqueda*, aumenta conforme se incrementa la distancia a que debe moverse el brazo. Resulta útil almacenar información relacionada en la misma pista o en pistas cercanas físicamente siempre que sea posible para minimizar el tiempo de búsqueda.

Los discos actuales son paquetes de varios platos. El *impulsor* mueve todos los brazos del disco como una unidad. Cada brazo tiene dos cabezas: una para leer y escribir

en la superficie superior del plato que está bajo la cabeza, y una para leer y escribir en la superficie inferior del plato que está sobre la cabeza. El conjunto de pistas sobre las que están colocadas las cabezas forman un *cilindro*. Un cilindro tiene los datos que pueden accederse sin ningún movimiento del impulsor. Es decir, todos los datos del cilindro son accesibles dentro del tiempo de latencia del disco.

Los datos se transfieren entre el disco y la memoria principal en unidades llamadas *bloques*. Un bloque es una secuencia contigua de bytes de una sola pista de un plato. El tamaño de los bloques puede ir desde 512 bytes hasta varios miles de bytes. La forma más sencilla de optimizar el tiempo de acceso al bloque es organizar los bloques en el disco de una forma que corresponda fielmente a la manera en que esperamos que se acceda a los datos. Sin embargo, puede ser costoso mantener esta organización cuando se inserta o elimina información de la base de datos.

Un *archivo* está organizado lógicamente como una secuencia de registros. Estos registros se asignan a bloques de discos. Aunque los bloques son de tamaño fijo determinado por las propiedades físicas del disco y por el sistema operativo, los tamaños de los registros varían.

Una forma de enfocar la asignación de la base de datos a los archivos es utilizar varios archivos y almacenar en un archivo dado solamente registros de una longitud fija. Una alternativa es estructurar los archivos de una forma tal que podamos acomodar registros de varias longitudes diferentes. Los archivos de registros de longitud fija son más fáciles de implementar que los archivos de registros de longitud variable.

Con archivos de longitud fija se conoce el tamaño total en bytes del registro. Se tienen valores para cada tipo de dato utilizado, por ejemplo cada carácter ocupa un byte, un número entero ocupa 4 bytes. Lo más sencillo sería utilizar los primeros *n* bytes para el primer registro, los siguientes *n* bytes para el segundo registro y así sucesivamente. Sin embargo este enfoque sencillo presenta dos problemas:

- Es difícil eliminar un registro de esta estructura. El espacio que ocupa el registro que se va a eliminar debe llenarse con algún otro registro del archivo, o se debe marcar el registro eliminado para ser ignorado.
- Al menos que el tamaño del bloque sea un múltiplo de n, algunos registros pasarán los límites del bloque. Es decir, parte del registro estará almacenada en un bloque y parte en otro. En este caso se requeriría tener acceso a dos bloques para leer o escribir el registro.

Cuando se elimina un registro podríamos mover el registro que le seguía al espacio que ocupaba anteriormente el registro eliminado, y así sucesivamente, hasta que se hubiera movido hasta adelante todos los registros que seguían al registro eliminado. El enfoque anterior requiere mover un gran número de registros. Podría ser mejor simplemente mover el último registro del archivo al espacio que ocupaba el registro eliminado.

No es deseable mover registros para ocupar el espacio que deja libre un registro eliminado, ya que esto requiere tener acceso a bloques adicionales. Puesto que las inserciones suelen ser más frecuentes que las eliminaciones, es aceptable dejar abierto el espacio que ocupaba el registro eliminado y esperar a una inserción posterior antes de volver a utilizar el espacio.

Al principio del archivo asignamos un cierto número de bytes como *encabezado del archivo*. El encabezado contendrá información diversa acerca del archivo, incluyendo la dirección del primer registro cuyo contenido ha sido borrado. El primer registro borrado almacena la dirección del segundo registro borrado y así sucesivamente. Al insertar un nuevo registro utilizamos el registro al que apunta el encabezado; enseguida se cambia para que apunte al siguiente registro disponible. Si no hay espacio disponible añadimos el registro al final del archivo.

La inserción y eliminación en archivos de registros de longitud fija es bastante fácil de implementar porque el espacio que deja disponible un registro eliminado es exactamente el que se necesita para insertar un registro.

Para implementar los registros de longitud variable de manera eficiente en un sistema de archivos, utilizamos uno o más registros de longitud fija para representar un registro de longitud variable. Existen dos técnicas para implementar archivos de registros de longitud variable empleando registros de longitud fija:

- **Espacio reservado**. Si hay una longitud máxima de registro que nunca se excede, podemos utilizar registros de longitud fija de esa longitud. El espacio no utilizado se llena con símbolos especiales como nulos o de fin de registro.
- **Punteros**. El registro de longitud variable se representa por una lista de registros de longitud fija encadenado por medio de punteros.

El método de espacio reservado es útil cuando gran parte de los registros es de longitud cercana al máximo. Si no es así, puede desperdiciarse una cantidad considerable de espacio. Esto nos lleva a considerar el uso del método de punteros. Para representar el

archivo empleando el método de punteros, añadimos un campo que corresponda al puntero. Se utiliza también dos tipos de bloques en el archivo:

- **Bloque de ancla**, que contiene el primer registro de una cadena.
- **Bloque de desbordamiento**, que contiene registros que no son el primer registro de una cadena.

Así todos los registros dentro de un bloque tienen la misma longitud, aunque no todos los archivos del registro tienen la misma longitud.

Como alternativa se considera una estructura compuesta de dos partes, una fija y otra variable:

- El primer registro de la cadena contiene el valor que no se repite y sólo lo encontramos una vez en la estructura.
- Los siguientes registros contienen los campos que se repiten y que no tienen un número fijo en la estructura.

Cuando los datos almacenados rebasan el tamaño de un bloque, encadenamos los bloques utilizando métodos de punteros. Reservamos un espacio fijo al principio de cada bloque como encabezado y lo usamos para almacenar los punteros de la cadena.

Al crecer la cadena por inserción de registros, puede que se necesite añadir nuevos bloques. Al eliminar registros, un bloque podría quedar vacío. En el caso de los registros borrados se puede mantener una cadena de bloques disponibles y volver a utilizarlos para otras cadenas que se extiendan más allá de los bloques que ocupan. Volver a utilizar los bloques ahorraría espacio, sin embargo no es la solución óptima desde el punto de vista de rapidez de acceso.

Una *cubeta* es un conjunto de uno o más bloques encadenados. Si quedará vacío un bloque, sería preferible que volviera a ser utilizado por la cadena que lo contenía anteriormente, en vez de que lo usará otra. Un sistema semejante podría resultar en un gran número de bloques vacíos, pero esto sucederá solamente si la eliminación es más frecuente que la inserción. Puesto que la mayoría de las aplicaciones de base de datos tienen por lo menos tantas inserciones como eliminaciones, es probable que no se desperdicie mucho espacio.

En la práctica no es posible mantener una colocación perfecta de los bloques en el disco sin dejar una cantidad excesiva de espacio. Tarde o temprano las cadenas sobrepasarán su cilindro o cilindros y puede que no exista espacio disponible en cilindros

cercanos. Si la cadena queda tan fragmentada que el rendimiento empieza a disminuir, puede reorganizarse la base de datos.

Un archivo secuencial esta diseñado para procesar de manera eficiente registros en un determinado orden basándose en alguna *clave de búsqueda*. Para permitir una recuperación rápida de registros en el orden de la clave de búsqueda, los registros se encadenan por medio de punteros. El puntero de cada registro apunta al siguiente registro en el orden de la clave de búsqueda. Además, para minimizar el número de accesos a bloque en el procesamiento de archivos secuenciales, los registros se almacenan físicamente en el orden de la clave de búsqueda o tan cerca de éste como sea posible.

Es difícil mantener el orden secuencial físico cuando se insertan o eliminan registros, ya que es muy costoso mover muchos registros como resultado de una sola inserción o eliminación. La eliminación puede manejarse utilizando cadena de punteros, para la inserción se aplican las siguientes reglas:

1. Localizar el registro en el archivo que precede al registro que se va a insertar en el orden de la clave de búsqueda.
2. Si existe algún registro libre dentro del mismo bloque se inserta en el mismo bloque el nuevo registro. Si no se inserta un nuevo bloque de desbordamiento. En cualquier caso se ajustan los punteros para que los registros queden encadenados en el orden de la clave de búsqueda.

Si los registros que se necesita almacenar en bloques de desbordamiento son relativamente pocos, este enfoque funciona bien. Sin embargo, puede llegar a ocurrir que se pierda totalmente la correspondencia entre el orden de la clave de búsqueda y el orden físico, y el procesamiento secuencial llegue a ser mucho menos eficiente. En este punto es preciso reorganizar el archivo para que quede otra vez en orden secuencial físicamente. Estas reorganizaciones son costosas y se hacen en momentos en que la carga del sistema es baja. La frecuencia con que se necesita hacer las reorganizaciones depende de la frecuencia de inserción de registros.

Una forma de reducir el número de accesos a disco es mantener tantos bloques como sea posible en la memoria principal. El objetivo es incrementar al máximo la posibilidad de que, cuando se necesite acceder a un bloque, éste ya esté en la memoria principal y, por tanto, no se requiera acceso al disco.

Debido a que no es posible mantener todos los bloques en la memoria es preciso gestionar la asignación de espacio disponible en la memoria principal para almacenamiento

de bloques. Los registros intermedios (*buffers*) son la parte de la memoria principal disponible para almacenar copias de bloques de disco. El responsable del subsistema para la asignación de espacio en el buffer se denomina *gestor del buffer*. Con el fin de servir adecuadamente al sistema de base de datos, el gestor de buffer debe utilizar técnicas más sofisticadas que las que se emplean en los esquemas comunes de gestión de memoria virtual:

- **Estrategia de reemplazo**. Cuando no hay espacio libre en el buffer, debe sacarse un bloque antes de que pueda escribirse uno nuevo.
- **Bloques sujetos**. Para los casos en los que la base de datos tenga que recuperarse de caídas, es necesario restringir las oportunidades en las que se puede grabar un bloque en el disco. Un bloque al que no se permite que se vuelva a grabar en el disco se dice que está *sujeto*.
- **Salida forzada de bloques**. Existen situaciones en las que es necesario escribir el bloque en el disco aunque no se necesite el espacio que ocupa el buffer.

El objetivo de una estrategia de reemplazo de bloques en el buffer es la minimización de accesos al disco. En programas de propósito general, no es posible predecir con exactitud a qué bloques se va a hacer referencia. Por ello, los sistemas operativos utilizan el patrón de referencias anteriores a bloques para predecir las referencias que se harán en el futuro. La suposición que generalmente se hace es que es probable que se vuelva a hacer referencia a los bloques a los que se hizo referencia recientemente. Por lo tanto, si se debe sustituir un bloque se sustituye aquel al que se hizo referencia menos recientemente. Esto se denomina esquema de reemplazo de bloques LRU.

Un *índice* "es una tabla de números de registros. Estos números llamados punteros estan dispuestos para localizar rápidamente un registro particular mediante una clave. Permite la recuperación de registros en orden de secuencia y permite la inserción de nuevos registros"[7]. Como una alternativa a los índices se puede utilizar técnicas que utilizan funciones de asociatividad. Cada técnica debe evaluarse basándose en:

- **Tiempo de acceso**. El tiempo que se tarda en encontrar un dato determinado utilizando la técnica en cuestión.

[7] Kruglinsk, David. Sistemas de administración de bases de datos. Ed. McGraw Hill. 1984. p 11.

- **Tiempo de inserción**. El tiempo que se tarda en insertar un dato nuevo. Incluye el tiempo que se tarda en encontrar el lugar correcto para el nuevo dato, así como el tiempo utilizado en actualizar la estructura de indexación.
- **Tiempo de eliminación**. El tiempo que se tarda en eliminar un dato. Incluye el tiempo que se tarda en encontrar el dato a borrar, así como el tiempo utilizado en actualizar la estructura de indexación.
- **Espacio extra**. El espacio adicional que ocupa la estructura de indexación. Siempre que la cantidad de espacio no sea muy grande, normalmente merece la pena sacrificar el espacio para lograr una mejora en el rendimiento.

El atributo o conjunto de atributos que se usa para buscar registros en un archivo se llama *clave de búsqueda*.

Un índice permite el acceso aleatorio rápido a los registros de un archivo. Cada estructura de índice está asociada con una clave de búsqueda determinada. Si el archivo está ordenado secuencialmente y elegimos incluir varios índices en diferentes claves de búsqueda, el índice cuya clave de búsqueda especifica el orden secuencial del archivo es el *índice primario*. Los demás se llaman *índices secundarios*. La clave de búsqueda de un índice primario es normalmente la clave primaria. Hay dos tipos de índices que pueden usarse:

- **Indice denso**. Aparece un registro índice para cada valor de la clave de búsqueda en el archivo. El registro contiene el valor de la clave de búsqueda y un puntero al registro.
- **Indice escaso**. Se crean registros índices solamente para algunos de los registros. Para localizar un registro, encontramos el registro menor o igual que el valor de la clave de búsqueda que estamos buscando. Empezamos en el registro al que apunta el registro índice y seguimos los punteros del archivo hasta encontrar el registro deseado.

Generalmente es más rápido localizar un registro con un índice denso que con un índice escaso. Sin embargo, los índices escasos tienen una ventaja sobre los índices densos, requieren menos espacio e imponen menos mantenimiento adicional para inserciones y eliminaciones.

El diseñador del sistema debe lograr un equilibrio entre el tiempo de acceso y el espacio extra. Aunque la decisión con respecto al punto de equilibrio es dependiente de la aplicación específica, resulta conveniente tener un índice escaso con una entrada de índice por bloque. La razón de que este diseño sea un buen criterio es que el costo dominante en

el procesamiento de una solicitud de base de datos es el tiempo que se tarda en llevar un bloque del disco a la memoria principal.

Si un índice es lo bastante pequeño como para guardarlo en memoria principal, el tiempo de búsqueda es corto. Por lo contrario, si el índice es grande como para como para guardarlo en disco, una búsqueda resulta en varias lecturas de bloques del disco.

Para resolver el problema anterior tratamos el índice como trataríamos cualquier otro archivo secuencial, y construimos un índice escaso sobre el índice primario.

Para localizar un registro, primero empleamos una búsqueda binaria en el índice externo para encontrar el registro que corresponda al valor de la clave de búsqueda más grande, menor o igual que el que deseamos. El puntero señala a un bloque del índice interno. Examinamos este bloque hasta que encontramos el registro que tiene el valor clave de búsqueda más grande, menor o igual que el que deseamos. El puntero de este registro apunta al bloque del archivo que contiene el registro que estamos buscando.

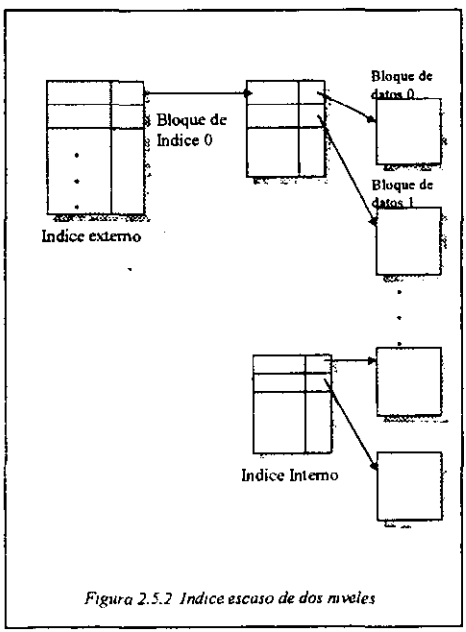


Figura 2.5.2 Indice escaso de dos niveles

Sin importar cual sea la forma de índice que se utilice, se deben actualizar todos los índices cada vez que se inserta o elimina un registro del archivo. El algoritmo para actualizar índices de un solo nivel es el siguiente:

- **Eliminación**. Es necesario buscar el registro que se va a eliminar. Si el registro eliminado era el último que quedaba con ese valor particular en la clave de búsqueda, entonces eliminamos el valor de la clave de búsqueda del índice. Para índices densos eliminamos un valor de la clave de la misma manera que se suprime en un archivo. Para índices escasos, eliminamos un valor de clave sustituyendo su entrada en el índice (sí existe una) por el siguiente valor de la clave de búsqueda. Si el siguiente valor de la clave de búsqueda ya tiene una entrada de índice, eliminamos la entrada.
- **Inserción**. Se hace una búsqueda usando el valor de la clave de búsqueda que aparece en el registro a insertar. Si el índice es denso y el valor de la clave de búsqueda no aparece en el índice, lo inserta. Si el índice es escaso no se necesita hacer ningún cambio en el índice a menos que se cree un nuevo bloque. En este caso el primer valor de la clave de búsqueda que aparezca en el nuevo bloque se inserta en la base de datos.

Los índices secundarios pueden estructurarse de forma diferente a los índices primarios. Los punteros en el índice secundario no señalan directamente al archivo. En vez de ello, cada puntero señala a una cubeta que contiene punteros al archivo. Este enfoque permite almacenar juntos todos los punteros de un valor de clave de búsqueda secundario determinado.

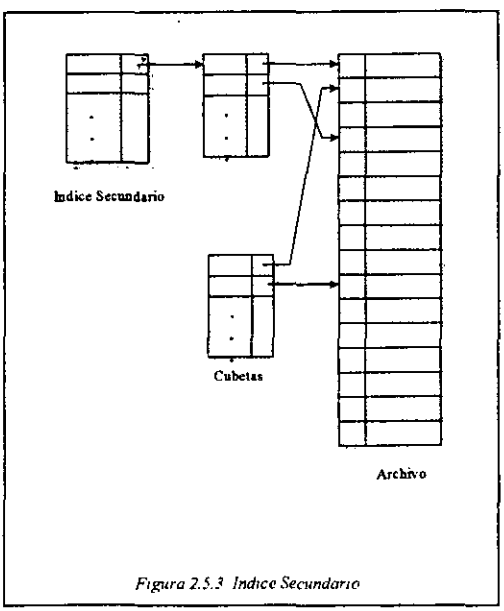


Figura 2.5.3 Índice Secundario

Una revisión secuencial en orden de clave primaria es eficiente porque los registros están almacenados físicamente en un orden que se aproxima al orden de clave primaria. Sin embargo, no podemos almacenar un archivo físicamente ordenado tanto por la clave primaria como por una clave secundaria. Como el orden de clave secundaria y el de clave física son distintos, al leer el archivo en orden de la clave secundaria, es probable que la lectura de cada registro requiera la lectura de un nuevo bloque de disco.

Almacenando punteros en una cubeta se elimina la necesidad de punteros adicionales en los registros mismos y de revisiones secuenciales en orden de clave secundaria.

El índice secundario puede ser denso o escaso. Si es denso, el puntero de cada cubeta individual señala a los registros con el valor de la clave de búsqueda apropiado. Si el índice secundario es escaso, el puntero de cada cubeta individual señala a los registros con valores de la clave de búsqueda en el rango apropiado. En este caso cada entrada de cubeta es un puntero único o bien un registro que consta de dos campos: un valor de la clave de búsqueda y un puntero a algún registro de archivo.

Los índices secundarios mejoran el rendimiento de las consultas que utilizan claves que no son primarias. Sin embargo, implican un gasto extra considerable en la modificación de la base de datos. El diseñador de una base de datos decide que índices secundarios son deseables basándose en una estimación de la frecuencia relativa de consultas y modificaciones.

Un *índice de árbol B+* toma la forma de un árbol equilibrado en el que cualquier camino desde la raíz del árbol hasta una hoja tiene la misma longitud. Todos los nodos del árbol tienen entre [n/2] y n hijos, donde n es fijo para un determinado árbol.

La estructura de árbol B+ impone un cierto gasto extra durante la inserción y eliminación, además de requerir un espacio extra. No obstante, esto es aceptable en el caso de archivos con una alta frecuencia de modificación, ya que se evita el costo de la reorganización del archivo.

Un índice de árbol B+ tiene varios niveles. Contiene hasta n-1 valores de clave de búsqueda K_1, K_2, ..., K_{n-1} y n punteros P_1, P_2, ..., P_n. Los valores de la clave de búsqueda dentro de un nodo se guardan en un determinado orden; así, si $i<j$, entonces $k_i \leq K_j$.

Nodo típico de un árbol B+

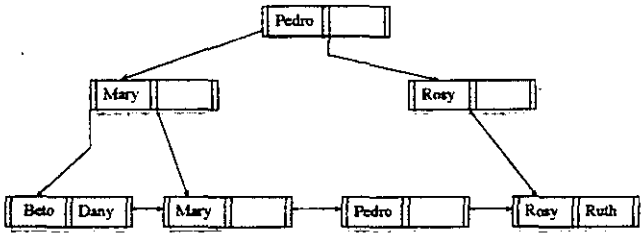


Figura 2.5.4 Ejemplo de una estructura de árbol B+

Consideramos primero la estructura de los nodos hoja. Para $1 \leq i < n$, P_i apunta a cualquier registro del archivo con un valor de clave de búsqueda K_i o a una cubeta de punteros cada uno de los cuales apunta a un registro del archivo con valor de clave de búsqueda K_i. La estructura de cubeta se utiliza solamente si la clave de búsqueda no forma una clave primaria y el archivo no está ordenado en el orden del valor de la clave de búsqueda.

Cada nodo hoja puede tener hasta n-1 valores. El conjunto de nodos hoja en un árbol B+ debe formar un índice denso de manera que cada valor de la clave de búsqueda aparezca en algún nodo hoja.

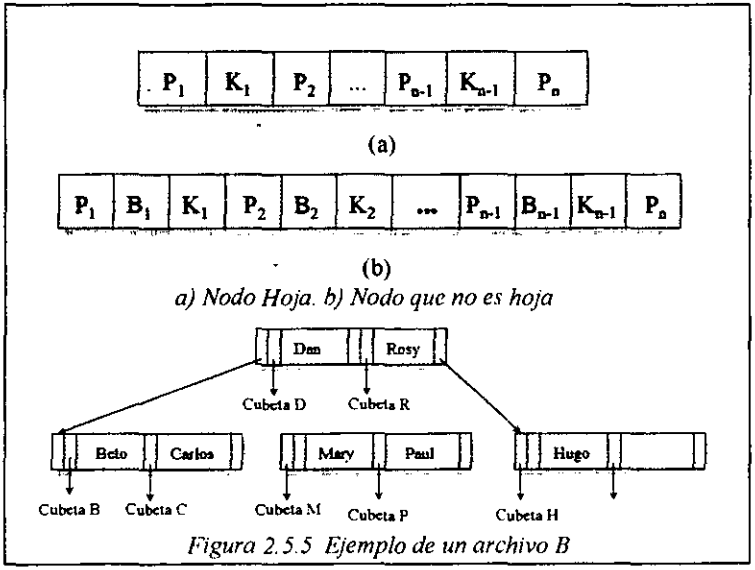
Ya que existe un orden lineal en las hojas basados en los valores de clave de búsqueda que contienen, usamos P_n para encadenar los nodos en orden de clave de búsqueda. Esto permite un procesamiento secuencial del archivo en forma eficiente.

Los nodos del árbol B+ que no son hojas forman un índice (escaso) de varios niveles de los nodos hoja, la estructura de los nodos que no son hoja es la misma que la de los nodos hoja excepto que todos los punteros apuntan a nodos del árbol. Un nodo puede tener hasta n punteros, pero debe tener por lo menos [n/2] punteros.

El requisito de que cada nodo tenga por lo menos [n/2] punteros es obligatorio en todos los niveles del árbol excepto para la raíz. La propiedad de equilibrio de los árboles es la que asegura un buen rendimiento en las búsquedas, inserciones y eliminaciones.

Aunque las operaciones de inserción y eliminación en árboles son complicadas, requieren relativamente pocas operaciones. La velocidad de las operaciones en los árboles B+ es la que hace que se utilicen frecuentemente como estructuras de índices en implementaciones de bases de datos.

Los *índices de árbol B* son similares a los índices de árbol B+. La principal diferencia entre los dos enfoques es que un árbol B elimina el almacenamiento redundante de valores de clave de búsqueda. Un árbol B permite que los valores de clave de búsqueda aparezcan una sola vez. Puesto que no se repiten las claves de búsqueda en el árbol B, podemos almacenar el índice usando menos nodos que en el índice de árbol B+ correspondiente. Sin embargo, debido a que las claves de búsqueda que aparecen en los nodos que no son hoja no aparecen en ningún otro sitio del árbol B, estamos obligados a incluir un campo puntero adicional para cada clave de búsqueda en un nodo que no sea hoja. Estos punteros adicionales apuntan a registros de archivos o cubetas para la clave de búsqueda asociada.



a) Nodo Hoja. b) Nodo que no es hoja

Figura 2.5.5 Ejemplo de un archivo B

Los árboles B ofrecen una ventaja sobre los árboles B+ aparte de eliminar el almacenamiento redundante de claves de búsqueda. En una búsqueda en un árbol B+, siempre es necesario atravesar un camino desde la raíz del árbol hasta algún nodo hoja. En cambio, en un árbol B, a veces es posible encontrar el valor deseado antes de leer un nodo

hoja. Así la búsqueda es ligeramente más rápida en un árbol B. Estas ventajas del árbol B sobre el árbol B+ se compensan con varias desventajas:

- Los nodos hoja y los que no son hoja tienen el mismo tamaño en un árbol B+. En un árbol B, los nodos que no son hoja son más grandes. Esto complica la gestión del almacenamiento del índice.
- La eliminación en un árbol B es más complicada. En un árbol B+ la entrada eliminada siempre aparece en una hoja. En un árbol B la entrada puede aparecer en un nodo que no sea hoja. Se debe seleccionar del subárbol el nodo que contiene la entrada eliminada el valor apropiado para sustituirlo.

Las ventajas de los árboles B son de poca importancia para índices grandes. Así muchos implementadores de sistemas de bases de datos prefieren la simplicidad estructural de los árboles B+.

2.5 Arquitectura cliente/servidor

Las bases de datos por lo general se encuentran en un lugar remoto al que trabajamos, por lo que es necesario hacer referencia a los conceptos relacionados con la tecnología de transmisión de datos, en especial de la arquitectura cliente/servidor.

La arquitectura cliente/servidor en su forma más simple se compone de una aplicación en una computadora llamada cliente, que realiza peticiones a otra aplicación en otra computadora, llamada servidor. La estructura fundamental cliente/servidor consiste de tres elementos físicos:

- **Cliente**. Es la computadora que alberga el componente cliente de la aplicación, el cual hace una o más peticiones por servicios. Se forma de la combinación de hardware y software que proporcionan interface con el usuario, integridad lógica con las aplicaciones y servicios de comunicaciones.
- **Servidor**. Es la computadora que tiene el elemento servidor de la aplicación. Este satisface peticiones del cliente para un servicio y regresa los resultados al componente cliente. Se forma de la combinación de hardware y software que proporcionan servicios de un servidor: mantenimiento, sistema de acceso a datos, integridad lógica entre aplicaciones, manipulación de datos y servicios de comunicaciones.
- **Red de comunicaciones**. Proporciona las facilidades de comunicación que permiten que las peticiones sean pasadas desde un cliente al servidor y que los resultados sean pasados desde el servidor al cliente.

A continuación se describen algunos de los aspectos más importantes de esta arquitectura:

- Separa el sistema de mantenimiento y las funciones del programa de aplicación de la base de datos.
- Centraliza la seguridad, integridad de los datos, y permite alta capacidad de procesamiento y almacenamiento en el servidor.
- Distribuye la interface de usuario y diseño, permite el desarrollo e implementación de aplicaciones y cuando es apropiado, controla la integridad de los datos para los clientes.
- Algunas aplicaciones necesitan de la misma información para funcionar, las aplicaciones cliente permiten acceder a múltiples sistemas de mantenimiento de bases de datos.
- Ofrece máxima seguridad, poder y velocidad.
- Permite la localización de los datos entre las aplicaciones.
- Reduce considerablemente el tiempo de desarrollo de aplicaciones.
- Se desarrolla con herramientas muy comerciales.
- Necesita de hardware y sistemas operativos relativamente nuevos.
- Los paradigmas del desarrollo de aplicaciones de datos ubican como muy costosos el desarrollo de aplicaciones bajo esta metodología.

El paradigma cliente/servidor permite a muchos clientes acceder a un servicio común que es proporcionado por un servidor compartido. También permite a un cliente acceder a múltiples servidores. El paradigma cliente/servidor puede ser adaptado para crear formas variadas de computo distribuido. Las variaciones de computo distribuido permiten que se construyan aplicaciones cliente/servidor utilizando una gran variedad de tecnologías de redes, bases de datos y de desarrollo.

El componente cliente es generalmente conocido como el componente activo debido a que es el que inicia una petición para un servicio. El servidor es conocido como el componente pasivo que espera por requerimientos de un cliente antes de hacer su trabajo. Sin embargo, un servidor que contenga una base de datos podría implementar *procedimientos almacenados* y *triggers* (disparadores) para jugar un papel activo.

Los *procedimientos almacenados* son programas para cumplir la parte lógica de la aplicación que se ejecuta en el servidor. Puede ser llamado por otro procedimiento almacenado, por un trigger o directamente desde un cliente.

Los *triggers* son eventos que se disparan cuando se detectan ciertos estados en la base de datos. Su función es permitir la implementación de reglas corporativas y permanentes y su uso más típico es proteger la integridad referencial de la base de datos. El trigger llama procedimientos almacenados para atender cada uno de los eventos.

En el modelo de base de datos bajo una arquitectura cliente/servidor la base de datos reside en una computadora diferente a la que corre programas aplicativos. Sin embargo el software de base de datos es divido entre el sistema cliente que corre programas y el sistema servidor que almacena la base de datos.

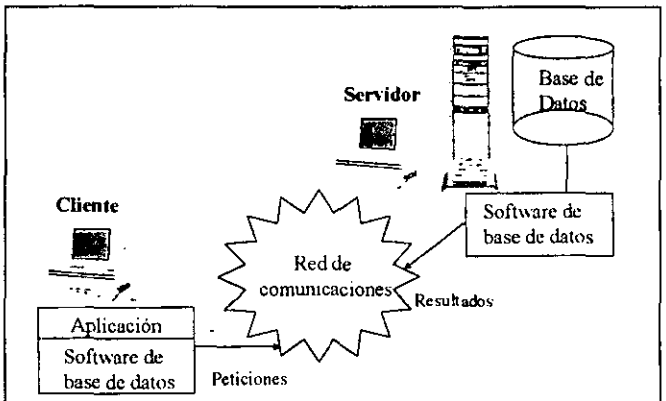


Figura 2.6.2 Modelo de base de datos en arquitectura cliente servidor

En este modelo, los programas aplicativos en el cliente realizan peticiones del software de base de datos local. El componente de software de base de datos local en el cliente se comunica con el software de base de datos complementario en el servidor. El software de base de datos en el servidor hace la petición para acceder la base de datos y regresa el resultado al cliente.

En el modelo de base de datos en arquitectura cliente/servidor es común referirse a software *front-end* y *back-end*.

Front-end típicamente corre en una computadora personal y satisface las necesidades de cómputo para un solo individuo. El software front-end tiene el papel de cliente en el modelo y ejecuta funciones que están orientadas a las necesidades del usuario final. Este tipo de software generalmente tiene las siguientes categorías:

- **Consultas simples y reportes**. Proporciona facilidad de uso para recuperar datos de la base de datos y preparar reportes simples.
- **Software de análisis de datos**. Además de recuperar datos, realiza análisis complejos en los datos.
- **Herramientas de desarrollo de aplicaciones**. Proporciona facilidades para producir aplicaciones de bases de datos personalizadas. Las herramientas en esta categoría pueden ser desde intérpretes y compiladores hasta herramientas CASE.
- **Herramientas de administración de base de datos**. Permite a los administradores de base de datos realizar tareas de administración en una computadora personal, tal como respaldo y recuperación.

Back-end es el software de base de datos y de red en una computadora que tiene el papel de servidor.

3. DESARROLLO DEL PROCESO DE AFINACIÓN Y OPTIMIZACIÓN DE UNA BASE DE DATOS

El desarrollo del proceso de afinación y optimización de la base de datos que se propone en esta tesis, está desarrollado sobre una plataforma Informix, por lo que es necesario conocer sus elementos específicos:

Informix™ es un manejador de base de datos relacional que actúa sobre una plataforma cliente/servidor normalmente con sistema operativo UNIX. El software de Informix es abierto, escalable, fácil de administrar y completamente extensible, proporcionando el tipo de flexibilidad esencial para las organizaciones en desarrollo. Ya sea para el almacenamiento de datos, el análisis y soporte de decisiones, entrega de contenidos de Web, o difusión de películas, sonido, fotografías y texto desde una biblioteca de información digital. Informix esta diseñados para permitir a las empresas actuales administrar en forma eficiente cualquier tipo de información en todo momento. Sus principales características son:

- Arquitectura de base de datos escalable, rápida y flexible.
- Aplicaciones Web y datawarehouse.
- Ambiente paralelo.
- Soluciones OLTP (Sistemas de procesamiento de transacciones en línea) para soportar volúmenes de transacciones cada vez más grandes y complejos, sin sacrificar el rendimiento.

3.1 Sistema operativo Unix

UNIX es un sistema operativo de propósito general, multiusuario, multitarea e interactivo. Ofrece las siguientes características:

- Un sistema de archivos jerárquico de volúmenes desmontables.
- Entrada/Salida compatible para archivos, dispositivos y comunicaciones entre procesos.
- Capacidad de iniciar procesos en forma paralela.
- Múltiples lenguajes para sistemas, seleccionables por el usuario.
- El software producido en las máquinas tiene un alto grado de portabilidad.

La filosofía de UNIX es ser portátil, migrable, modular, escalable, flexible, popular, actualizable y que pueda implementarse fácilmente. La forma de organizar la información en

los discos es a través de archivos, un *archivo* es una área en el sistema que contiene información. Un directorio es una clase de archivo que contiene otros archivos y directorios en una estructura jerárquica. En la cima de la estructura jerárquica esta el directorio *raíz* o *root*.

Un *programa shell* es un archivo regular que contiene comandos de sistema UNIX. Para ejecutarlo basta con teclear el nombre del archivo. Cuando se desea ejecutar una serie de comandos en repetidas ocasiones, es mejor guardar todos los comandos y ejecutarlos llamando el archivo que los contiene. Al programa shell se le pueden pasar datos a través de variables de ambiente, argumentos de comandos o por entrada de usuario.

3.2 *Informix OnLine Server*

Informix OnLine Server es un servidor de base de datos paralelo totalmente optimizado y completamente integrado diseñado para ayudar a transformar los recursos de información en una poderosa fuerza para la innovación comercial. Implementa una arquitectura multihilos (multithreaded), esto significa que son requeridos pocos procesos para efectuar actividades de DBMS, y que un proceso puede trabajar para más de una aplicación a través del uso de hilos (threads). Esos procesos son conocidos colectivamente como *el servidor de base de datos*. Se pueden alojar procesos dinámicamente para el servidor de base de datos como sea necesario.

La arquitectura multihilos permite una mejor escalabilidad. Esto significa que conforme se añadan más usuarios, será requerida una cantidad mínima de recursos adicionales del servidor para acomodar esos usuarios. Esto es debido a la escalabilidad inherente y la eficiencia de la implementación multihilos.

El sistema OnLine está conformado de 3 componentes principales:

El componente de proceso. Crea el servidor de base de datos. Los procesos que crean el servidor de base de datos son conocidos como *procesadores virtuales*. Cada procesador virtual (vp) pertenece a una *clase de procesador virtual*. Una clase vp es responsable de un conjunto de procesos para un conjunto específico de tareas. Esto significa que un vp de una cierta clase sólo puede correr *hilos* de la misma clase. Cada vp en la clase de vp es sólo otra instancia del mismo programa. Un vp puede pertenecer sólo a una clase. Una clase de vp puede tener uno o más vps, lo cual en la mayoría de los casos es configurable por el administrador. Los vps corren con un nivel de usuario *root* (el superusuario de UNIX). Esto es necesario porque el vp debe correr ciertas tareas como el usuario que inicio la tarea.

Tener vps corriendo como *root* proporciona el beneficio adicional de protección a los procesos del servidor de base de datos.

El componente de memoria compartida. La memoria compartida está dividida en 3 porciones. Las porciones residente y de mensaje son estáticas.

- **La porción residente** - La porción residente contiene el buffer cache y otra información de sistema. Esta porción de memoria compartida puede ser configurada para permanecer en la memoria principal. Incluye áreas que registran el estado del servidor, como buffers, candados, logs y localización de espacios y tablas en la base de datos.
- **La porción virtual** - La porción virtual contiene información acerca de los procesos y sesiones, y de los datos que son usados por ellos. Esta información crece y decrece constantemente, entonces el servidor de base de datos es responsable de manejar la alojación y dealojación de la memoria en esta porción.
- **La porción de mensajes** - La porción de mensajes compartida contiene los buffers de mensajes que son usados en los mecanismos de comunicación entre el cliente y el servidor con el objeto de coordinar sus actividades.

El componente de disco. Es una colección de una o más unidades de espacio de disco asignadas a la base de datos. La forma en como Informix maneja este componente es através de jerarquias de almacenamiento que guardan uno o más objetos de la base de datos. Inicialmente se tienen los dispositivos de disco dedicados a la base de datos, cada disco se divide en uno o más espacios de base de datos (*dbspace*). Cada espacio de base de datos , a su vez, esta formado por uno o más *chunks*. Dentro de cada *chunk* puede almacenarse una tabla, un índice, vista o cualquier otro objeto de la base de datos. Sin embargo, una tabla puede residir en una o más porciones de disco o discos. Se controla la disposición de una tabla en disco cuando se crea asignándole un *dbspace*.

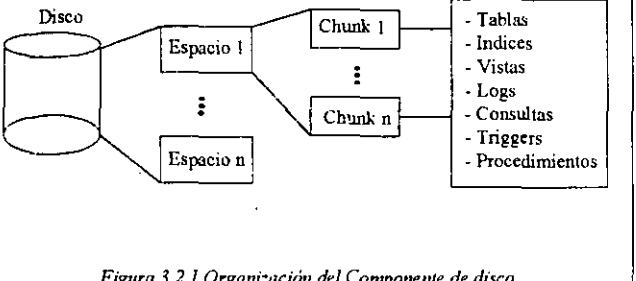


Figura 3.2.1 Organización del Componente de disco

Cada componente de tiene repercusión directa en el desempeño del sistema ya que maneja los principales recursos de una computadora: CPU, memoria y disco. Antes de iniciar el proceso de afinación de base de datos es necesario revisar la configuración de estos tres elementos y determinar si representan un problema de desempeño.

- **Cpu.** Un alto uso de CPU no siempre significa un problema de desempeño. Un alto grado de utilización de CPU podría indicar que la computadora esta siendo utilizada al máximo de su capacidad.
- **Memoria.** El sistema operativo utiliza la memoria para ejecutar procesos. Primero, busca en páginas no utilizadas, en caso de que todas las páginas estén utilizadas, el sistema tiene que escoger páginas que otros procesos están usando y que son las menos necesarias; esté proceso de localización de páginas es llamado *page scan.* El manejador de memoria utiliza el algoritmo LRU (last recently use) para seleccionar las páginas que serán copiadas a disco y liberarlas para poder ser utilizadas por otros procesos. Eventualmente, las páginas copiadas a disco deberán ser regresadas para ser utilizadas por su proceso.
- **Disco.** Todos los datos que residen en la base de datos son almacenados en disco. La rapidez con la que el servidor puede copiar las páginas de datos desde y hacia el disco determina el desempeño de una aplicación. Los discos son comúnmente el elemento más lento en el proceso de I/O para una transacción o consulta. Los discos pueden llegar a estar subutilizados o saturados cuando los usuarios requieren páginas muy constantemente. Un aspecto importante a considerar es la disposición de *chunks* y *dbspaces* a lo largo de todos los discos.

La forma en como Informix maneja estos recursos para lograr un mejor desempeño es a través de los siguientes mecanismos:

- **Administración de discos (unbuffered disk).** Utilizados para mejorar la rapidez en las operaciones de I/O. Cuando las tablas se almacenan en estos dispositivos, el servidor de base de datos maneja la organización física y minimiza operaciones de I/O.
- **Manejo de memoria compartida en forma dinámica.** Todas las aplicaciones que utilizan la base de datos comparten datos en el espacio de memoria de la base de datos. Después de que una aplicación lee datos de una tabla, otra aplicación puede acceder datos independientemente si están o no en la memoria. A través del manejo de memoria compartida se minimizan los accesos a disco y el impacto asociado con el desempeño.
- **Ubicación de hilos (threads) dinámicamente.** Para soportar múltiples aplicaciones clientes, Informix utiliza un número relativamente pequeño de procesos llamado procesador virtual. Un procesador virtual es un proceso multihilos que puede servir a múltiples clientes y, donde necesariamente, corren múltiples hilos en paralelo para una consulta. En esta forma se proporciona una arquitectura flexible que proporciona balanceo dinámico de carga de trabajo.
- **Ejecución paralela.** Informix puede ubicar múltiples hilos para trabajar en paralelo en una consulta. Con la partición local de tablas (fragmentación) para distribuir tablas inteligentemente a través de los discos se mejora el desempeño.
- **Conexiones múltiples.** Una aplicación comunica con la base de datos a través de facilidades de conexión que el servidor proporciona. En el nivel de código fuente, un cliente conecta la base de datos a través de un enunciado SQL. Detrás de esto, el cliente utiliza facilidades transparentes a la aplicación.

Por último, es importante saber los tipos de datos que se manejan en Informix, con el objetivo de optimizar definición de tablas e índices y en especial de las columnas de tipo TEXT y BYTE. Los tipos de datos que Informix maneja son:

Tipo	Explicación
BYTE	Almacena cualquier clase de dato binario
CHAR(n)	Almacena secuencias de caracteres, incluyendo letras, números y símbolos
DATE	Almacena fechas de calendario
DATETIME	Almacena fechas de calendario combinadas con horas
DECIMAL	Almacena números con escala y precisión definidos
DOUBLE	Almacena números de doble precisión de punto flotante
FLOAT(n)	Almacena números de doble precisión de punto flotante
INTEGER	Almacena números enteros en el rango de -2147483647 a 2147483647
INTERVAL	Almacena periodos de tiempo
MONEY(p,s)	Almacena cantidades de dinero
SERIAL	Almacena enteros secuenciales
SMALLFLOAT	Almacena números de precisión sencilla de punto flotante
SMALLINT	Almacena números enteros en el rango de -32767 a 32767
TEXT	Almacena cualquier clase de dato texto
VARCHAR(m,r)	Almacena secuencia de caracteres de longitud variable

El tipo de dato BYTE almacena cualquier tipo de dato binario en un flujo byte no diferenciado. Datos binarios típicamente consisten de archivos de hojas de cálculo, módulos de carga de programas o patrones de voz digitalizados. El tipo BYTE no tiene un tamaño máximo. Una columna BYTE tiene un límite teórico de 2^{31} bytes y un límite práctico determinado por la capacidad en disco.

Se puede almacenar, recuperar, actualizar o borrar el contenido de una columna BYTE. Sin embargo, no se puede utilizar una columna BYTE en operaciones aritméticas o de cadenas. Tampoco se puede utilizar una columna BYTE con funciones agregadas, cláusulas IN, cláusulas MATCHES o LIKE, cláusulas GROUP BY o cláusulas ORDER BY. El tipo BYTE se puede utilizar como objeto en una expresión booleana sólo si se esta evaluando para valores nulos.

El tipo de datos CHAR almacena secuencia de letras, números y símbolos. Puede ser un carácter de uno o muchos bytes. Una columna carácter tiene una longitud n máxima en bytes, donde $1 \leq n \leq 32767$.

El tipo de dato DATE almacena fechas de calendario. Este tipo requiere 4 bytes. Una fecha de calendario es almacenada internamente como un valor entero igual al número de días desde el 31 de Diciembre de 1899. Debido a que DATE almacena enteros, se pueden utilizar expresiones aritméticas, tales como sumas y restas.

DATETIME almacena instantes en el tiempo expresados como una fecha de calendario y hora del día. Se puede elegir la forma en que el valor DATETIME será almacenado, su precisión va desde el rango de un año hasta fracciones de segundo. El tipo DATETIME está compuesto de una secuencia de campos que representan cada componente de tiempo que se quiere registrar.

DECIMAL almacena números de tipo flotante de hasta 32 dígitos significativos como máximo. Puede tomar dos formas: de punto flotante o con decimales fijo.

FLOAT almacena números de punto flotante de doble precisión de hasta 16 dígitos significativos.

INTEGER guarda números enteros del el rango de -2147483647 a 2147483647. Requiere 4 bytes por valor.

INTERVAL representa un intervalo de tiempo. Siempre esta compuesto de un valor o una secuencia continua de valores que representan un componente de tiempo.

MONEY almacena cantidades de dinero. Al igual que un decimal, este tipo guarda números de hasta 32 dígitos significativos.

SERIAL almacena enteros secuenciales asignados automáticamente por el servidor cuando se hace una inserción. Sólo se puede definir una columna SERIAL en una tabla.

SMALLFLOAT almacena números de punto flotante de doble precisión con aproximadamente 8 dígitos significativos.

SMALLINT almacena números enteros en el rango de -32767 a 32767

TEXT almacena cualquier tipo de texto. Al igual que el tipo BYTE tiene una longitud máxima de 2^{31} bytes, aunque en realidad el tamaño esta determinado por la capacidad en disco disponible. Las columnas TEXT típicamente almacenan memos, capítulos de manuales, documentos o archivos de programas fuente. Se puede almacenar, recuperar, actualizar o borrar el contenido de una columna TEXT. Sin embargo no se puede utilizar en operaciones aritméticas y de cadenas. Tampoco se pueden utilizar en funciones de agregación, en cláusulas IN, con cláusulas LIKE o MATCH y con cláusulas GROUP BY u ORDER BY.

3.2.1 AFINACIÓN Y OPTIMIZACIÓN DE LA BASE DE DATOS

Una vez implementado nuestro sistema de información utilizando una plataforma Informix y con el sistema en operación, el componente de control debe realizar su función en base a los objetivos y estándares de información definidos previamente. Por lo que para mantener un desempeño óptimo de las aplicaciones y la base de datos, se debe desarrollar un plan para medir el desempeño del sistema en relación con estos mecanismos, realizar los ajustes necesarios para mantener un buen desempeño y tomar las medidas correctivas que se anticipen y/o corrijan problemas, todo dentro de un proceso iterativo. La siguiente guía es una buena base que Informix recomienda:

1. Establecer objetivos de desempeño del sistema
2. Tomar mediciones periódicas de utilización de recursos y actividad de la base de datos
3. Identificar síntomas de problemas de desempeño: desproporcionado uso de CPU, memoria o disco
4. Optimizar la configuración de *chunks* y *dbspace*, incluyendo disposición de *logs*, espacio para sorts y espacio para tablas temporales
5. Optimizar la disposición de tablas, tamaño de *extents* y fragmentación
6. Mejorar índices
7. Optimizar actividades de I/O de fondo (background), incluyendo logging, puntos de verificación (checkpoints) y paginación.
8. Programar respaldos y operación batch en horas no críticas
9. Optimizar la implementación de las aplicaciones de base de datos

Los objetivos, y estándares se definieron en la fase de análisis y diseño del sistema de información y son el punto de comparación con la operación diaria del sistema. Las mediciones que describen el desempeño, es decir las que determinan si es necesario o no llevar a cabo un proceso de afinación de la base de datos son:

- **Rendimiento (Throughput).** Mide el desempeño total del sistema. Típicamente es medido en transacciones por segundo o minuto (TPS o TSM). Depende de los siguiente factores:
  1. Especificaciones de la computadora host
  2. La carga de proceso en el software
  3. La disposición de los datos en el disco
  4. El grado de paralelismo para hardware y software
  5. El tipo de transacciones procesadas

- **Tiempo de respuesta.** Se mide para transacciones o consultas individuales. Es tratado típicamente como el tiempo transcurrido desde el momento que el usuario introduce un comando o activa una función hasta el tiempo en que la aplicación indica que la función o comando se han completado. El tiempo de respuesta para un a típica aplicación de base de datos incluye la siguiente secuencia de acciones:
  1. La aplicación hacia una consulta a la base de datos
  2. Ocurre una optimización de la consulta y se recupera algún procedimiento almacenado
  3. Se recupera, agrega, borra o actualiza los renglones apropiados y ejecuta operaciones de I/O relativas a la consulta
  4. Se realizan algunas operaciones de fondo de I/O, tales como registro de eventos (logging)y paginación, que ocurren durante el periodo en el cual la consulta o transacción está pendiente
  5. El resultado es regresado a la aplicación
  6. La aplicación muestra la información

  El tiempo de respuesta se mide a través de los siguientes métodos:

  1. Comandos de sistema operativo. El sistema operativo típicamente tiene una utilería que se utiliza para cronometrar un comando. El comando *time* en muchos sistemas cronometra los comandos, el siguiente ejemplo muestra la salida de un comando UNIX time:

     time commands.dba

     4.3 real 1.5 user 1.3 sys

     El ejemplo muestra la cantidad de tiempo transcurrido (real), la cantidad de tiempo utilizado en ejecutar rutinas definidas por el usuario (user), y el tiempo ejecutando llamadas al sistema (sys).
  2. Monitor del sistema operativo. El sistema operativo usualmente cuenta con un monitor de desempeño (performance monitor) que se puede utilizar para medir el tiempo de respuesta para una consulta o proceso.
  3. Funciones de tiempo dentro de la aplicación. Muchos lenguajes de programación tienen librerías para funciones de tiempo. Se pueden insertar estas funciones de tiempo en el código para registrar el tiempo transcurrido entre acciones especificas.

- **Costo por transacción.** Es una medida financiera que es utilizada para comparar el costo operativo total entre aplicaciones, servidor de base de datos o plataformas de hardware. Para medir el costo por transacción:
  1. Calcular todo el costo asociado con la operación de una aplicación. Este costo puede incluir el precio del hardware y software instalado, costos operativos y otros gastos.
  2. Proyecta el número total de transacciones y consultas para el tiempo de vida efectivo de una aplicación.
  3. Divide el total del costo entre el total de transacciones.

  A pesar de que esta medida es útil para planeación y evaluación, difícilmente nos ayuda en cuestiones diarias de optimización de la base de datos.
- **Utilización de recursos.** Siempre que un recurso del sistema, tal como CPU, memoria o disco, está ocupado por una transacción o consulta, este recurso no esta disponible para procesar otros requerimientos. Los requerimientos que quedan pendientes deben esperar por el recurso. Cuando un componente está demasiado ocupado llega a formar un cuello de botella (bottleneck) en el flujo de la actividad.

Para registrar las mediciones anteriores, Informix proporciona utilerías para capturar información acerca de la configuración y desempeño del sistema. Se pueden utilizar estas utilerías regularmente para obtener un perfil histórico de la actividad de la base de datos con el fin de identificar y analizar actividades que tengan alto impacto en la operación diaria del sistema, aunque su utilización varía de una instalación a otra. Las utilerías generales que proporciona Informix son:

- Onstat.
- Onlog.
- Oncheck.

Onstat. Se utiliza para verificar el estado actual de la base de datos y monitoreo de actividades. Despliega una amplia variedad de información relativa a desempeño y estado de la base de datos. Tiene una serie de opciones, las cuales se describen a continuación:

Opción	Información
onstat -p	Perfil de desempeño
onstat -b	Buffers actualmente usados
onstat -l	Información de logs
onstat -x	Transacciones
onstat -u	Hilos de usuarios
onstat -r	Colas LRU (last recently used)
onstat -f	Estadísticas de paginación
onstat -g	Información general

La opción onstat -g acepta un número de argumentos para especificar información adicional. Los siguientes argumentos muestran información de la utilización de CPU:

Argumento	Descripción
Act	Muestra hilos activos
Ath	Despliega todos los hilos
glo	Muestra información global de multihilos, incluyendo información de sesiones y procesos
Ntd	Despliega estadísticas de red por servicio
Ntt	Muestra usuarios de red
Ntu	Muestra estadísticas de usuarios de red
Qst	Despliega estadísticas de colas
Rea	Muestra hilos disponibles
Sch	Despliega número de operaciones semáforo, esperas para cada procesador virtual
ses	Muestra información por sesión
Sle	Despliega todos los hilos suspendidos (sleeping)
Sql	Muestra información SQL por sesión
Sts	Despliega máximo y actual uso de pila por hilo
Tpf	Despliega un perfil de hilo
Wai	Muestra espera de procesos
Wst	Despliega estadísticas de espera

Los argumentos de utilización de memoria son:

Argumento	Descripción
Ffr	Despliega fragmentos libres para sesión o por memoria compartida
Dic	Muestra una línea de información para cada tabla en el diccionario
Mem	Muestra estadísticas de memoria
Nsc	Muestra el estado de memoria compartida por cliente ID
Nsd	Despliega información de memoria compartida para hilos en red
Nss	Despliega el estado de memoria compartida por sesión
Seg	Muestra estadísticas de segmentos de memoria
Ufr	Muestra fragmentos ubicados por usuario o sesión

Los argumentos relativos a la utilización de disco son:

Argumento	Descripción
Iof	Despliega estadísticas asíncronas de I/O por chunk o archivo
Iog	Muestra información global de I/O
Ioq	Despliega estadísticas de colas de I/O
Iov	Despliega estadísticas de I/O por procesador virtual

Onlog. Despliega todo o una porción de los archivos logs lógicos. Este comando puede actuar sobre algunos archivos seleccionados o sobre todo el log. Puede ser útil para identificar transacciones problemáticas en periodos de alta utilización.

Oncheck. Despliega información acerca de estructuras de almacenamiento en un disco, incluyendo chunks, dbspaces, blobspaces, extents, renglones, catálogos de tablas y otras opciones. También se puede utilizar para reconstruir un índice que reside en el mismo espacio de la tabla referenciada. Tiene las siguientes opciones:

Opción	Información
-pB	Información acerca de blobsapce (datos TEXT o BYTE)
-pc	Tablas del catálogo de la base de datos
-pd	Información de renglones sin datos TEXT o BYTE
-pD	Información de renglones con datos TEXT o BYTE
-pe	Información de chunks y extents
-pK	Valores de índices e identificadores de renglones
-pp	Páginas por tabla o fragmento
-pP	Páginas por chunk
-pr	Páginas reservadas
-pt	Espacio utilizado por tabla o fragmento
-pT	Espacio utilizado por tabla, incluyendo índices

Las mediciones tomadas nos proporcionan una aproximación inicial de la situación actual del sistema, con esta primera percepción se identifican los recursos críticos para llevar a cabo un monitoreo más a detalle con el propósito de tener claramente identificadas las causas de los problemas de desempeño. Estas causas podrían tener su origen en alguno de los recursos que maneja la base de datos como CPU, memoria o disco; o también en algún objeto propio de la base de datos como una tabla o índice.

Las mediciones tomadas con los comandos *onstat* relativos a la utilización del disco podrían indicarnos un problema en la configuración del componente de disco (espacios y chunks). Hay que poner especial cuidado en discos que tiene más actividad que otros y tratar de balancear la carga entre todos los disco disponibles. Así, si en un disco estan las tablas con mayor actividad en consultas, por ejemplo, tratar de distribuirlas en cada uno de los discos dedicados a la base de datos.

Cuando el problema es relativo a una tabla o índice a través del comando *oncheck*, por ejemplo tablas con demasiada actividad, los factores que se habrán de analizar para saber de que manera se afecta el desempeño de una tabla individual o un fragmento de tabla son:

- La disposición de la tabla o fragmento.
- El tamaño de la tabla o fragmento.
- La estrategia de indexación utilizada.
- La frecuencia de acceso a la tabla.

Para tablas con alta actividad de operaciones de disco (I/O) se pueden colocar en un dispositivo dedicado y así reducir la contención para los datos que están almacenados en la tabla.

El tamaño de una tabla incluye todas las páginas dentro del tblspace: páginas de datos, páginas index y páginas que almacenan datos TEXT o BYTE. Un espacio puede comprender múltiples *chunks*, y cada *chunk* puede representar un disco diferente. Este arreglo permite distribuir datos en un *dbspace* sobre múltiples discos.

Las páginas de disco ubicadas para una tabla son llamadas *tblspace*. El tblspace incluye páginas de datos y páginas índice. Si hay datos BYTE o TEXT asociados con la tabla esta no es almacenada en un alternativo dbspace, sino que se incluye en el tblspace. El tblspace no corresponde a alguna región fija dentro de un dbspace. Los *extents* e índices hacen que la tabla se distribuya a través del dbspace. Un *blobspace* es una unidad de almacenamiento lógica compuesta de uno o más chunks que almacenan sólo datos TEXT o BYTE. Se puede utilizar discos separados para almacenar este tipo de datos. Cuando se definen blobspaces en discos diferentes de los demás datos se tienen las siguientes ventajas:

- Acceso paralelo a la tabla y a los datos TEXT o BYTE.
- Datos TEXT y BYTE no son registrados en el log, por lo que reducen actividades de I/O.

El tamaño de una tabla incluye todas las páginas dentro del tblspace: páginas de datos, páginas índice y páginas que almacenan datos TEXT o BYTE. La forma en cómo se estima las páginas de datos de una tabla depende del tipo de renglones que contiene, que pueden ser de longitud fija o longitud variable. Una tabla con registros de longitud fija no contiene columnas de tipo VARCHAR o NVARCHAR.

Para estimar el tamaño de la página, tamaño del renglón, número de renglones y número de páginas de datos, se siguen los siguientes pasos:

1. Utilizar el comando *oncheck -pr* para obtener el tamaño de la página. Al valor obtenido se le resta la cantidad de 28 y el nuevo valor es llamado *pageuse* (uso de página).
2. Para calcular el tamaño de un renglón, sumar las longitudes de todas las columnas en la definición de la tabla. Columnas TEXT y BYTE utilizan 56 bytes cada una.
3. Estimar el número de renglones esperados de la tabla.
4. Se utiliza la siguiente fórmula para calcular el número de páginas de datos. El número máximo de renglones por página es 255, independientemente del tamaño del renglón.

   Pag_dat = renglones / trunc(pageuse/(rowsize + 4))

Conforme se agregan renglones a la tabla, Informix ubica espacio en disco en unidades llamadas *extents*. Cada *extent* es un bloque de páginas físicas contiguas. Siempre que el dbspace incluye más de un chunk, cada *extent* se ubica de forma completa en un chunk para permanecer continuo. Cuando se crea una tabla, se especifica el tamaño del primer *extent* así como el tamaño de los *extents* a ser agregados conforme la tabla aumente su tamaño. El valor por omisión para el tamaño de *extents* y el siguiente *extent* es ocho veces el tamaño de la página en el sistema. Sin embargo, se puede cambiar el tamaño del extent a ser agregado. Este cambio no afecta los extents existentes.

Al consultar una tabla, Informix debe prevenir el uso de este recurso a otros usuarios. Un *candado* (*lock*) es un mecanismo de software que previene que otros usuarios utilicen un recurso. La cantidad de datos que un candado protege se llama *granularidad.* La granularidad de candados afecta el desempeño de la base de datos. Cuando un usuario no puede acceder un renglón, tiene que esperar a que otro usuario quite el candado sobre el renglón. Si un usuario bloquea toda una página, existe una posibilidad mayor de que más usuarios esperaran por un renglón en la página. Los diferentes tipos de candados que se pueden utilizar son:

- **Candado por renglón individual**. Cuando se inserta o actualiza la tabla, el servidor crea un renglón bloqueado. Generalmente candados por renglón proporcionan la mejor solución para el desempeño de la base de datos cuando se actualiza un número relativamente pequeño de renglones. Sin embargo, existe una carga mayor para el servidor en obtener un candado.
- **Candados por página**. Es el candado por omisión cuando se crea una tabla y no se especifica el tipo de candado. Informix bloquea la página que contiene el renglón a ser afectado. Si se actualizan diferentes renglones en la misma página, se usará sólo un candado para la página. El bloqueo por página es útil para las tablas con cambios en muchos renglones al mismo tiempo.
- **Candados por tabla**. En el ambiente warehouse resulta apropiado que las consultas tengan una granularidad grande en candados. Si una consulta requiere casi toda la tabla, la eficiencia se incrementará si se tiene un número pequeño de candados por tabla en vez de tener muchos candados por página.
- **Candados por base de datos**. Asegura que nadie pueda modificar la base de datos.

Para tablas de gran tamaño y actividad una alternativa es utilizar la *fragmentación*, con esto se almacenan los datos de una tabla en múltiples dispositivos de disco. El uso

apropiado de fragmentación de tablas reduce significativamente la contención en operaciones de I/O y reduce las esperas por información. Una estrategia de fragmentación consiste de dos partes:

- Un esquema de distribución que especifica como agrupar renglones dentro de fragmentos.
- El conjunto de espacios en el cual se localizarán los fragmentos.

La estrategia de fragmentación implica las siguientes decisiones:

- Determinar el objetivo de fragmentación.
- Elegir la manera en que las tablas serán fragmentadas, incluyendo índices.
- Elegir el esquema de distribución.
- Decidir el número y localización de los fragmentos.

Analizando la aplicación y la carga de trabajo, se determina el balance para establecer los objetivos de fragmentación:

- **Mejorar desempeño de consultas individuales**. Tratar de distribuir todos los renglones de una tabla equitativamente sobre todos los discos. El tiempo de consulta se reduce cuando no se tiene que esperar para recuperar información de un fragmento de tabla que tiene más renglones que otros fragmentos.
- **Reducir contención entre consultas y transacciones**. La fragmentación reduce la contención para datos que múltiples consultas o transacciones utilizan. La contención es reducida cuando consultas simultaneas a una tabla acceden a datos indexados para regresar pocos renglones. Para fragmentar una tabla para reducir contención, se comienza por investigar que consultas acceden que parte de la tabla y fragmentar los datos de manera que las consultas puedan ser dirigidas a un fragmento mientras otras consultas son dirigidas a otros fragmentos. Informix realiza el direccionamiento cuando evalúa la regla de fragmentación para la tabla. Finalmente es importante almacenar los fragmentos de la tabla en discos separados.
- **Incrementar la disponibilidad de datos.** Cuando se distribuye una tabla e índice en diferentes discos, se incrementa la disponibilidad de datos durante fallas de alguno de los discos. Informix continuará manejando accesos a fragmentos almacenados en discos que permanecen en operación.

Para determinar la estrategia de fragmentación hay que conocer los datos que son utilizados en la tabla.

- Identificar las consultas críticas a la tabla.
- Establecer la forma en como los datos están siendo accesados.
- Determinar que porción de los datos examina cada consulta.
- Determinar que operación crea archivos temporales.
- Examinar columnas en la tabla para determinar que escenario de fragmentación podría mantener cada proceso con la misma carga de trabajo.

Cuando se fragmenta una tabla, la disposición física aplica a fragmentos de tabla individuales. Debido a que cada fragmento reside en su propio espacio de disco, la disposición debe ser direccionada separadamente para los fragmentos en cada disco. Las tablas fragmentadas y no fragmentadas difieren en la siguiente forma:

- Para tablas fragmentadas, cada fragmento esta por separado en un espacio. Para tablas no fragmentadas, la tabla puede ser colocada en el espacio por omisión de la base de datos actual. Independientemente de si la tabla esta fragmentada o no, Informix recomienda crear un simple chunk en cada disco para cada espacio.
- El tamaño de los extents para una tabla fragmentada es usualmente más pequeño que el tamaño del extent para su equivalente para una tabla no fragmentado debido a que los fragmentos no crecen tanto como la tabla entera.

Después de decidir que es lo que se va a fragmentar ya sea tablas, índices o ambos, además de saber la manera como serán distribuidos los fragmentos, es tiempo de elegir un escenario para implementar la fragmentación. Informix soporta los siguientes escenarios de distribución:

- **Round-robin**. Coloca renglones uno después de otro alternando los fragmentos para distribuir los renglones de forma constante. Para una operación de INSERT se utiliza una función hash en un número aleatorio para determinar el fragmento en el cual colocar el registro.
- **Basados en expresión**. Coloca los renglones que contienen valores específicos en el mismo fragmento. Se especifica una expresión de fragmentación que define un criterio para asignar un conjunto de renglones a cada fragmento, ya sea como un rango o una regla arbitraria. Un *fragmento residuo* (*remainder*) especifica el fragmento que contendrá todos los renglones que no ajusten con el criterio de algún otro fragmento.

Algunos factores a considerar al momento de elegir el escenario de distribución elegido se listan a continuación:

- Si las consultas tienden a leer toda una tabla.
- Si se conoce la distribución de los datos a ser agregados.
- Si los programas aplicativos tienden a borrar muchos registros.

Básicamente el escenario Round-robin es la forma más fácil y segura de balancear los datos. Sin embargo, con la distribución Round-robin no se tiene información acerca del fragmento en el que se localiza el renglón, por lo que el servidor no puede discriminar fragmentos. En general el escenario Round-robin es la elección correcta sólo cuando las condiciones siguientes aplican:

- Las consultas por lo general leen tablas enteras.
- No se conoce la distribución de los datos a ser agregados.
- Las aplicaciones tienden a no borrar muchos registros.

Un escenario basado en expresión podría ser la mejor elección con las siguientes condiciones:

- Las consultas o transacciones leen porciones especificas de las tablas.
- Se conoce la distribución de los datos.

Para el escenario de distribución basado en expresiones el primer paso es determinar la distribución de los datos en la tabla, particularmente la distribución de los valores para la columna en la que se basará la expresión de fragmentación.

Una vez con la distribución de los datos, se diseña la regla de fragmentación que distribuye los datos a lo largo de los fragmentos. Es importante no utilizar columnas que están sujetas a constantes operaciones de actualización. Tales actualizaciones pueden causar que renglones se muevan de un fragmento a otro y esta actividad incrementa la carga en el CPU y operaciones de lectura y escritura. También es importante crear fragmentos que no se encimen y basados en una sola columna y sin un fragmento residuo para lograr una mejor eliminación de fragmentos.

Las principales recomendaciones para fragmentar tablas e índices son:

- Para un desempeño óptimo en reportes y transacciones, fragmentar la tabla para incrementar paralelismo, pero no fragmentar los índices. Separar los índices y colocarlos en un espacio separado.

- Para un mejor desempeño en transacciones en línea, fragmentar índices para reducir contención entre sesiones.
- Utilizar fragmentación Round-robin en los datos cuando la tabla es leída de forma secuencial. Round-robin es un buen método para colocar los datos equitativamente en todos los discos cuando no existe una columna en la tabla que pueda ser utilizada en una expresión de fragmentación.
- Mantener expresiones de fragmentación simples. Las expresiones de fragmentación pueden ser tan complejas como se desee, sin embargo expresiones complejas toman más tiempo para ser evaluadas.
- Evitar expresiones que requieran una conversión en los tipos de datos.
- No fragmentar columnas que cambien frecuentemente.
- No fragmentar todas las tablas. Identificar las tablas críticas que son accesadas más frecuentemente y colocar sólo un fragmento en cada disco.
- No fragmentar tablas pequeñas.

Cuando se fragmenta una tabla los índices que tiene asociados son fragmentados implícitamente, de acuerdo al escenario de fragmentación utilizado. También se puede utilizar la cláusula FRAGMENT BY EXPRESSION del comando CREATE INDEX para fragmentar un índice para una tabla en forma explicita. Cada índice de una tabla fragmentada ocupa su propio espacio de tabla (tblspace) con sus extents. En otras palabras, los índices pueden tener la misma estrategia de fragmentación que la tabla o una estrategia diferente.

La fragmentación de una tabla afectará directa o indirectamente la estrategia de fragmentación del índice. Un *índice sujeto (attached)* es un índice que explícitamente sigue la estrategia de fragmentación de la tabla. Informix crea un índice sujeto automáticamente cuando se fragmenta la tabla. Para crear un índice sujeto, no se especifica una estrategia de fragmentación, como en el siguiente enunciado:

```
CREATE TABLE tb1(a int)
        FRAGMENT BY EXPRESSION
                (a >= 0 and a < 5) IN dbsapce1,
                (a >=5  and  a< 10) IN dbspace2;
CREATE INDEX idx ON tb1(a);
```

El servidor en Informix fragmenta el índice sujeto de acuerdo al mismo escenario de distribución de la tabla, utilizando la misma regla de fragmentación. Como resultado los índices sujetos tienen las siguientes características físicas:

- El número de fragmentos de índice es el mismo que el número de fragmentos de datos.
- Cada fragmento de índice reside en el mismo espacio de tabla (tblspace) con la correspondiente tabla.
- El índice utiliza 4 bytes para cada entrada en el índice.

Un *índice separado (dettached)* es un índice con una estrategia de fragmentación separada y que se coloca explícitamente en el comando CREATE INDEX.

```
CREATE  TABLE  tb1 (a int)
        FRAGMENT BY EXPRESSION
                (a <= 10) IN tabdbspc1,
                (a <= 20) IN tabdbspc2,
                (a <= 30) IN tabdbspc3;
CREATE INDEX  idx1  ON  tb1 (a)
        FRAGMENT BY EXPRESSION
                (a <= 10) IN idxdbspc1,
                (a <= 20) IN idxdbspc2,
                (a <= 30) IN idxdbspc3;
```

El ejemplo ilustra una estrategia común de fragmentación igual para datos y para índices pero especificando diferentes espacios de base de datos para el índice fragmentado. Si no se requiere fragmentar el índice, el índice completo puede colocarse en un espacio separado.

Los índices se pueden fragmentar para tablas fragmentadas por expresión. Sin embargo, no se puede crear un escenario Round-robin para un índice. Siempre que una tabla utilice la estrategia Round-robin, es recomendable convertir todos los índices a índices separados para un mejor desempeño. Los índices separados tienen las siguientes características:

- Cada fragmento de índice reside en diferente tblspace del que tiene la tabla.
- Un índice separado utiliza 8 bytes de espacio en disco por entrada en el índice.

Si Informix lee un índice fragmentado se deben leer múltiples fragmentos, unir la información recuperada de cada fragmento y el resultado mostrarlo como uno sólo. Debido a este requerimiento el desempeño en lecturas de índices podría decaer si el índice esta fragmentado. Por lo que el servidor coloca las siguientes restricciones en índices:

- No se pueden fragmentar índices por escenario Round-robin.
- No se pueden fragmentar índices por expresión que contienen columnas que no están en el índice.

La *discriminación de fragmentos* es la característica que tiene Informix de reducir el número de fragmentos involucrados en una operación de base de datos. Esta capacidad puede mejorar el desempeño y reducir la contención para los discos en los que los fragmentos residen. También proporciona mejor tiempo de respuesta y concurrencia en consultas. Debido a que no se necesita leer fragmentos innecesarios en las operaciones de lectura/escritura para una consulta, así como la actividad en las colas LRU. Siempre que se pueda discriminar fragmentos de una búsqueda, depende de dos factores:

- La forma en como se expresa la consulta (la expresión en la cláusula WHERE de un comando SELECT, INSERT, DELETE o UPDATE).
- El escenario de distribución de la tabla.

Una vez que se determina la estrategia de fragmentación se tiene que monitorear la fragmentación. Con la utilería *onstat* se monitorea la actividad el disco. La utilería onstat -g ppf muestra el número de requerimientos de lectura y escritura enviados a cada fragmento que están en uso.

Finalmente las actividades de fondo son esenciales para mantener la base de datos, sin embargo crean una carga adicional en el CPU y operaciones de I/O. Estas actividades toman lugar en seguida que surge una transacción. Algunas de estas actividades son:

- Puntos de verificación (checkpoints).
- Registro de eventos (Logging).
- Paginación (page cleaning).
- Recuperación.
- Replicación de datos.
- Consumación o no-consumación de transacciones (commit o rollback).
- Auditoría.

Los puntos de verificación existen, independientemente si hay mucha o poca actividad en la base de datos, sin embargo, se pueden incrementar conforme la actividad de la base de datos se incrementa. Los puntos de verificación, paginación y registro de eventos (logging) son necesarios para conservar la consistencia en la base de datos.

Los discos que contienen páginas reservadas por el sistema, logs físicos, y el dbspace que contiene los logs lógicos son críticos para la operación de la base de datos. Informix coloca estos tres elementos claves en el dbspace llamado *root*. Aunque el servidor también coloca tablas temporales y archivos para ordenación en el espacio root (dbspace root), se recomienda utilizar áreas para tablas temporales y archivos de ordenación, a través de la sección DBSPACETEMP dentro del ambiente de la base de datos.

4. CASO PRÁCTICO

Con el propósito de ejemplificar al proceso de afinación de una base de datos se presenta a continuación un caso práctico referente a una base de datos en un sistema de riesgos de mercado. Este caso fue obtenido de la institución financiera de banca múltiple BanCrecer, concretamente del área de administración de riesgos de mercado.

4.1 Administración de riesgos en las instituciones financieras

A raíz de los cambios que se presentan en los mercados financieros, se ha venido desarrollando una cultura de administración de riesgos en las instituciones financieras del mundo. Los factores que han contribuido a la necesidad de estimar los diferentes riesgos a los que se enfrentan las instituciones financieras son los siguientes:

- Impacto de la crisis económica.
- Gran volatilidad en los mercados financieros.
- Medidas reguladoras.
- Nuevos productos.

Los diferentes tipos de riesgos a los que se puede enfrentar una institución financiera son los siguientes:

- **Riesgo crediticio.** Provocado por todas las operaciones realizadas en las cuales existe el riesgo de no cobrarlas a su vencimiento, o incluso que no lleguen a ser recuperadas.
- **Riesgo de Mercado.** Riesgo por los cambios en las variables financieras tales como las tasas de interés y la fluctuación en el tipo de cambio.
- **Riesgo de Liquidez.** Provocado por la probabilidad de no poder comprar o vender los activos o instrumentos que se tengan o se deseen tener en posición en las cantidades requeridas.
- **Riesgo de Operación.** Riesgo que tienen todas las instituciones financieras en cuanto a las operaciones realizadas por el personal de la institución y del manejo de información por parte de sus sistemas.

Las instituciones financieras desarrollan sus actividades dentro de un contexto de mercados y precios cada vez más globalizado, competido y hasta cierto punto incierto. Dentro de este contexto intervienen muchos factores que, individualmente contribuyen a que existan variaciones en los niveles en que estos factores se ubican. Estas variaciones es la

fuente primaria de lo que llamamos riesgo. De esta forma, el riesgo puede definirse como "la volatilidad de resultados, generalmente el valor de activos y pasivos de una institución, es decir la posibilidad pérdidas "[8].

Los movimientos que se presentan en variables como tipo de cambio y tasas de interés, crean riesgos no sólo para las instituciones financieras. Es importante destacar que nunca se podrá eliminar totalmente el riesgo, pero definitivamente, la exposición a estos riesgos se puede optimizar y gestionar. Además es importante tener presente que los riesgos y oportunidades van de la mano, la oportunidad de avance no puede existir sin tomar un riesgo por lo que podemos decir que el riesgo en si no es malo, más bien es esencial para el progreso. Entender el riesgo significa que la administración de un banco puede planear las consecuencias de eventos adversos y de esta forma estar mejor preparada ante la inevitable incertidumbre. Asi surge el proceso de administración de riesgos mediante el cual las exposiciones al riesgo se identifican, se miden y se controlan. El proceso de administración de riesgos permite que las instituciones se puedan concentrar en sus áreas de negocio estratégico sin estar expuestas a riesgos de mercado en el proceso.

Hasta hace pocos años la medición de riesgos financieros no contaba con las metodologías apropiadas para que tuviera utilidad como herramienta en la administración de un banco. Sin embargo, el aumento de la volatilidad de las variables financieras ha provocado el nacimiento de toda una serie de metodologías que permiten expresar el riesgo en forma cuantitativa y que responden a preguntas concretas: cuánto afecta en los ingresos un aumento en las tasas de interés; cuánto del capital está en riesgo ante un movimiento en el tipo de cambio, entre otras.

La administración de riesgos es una practica con procesos, métodos y herramientas para manejar riesgos en un proyecto. Esto provee un ambiente disciplinado en la toma de desiciones con el objetivo de valorar continuamente los riesgos e implementar una estrategia.

En primer lugar debe existir un área independiente que sea responsable del diseño e instalación del sistema de administración de riesgos del banco. Esta área tiene como responsabilidad producir informes que indiquen el valor en riesgo producto de los cálculos del sistema y diseñar estrategias de cobertura para compensar los riesgos. Esta área debe ser independiente del área de negocios del banco, esto asegura su integridad, y reportar directamente a la Dirección General.

---

[8] Carácter. Administración de riesgos garantía de salud financiera. Federico Ramírez Silva. 1997. P 10

Una vez que se tiene lo anterior, en coordinación con la administración del banco se debe integrar un sistema de límites y controles internos que permitan proteger el capital del banco ante movimientos inesperados en las variables financieras. En este punto se debe destacar que no se pueden evitar al 100% los efectos de una crisis financiera pero es muy factible que se puedan amortiguar estos efectos si se cuenta con una estructura adecuada de límites y se miden los riesgos en forma precisa.

El cumplir con lo anterior, coloca a cualquier institución en lo que podríamos llamar una actitud pasiva ante el riesgo: simplemente se tienen controlados los niveles de exposición al riesgo. Sin embargo, es importante dar un paso adelante y ser proactivo en la administración de los riesgos. Una de las ventajas de medir adecuadamente nuestros niveles de exposición, es que resulta sencillo proponer y utilizar instrumentos de mercado para cambiar nuestros perfiles de riesgo y de esta forma optimizar nuestra relación riesgo-rendimiento. Es necesario hacer notar que existe toda una gama de productos derivados en los mercados internacionales y que en México tenemos un naciente mercado de estos instrumentos. El medir adecuadamente los riesgos permitirá hacer uso de estos productos en forma prudente sin arriesgar los recursos del banco.

4.2 Base de datos SIR

El SIR (Sistema Integral de Riesgos) es una herramienta que permite medir y cuantificar el riesgo de mercado en la operación diaria de una institución financiera. Proporciona reportes diarios de rubros de riesgo. Por la naturaleza de las operaciones de una institución financiera, el SIR analiza y reporta información en dos grandes rubros de riesgo: Tesorería y Gestión Estructural, aunque se puede tener información tan detallada como se desee.

El sistema SIR para poder desarrollar funciones de gestión de riesgos estructurales, requiere ser alimentado con la información generada por todas las aplicaciones de la red comercial del Banco. Las aplicaciones base del banco se concentran casi en su totalidad dentro de la plataforma IBM Altamira, sin embargo existen algunas aplicaciones funcionando en otros entornos operativos. Aunque el número de operaciones de éstas sea relativamente muy inferior al manejado en Altamira, se debe recuperar esta información para asegurar la consistencia del Sistema de Riesgos contra los balances contables. Tales aplicaciones están distribuidas de la siguiente manera:

Aplicación	Plataforma
Colocación	Altamira DB2
Captación	Altamira DB2
Contabilidad	Altamira DB2
Extranjero	Altamira DB2
Tablas Corporativas	Altamira DB2
Operaciones de Gran Cayman	Oracle (UNIX)
Inversiones de Tesorería	Excel (Windows)
Tarjeta de crédito	Sybase (UNIX)

El sistema SIR administra esta información en una base de datos propia. Dependiendo de la dimensión de la institución financiera, el sistema SIR manejara volúmenes de información variables. Entre mayor sea la dimensión de la institución financiera se requerirá una mayor cantidad de recursos para manejar toda la información, tanto para almacenarlos como para procesarlos.

Dentro de las actividades realizadas dentro del sistema SIR existen actividades que se realizan diariamente, semanalmente y mensualmente. Formalmente se tienen las siguientes actividades:

Actividades diarias:

- Alimentación y captura de todas las operaciones de Tesorería de forma manual y semimanual a través de macros en Excel, pantallas de captura y correo electrónico.
- Ejecución del proceso batch de Tesorería.
- Emisión de reportes de Tesorería.
- Alimentación de las operaciones de Gestión Estructural, a través de interfaces automáticas provenientes del sistema central del banco (arquitectura ALTAMIRA).

Actividades semanales:

- Ejecución semanal del proceso batch de Gestión Estructural.
- Emisión de reportes de Gestión Estructural.

Actividades mensuales:

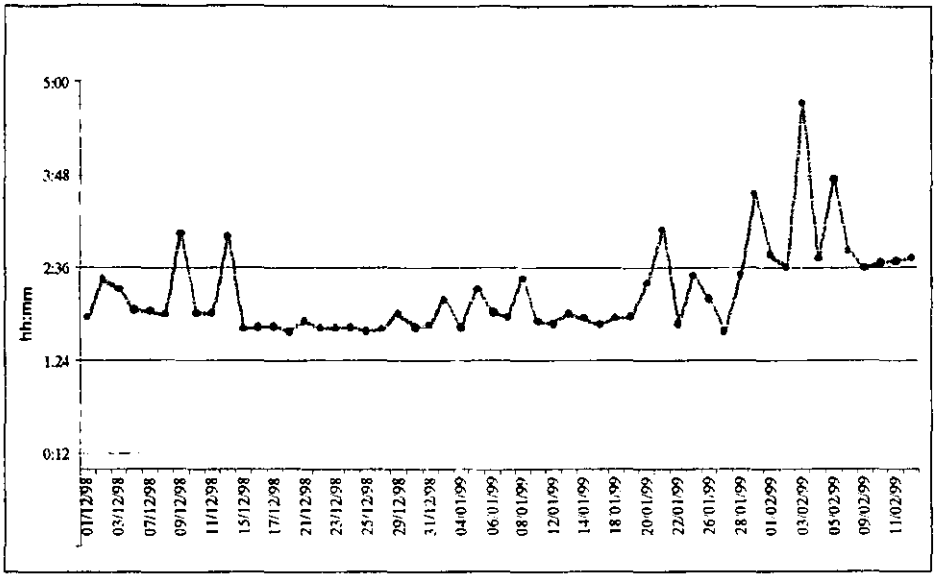
- Carga de tasas y saldos promedio de productos de captación y colocación.
- Carga de productos de tarjeta de crédito.
- Emisión de reportes de contabilidad y cuadre contable.

Informalmente, diariamente se realizan consultas a la base de datos del sistema, tanto por los usuarios finales como por el administrador de base de datos y personal de sistemas a través del lenguaje de consulta DBAccess o por medio de ODBC a través de MSQuery o Access.

Inicialmente sólo se tenia en operación el modulo de Tesorería, teniendo un desempeño aceptable pero que fue decrementando conforme fue aumentando el número de operaciones almacenadas. Cuando el módulo de Gestión Estructural, mucho más grande que el de tesorería, paso al ambiente productivo, el desempeño del sistema decayó enormemente.

En términos generales, la base de datos SIR tiene un desempeño aceptable cuando sólo se tiene una aplicación en proceso, ya sea Tesoreria, Gestión Estructural, emisión de reportes o cualquier otra actividad. Pero los procesos son muy sensibles y en cuanto se tiene alguna aplicación más ejecutándose, el desempeño de la base de datos decae notablemente.

El tiempo de proceso medio del proceso batch de Tesorería es de 2 horas aproximadamente, con algunas otras tareas mínimas en el servidor, tales como consultas a tablas, emisión de reportes de Gestión Estructural. Sin embargo, cuando el sistema tiene dos tareas fuertes ejecutándose, por ejemplo proceso batch de tesoreria e interfaces de alimentación de Gestión Estructural, el desempeño de la base de datos es muy pobre y en ocasiones los recursos son insuficientes para ambos procesos por lo que ningún proceso se completa y ambos terminan en error. La gráfica siguiente muestra un histórico de los tiempos de ejecución que ha tomado el proceso de Tesorería para los meses de Diciembre de 1998, Enero y Febrero de 1999; fechas en que el proceso de Gestión Estructural paso al ambiente productivo. Se observa claramente que la tendencia es que tome mayor tiempo debido al crecimiento de la base de datos. El tiempo fue tomado a través del comando time del sistema operativo UNIX.



De la gráfica de tiempos de ejecución obtenemos que el tiempo promedio en los tres meses analizados es de 2 horas 13 minutos aproximadamente. Mientras que el peor tiempo es de 4 horas 43 minutos, el cual representa un aumento del 100% aproximadamente con respecto al tiempo promedio. Por el contrario, el mejor tiempo lo representa 1 hora 46 minutos, el cual representa una reducción del 20% aproximadamente con respecto al tiempo promedio.

Los tiempos muy disparados de más de tres horas son un claro indicativo que cuando se ejecuto el proceso de Tesorería, estuvo corriendo otro proceso fuerte, casi siempre las interfaces de Gestión Estructural. Sin embargo, las últimas mediciones indican de forma clara que la ejecución del proceso batch de Tesorería se va degradando en tiempo de forma gradual. Todas las mediciones de Febrero, por ejemplo están por arriba de 2 horas 30 minutos. Por lo que el tiempo promedio del proceso de Tesoreria ha ido en aumento entre otras cosas porque la base de datos ha aumentado su volumen lo que hace más difícil al manejador de base de datos buscar, recuperar y escribir información.

Las interfaces automáticas de alimentación al módulo de Gestión Estructural, el proceso batch de Tesorería y el proceso batch de Gestión Estructural son los procesos que consumen mayor cantidad de recursos. Estos procesos en ocasiones se requiere se ejecuten simultáneamente, haciendo casi imposible el acceso a la base de datos.

Por su parte, el proceso de Gestión Estructural ha registrado un tiempo promedio de 27 horas aproximadamente ejecutándose bajo condiciones especiales como lo son en fin de semana y sin ningún otro proceso en el CPU. Cuando se ha requerido ejecutar el proceso de Gestión Estructural durante la semana y conviviendo con otros procesos ha tomado por lo menos 40 horas en su ejecución, lo que representa casi un 50% de aumento con respecto al tiempo de fin de semana. Además, de que durante el tiempo que estuvo en ejecución Gestión Estructural retraso a los demás procesos que se ejecutaron.

Por ultimo, las interfaces automáticas de alimentación a Gestión Estructural tienen un tiempo promedio de 9 horas ejecutándose también bajo condiciones especiales como lo son en las noches y sin ningún otro proceso en el CPU.

La infraestructura del sistema SIR está compuesta de los siguientes elementos:

- Servidor HP/9000 con sistema operativo UNIX HP-UX 8.12 con 512 megabytes de memoria y 10 Gigabytes en disco.
- Manejador de base de datos Informix OnLine Server 8.3.
- Computadoras personales Dell pentium con 16 megabytes de memoria y Windows 95.
- Software para cliente de base de datos *Developer 2000* (ORACLE) en las computadoras personales.
- Software para clientes de base de datos *Informix Cli* en las computadoras personales.
- Programas aplicativos desarrollados en lenguaje C y COBOL en el servidor HP/9000.
- Programas aplicativos, pantallas y reporte en Developer 2000 en las computadoras personales.
- Lenguaje Informix de consulta a la base de datos DBAccess.
- Shells de procesos batch en UNIX HP-UX.
- Computadora con módem conectada al sistema informativo Reuters para alimentación de tasas de interés y tipos de cambio.
- Interfaces manuales de alimentación al sistema provenientes de otros sistemas a través de correo electrónico y macros en Excel.
- Interfaces automáticas de alimentación al sistema desde el sistema principal del banco (ALTAMIRA) a través de descargas en el manejador de base de datos DB2 y transmisión de archivos vía ftp. La información transmitida se carga en la base de datos SIR.

La base de datos tiene la siguiente configuración:

- Esta distribuida en tres espacios: rootdbs, dbstmp y dbsriesgo.
- No se guarda imagen (mirror) de algún espacio (dbspace).
- Utiliza páginas de 2048 bytes.
- Tiene 20 archivos log para registrar información de transacciones y eventos de 400 páginas cada uno. Es decir, un total de 8000 páginas para archivos log.
- Los archivos log se encuentran ubicados en espacio *rootdbs*, en el chunk 1.
- Cuatro tablas manejan por lo menos un campo TEXT.
- Todas las tablas tienen un candado por página (lock).
- Los espacios de la base de datos se distribuyen en tres discos (vg00, vg01 y vg02).
- No existe espacios blobspaces para almacenamiento de campos TEXT o BYTE .

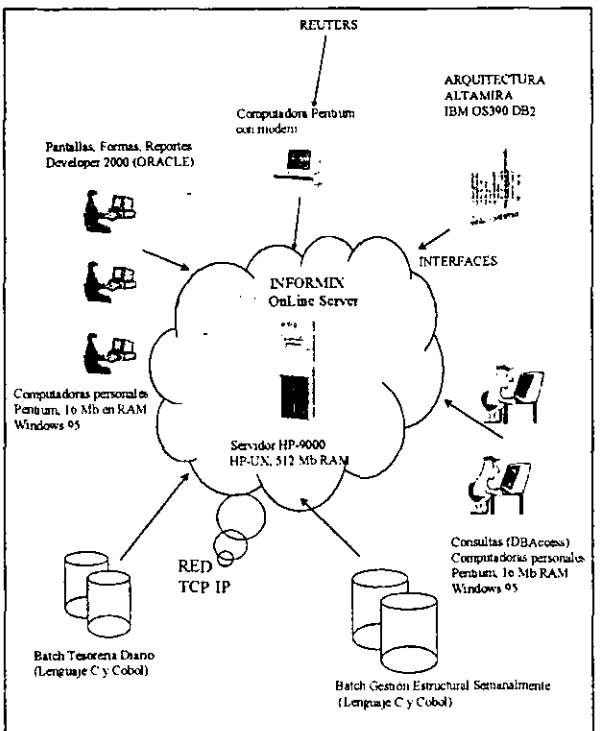


Figura 4.2.1 Infraestructura del sistema SIR

Se tiene aproximadamente 5 Gb de espacio en disco dedicados a la base de datos, los cuales se encuentran divididos en tres espacios: *rootdbs*, *dbstmp* y *dbsriesgo*. El espacio *rootdbs* almacena el diccionario de datos, información de usuarios y eventos registrados en la base de datos. El espacio *dbstmp* se utiliza para áreas temporales y ordenación de información en consultas, así como para tablas temporales. El espacio *dbsriesgo* guarda las tablas de las aplicaciones de Tesorería y Gestión Estructural.

La forma en cómo se encuentran distribuidos los espacios en el disco no presenta una secuencia lógica o estrategia de almacenamiento debido a que inicialmente las necesidades del sistema eran muy diferentes a las que se tienen actualmente ya que la base de datos y el sistema han ido creciendo con el paso del tiempo.

Los espacios de la base de datos tienen la siguiente configuración:

- ***rootdbs*** tiene un tamaño actual de 400,000 páginas, distribuido en dos chunks uno con 250,000 y otro con 150,000 páginas aproximadamente. Esta distribuido en dos discos: vg01 y vg02.
- ***dbstmp*** tiene un tamaño de 200,000 páginas, ubicadas en un solo chunk del mismo tamaño. Se encuentra sólo en el disco vg00.
- ***dbsriesgo*** tiene un tamaño de 2,250,000 páginas, distribuidas en seis chunks con las siguientes dimensiones: tres chunks de 250,000 páginas y tres chunks de 500,000 páginas aproximadamente. Esta distribuido en dos disco: vg01 y vg02.

La distribución de los espacios de la base de datos se resume en las siguientes tablas:

Número	No. Chunks	Propietario	Nombre
1	2	Informix	Rootdbs
2	6	Informix	Dbsriesgo
3	1	Informix	Dbstmp

El cuadro anterior nos indica el número de espacio, cuantos chunks le corresponden a cada espacio, el dueño del espacio y el nombre con el que se le identifica.

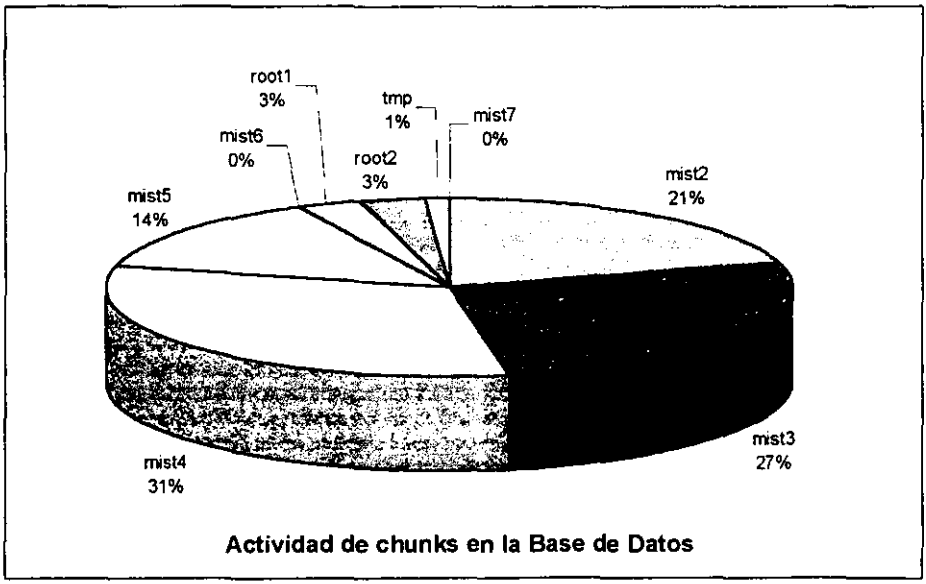
Dbspace	Chunk	Tamaño	Disponible	Usado	Libre
rootdbs	1	250,000	200,995	49,005	80.40%
rootdbs	9	150,000	109,997	40,003	73.33%
Total rootdbs		**400,000**	**310,992**	**89,008**	**77.75%**
dbsriesgo	2	250,000	34	249,966	0.01%
dbsriesgo	3	250,000	49	249,951	0.02%
dbsriesgo	4	500,000	16	499,984	0.00%
dbsriesgo	6	500,000	287,823	212,177	57.56%
dbsriesgo	7	500,000	499,997	3	100.00%
dbsriesgo	8	250,000	249,997	3	100.00%
Total dbsriesgo		**2,250,000**	**1,037,916**	**1,212,084**	**46.13%**
dbstmp	5	200,000	148,107	51,893	74.05%
Total dbstmp		**200,000**	**148,107**	**51,893**	**74.05%**
Total general		**2,850,000**	**1,497,015**	**1,352,985**	**52.53%**

La tabla anterior nos muestra la descripción individual de cada chunk, identificándolo por un número, el tamaño total y el uso o la actividad que tiene el chunk. La tabla esta dividida en secciones correspondientes a cada espacio.

De la tabla anterior se puede apreciar que la carga de los datos no esta balanceada sobre todos los chunks. Mientras que unos chunks están casi a su máxima capacidad (2, 3 y 4), hay otros que permanecen libres (7 y 8). Una idea de cómo esta la distribución del trabajo entre los chunks nos la da el comando onstat -g iof. Este comando nos muestra la actividad de los chunks en un momento en que no hay procesos fuertes ejecutándose.

Gfd	Path	Disco	Operaciones	Lecturas	Escrituras	IO/s
3	/dev/chkmist1	Vg01	571,877	86,217	485,660	0.9
4	/dev/chkmist2	Vg01	3,745,890	3,419,135	326,755	5.6
5	/dev/chkmist3	Vg01	4,827,022	4,574,576	252,446	7.3
6	/dev/chkmist4	Vg01	5,703,088	5,448,686	254,402	8.6
7	/dev/chktmp	Vg00	229,516	116,779	112,737	0.3
8	/dev/chkmist5	Vg02	2,552,324	2,393,026	159,298	3.8
9	/dev/chkmist6	Vg02	5	5	0	0.0
10	/dev/chkmist7	Vg02	5	5	0	0.0
11	/dev/chkmist8	Vg02	520,581	87,052	433,529	0.8

El cuadro anterior nos muestra una visión más detallada de cada chunk. La primera columna muestra la cola a la que esta asociada el chunk, la segunda columna es la ubicación física en el sistema de archivos, la tercera columna indica el disco y las columnas restantes muestran el total de operaciones, lecturas, escrituras y operaciones por segundo realizadas. Una gráfica nos ilustra claramente la situación.



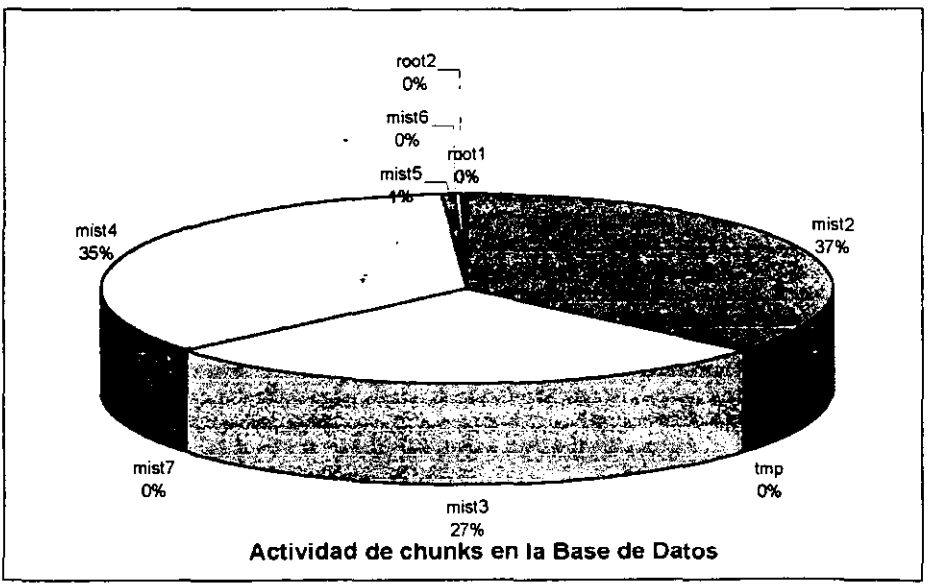
Actividad de chunks en la Base de Datos

La gráfica anterior nos muestra una desproporcionada carga de actividad de los chunks que contienen la información de la base de datos. Por un lado los chunks dedicados al espacio root y temporal tienen una dimensión adecuada y aunque la actividad se carga hacia un chunk, en caso del espacio *rootdbs*, no representa un problema serio. Por otro lado en los chunks dedicados a los datos la actividad se concentran en tres, los cuales realizan todo el trabajo mientras que otros tres prácticamente no tienen actividad. Si se analiza más de allá de los chunks y nos referimos a los discos, es todavía más grave la situación. Toda la actividad esta prácticamente en un solo disco, el vg01; mientras que el disco vg02 tiene una actividad mínima.

Sin embargo, cuando se tiene en ejecución algún proceso fuerte la actividad de los chunks se vuelve aún más desproporcionada. La siguiente lectura fue tomada mientras se ejecutaba el proceso batch diario de Tesorería.

Gdf	Path	Operaciones	Lecturas	Escrituras	IO/s
3	/dev/chkmist1	22,199	10,548	11,651	0.3
4	/dev/chkmist2	9,525,779	9,523,033	2,746	127.8
5	/dev/chkmist3	7,200,146	7,199,604	542	96.6
6	/dev/chkmist4	9,203,963	9,181,619	22,344	123.5
7	/dev/chktmp	5,747	2,637	3,110	0.1
8	/dev/chkmist5	161,071	140,100	20,971	2.2
9	/dev/chkmist6	108,221	95,265	12,956	1.5
10	/dev/chkmist7	1	1	0	0.0
11	/dev/chkmist8	34,379	10,028	24,351	0.5

La siguiente gráfica nos da una idea más clara de la actividad de los chunks y disco:



Actividad de chunks en la Base de Datos

La gráfica anterior nos muestra una desproporción todavía mayor en la actividad de los chunks. Los chunks dos y cuatro son los que prácticamente tienen todo el trabajo. Además, los chunks de mayor trabajo residen en el mismo disco, el vg01, por lo que los problemas de contención que nos provoca esta disposición de discos y chunks es muy alta.

Como consecuencia de la problemática presentada en la operación diaria de la base de datos, se identifica la necesidad de un proceso de afinación de base de datos para un uso óptimo de todos los recursos de la base de datos dentro del cual se definieron los siguientes objetivos del proceso de afinación:

- Lograr la convivencia entre los procesos principales del sistema.
- Disminuir la contención y esperas en los discos, a través de una distribución balanceada en los discos.
- Proporcionar mayor paralelismo posible al momento de hacer consultas y transacciones.

Cabe señalar que se da prioridad a la posibilidad de tener más de un proceso principal ejecutándose en el servidor sobre la rapidez o tiempo de respuesta de los mismos procesos. Debido a la necesidad de tener dos procesos principales simultáneamente y no por eso ambos no terminen o terminen incorrectamente, fue preferible que ambos procesos tomarán más tiempo pero con la certeza de que terminarán de forma normal.

De acuerdo a los objetivos del proceso de afinación las actividades a realizar se definieron de la siguiente manera:

- Cambiar la disposición de espacios en la base de datos. De un solo espacio se formaran dos espacios para las tablas SIR.
- Poner atención a tablas especiales de gran tamaño y actividad, así como aquellas que manejen campos TEXT.
- Afinar actividades de fondo, como respaldos y registro de eventos, que afecten el desempeño.
- Optimizar los programas aplicativos que realizan llamadas a la base de datos.

El primer paso indica cambiar la organización de la base de datos en espacios. En primer lugar se propone tener dos espacios y no sólo uno para los datos, es decir tener espacios *dbsriesgo1* y *dbsriesgo2*. El espacio anteriormente llamado *dbsriesgo* será divido en dos: *dbsriesgo1* y *dbsriesgo2* de la siguiente manera:

Espacio	Disco	Tamaño en páginas
Dbsriesgo1	Vg001	1,000,000
Dbsriesgo2	Vg002	1,250,000

Los cuales aprovecharán la definición existente de los chunks. Los chunks 2,3 y 4 formarán el espacio *dbsriesgo*, y los chunks 6,7 y 8 formarán el espacio *dbsriesgo2*. Los espacios rootdbs y dbstmp permanecerán sin cambio:

Espacio	Chunk	Tamaño
rootdbs	1	250,000
rootdbs	9	150,000
Total rootdbs		**400,000**
dbsriesgo1	2	250,000
dbsriesgo1	3	250,000
dbsriesgo1	4	500,000
Total dbsriesgo1		**1,000,000**
dbsriesgo2	6	500,000
dbsriesgo2	7	500,000
dbsriesgo2	8	250,000
Total dbsriesgo2		**1,250,000**
dbstmp	5	200,000
Total dbstmp		**200,000**
Total general		**2,850,000**

Debido a que la nueva disposición de los chunks en espacios no ayuda mucho, ya que la actividad se seguirá concentrando en los mismos chunks (2,3 y 4) ahora el espacio *dbsriesgo1*, se propone distribuir las tablas del sistema SIR de forma balanceada en ambos espacios. Con los espacios creados para la base de datos, en el espacio llamado *dbsriesgo1* se almacenaran todas las tablas de tesorería. En el espacio llamado *dbsriesgo2* se almacenaran todas las tablas de Gestión Estructural. Esto disminuirá la contención en los discos en consultas y transacciones, además que aumentara la convivencia e independencia de procesos.

Dentro de la aplicación de Gestión Estructural existen tablas de gran tamaño y actividad por lo que se propone implementar una estrategia de fragmentación. Las tablas *histcoco* e *histcoca*, concretamente, son las identificadas con mayor actividad y volumen (Tabla 4.2.1). La tabla *histcoco* tiene un tamaño por renglón de 788 bytes y 208,197 renglones para un total de 160 megabytes aproximadamente. De la misma manera, la tabla *histcoca* tiene un tamaño por renglón de 167 bytes y 3,428,934 renglones para un total de 550 megabytes aproximadamente. Estas tablas almacenan el histórico de contratos de colocación y captación respectivamente y son la base para generar la información. Por lo anterior se propone realizar una fragmentación de estas tablas entre los dos espacios dedicados a los datos. La fragmentación seguiría los siguientes criterios:

1. Un escenario de fragmentación Round Robin.
2. Los índices no seguirían ningún esquema de fragmentación.
3. El esquema para ambas tablas quedaría de la siguiente manera:

```
CREATE TABLE histcoca
        FRAGMENT BY ROUND ROBIN IN dbsriesgo1, dbsriesgo2;
CREATE INDEX idx_histcoca

CREATE TABLE histcoco
        FRAGMENT BY ROUND ROBIN IN dbsriesgo1, dbsriesgo2;
CREATE INDEX idx_histcoco
```

Tabla	Rowsize	Renglones	Tamaño	Tabla	Rowsize	Renglones	Tamaño
Alfpri	89	4,352	207.2	Histsald	47	16,780	430.3
Cashflow	261	16,426	2346.6	Inf_repinde	57	35	1.1
Cashge	86	502,506	22841.2	Inf_repinto	57	8,363	253.4
Cashnmap	261	10,302	1471.7	Inf_repltot	57	-	0.0
Catcuent	107	42,886	2382.6	Limconta	88	4,420	210.5
Clausesp	62	757	25.2	Macrocur	75	290	11.6
Clausest	59	61	1.9	Macrogen	37	1,767	36.1
Codicurv	31	139	2.4	Macrohipo	77	15	0.6
Codirefe	25	221	3.2	Numhipo	84	469	21.3
Coditasas	40	12	0.3	Opermat	85	603	27.4
Coefrefe	19	27,101	311.5	Orden	17	-	0.0
Conliqes	258	20,036	2862.3	Ordena	57	-	0.0
Controlge	60	1	0.0	Paracons	268	9	1.3
Cuamoliq	62	932,122	31070.7	Parmerog	29	21	0.3
Curva00m	62	8	0.3	Posige	111	14,403	847.2
Depurdat	371	-	0.0	Procesos	27	5	0.1
Descrip	57	505	15.3	Rbalance	77	4,339	180.8
Detcuent	57	87,722	2658.2	Repbalan	50	72	1.9
Duraconv	71	147,127	5658.7	Sirhipo	20	643	7.7
Geprodts	3,639	898	3267822.0	Tdcdat	115	273,992	17124.5
Grpcuent	57	87,574	2653.8	Tdcint	13	42	0.4
Grpmist	67	110	3.9	Tesodat	70	3,621	134.1
Histcoca	167	3,428,934	311721.3	Tipohipo	78	9	0.4
Histcoco	788	208,197	104098.5	Tramerog	39	107	2.3
Histcogc	126	8,718	581.2				

Tabla 4.2.1 Tablas de Gestión Estructural

Por otra parte, la actividad de registro de transacciones y puntos de verificación se realizan sobre el espacio *rootdbs*. El espacio rootdbs de acuerdo a las gráficas mostradas anteriormente está funcionando de manera adecuada para tales operaciones, además que

dicho espacio esta libre de la carga de almacenar tablas temporales y ordenación en consultas, ya que se cuenta con el espacio dbstmp. El espacio dbstmp no muestra tampoco una actividad intensa y sólo esta dedicado a tablas temporales, ordenación en consultas y archivos sort.

Los respaldos de la base de datos y de los programas aplicativos se tienen programados de la siguiente manera:

- Respaldos increméntales cada semana programados los Domingos.
- Respaldos totales mensuales programados al inicio de cada mes.

Adicionalmente a las actividades anteriores, se propone un programa constante de mantenimiento de la base de datos. El programa de mantenimiento respaldaría y depuraría las tablas indicadas para evitar se llenen los espacios asignados a los datos del sistema. Especialmente las tablas que utilizan campos TEXT.

Todos los programas aplicativos que realizan accesos a la base de datos se revisaron para verificar la forma en como estaban accesando la base de datos. Lo más importante fue revisar los programas que hacen múltiples accesos a la base de datos. Por ejemplo, un programa que requiere leer las tasas de intereses para doce mil dias, es decir, doce mil consultas a la base de datos, tomaba 30 minutos en ejecutarse. La forma en como trabajaba el programa era la siguiente:

1. Formaba una consulta a través del enunciado SELECT para leer la tasa de interés.
2. Utilizaba la tasa de interés en el cálculo correspondiente utilizando fórmulas básicas en las que intervienen divisiones y multiplicaciones.
3. Se guardaba el resultado en un vector para después insertarlo en la tabla correspondiente.
4. Se volvía al paso uno hasta completar el ciclo de 12000.
5. Al completar las 12000 iteraciones, se insertaba el vector en la base de datos.

La forma de obtener la tasa de interés involucra sólo máximo 10 renglones de la tabla de curvas de tasas *curvactu*. Utilizando el desplazamiento en el tiempo (número de día), de la pendiente y de la tasa base, se obtiene la tasa de interés correspondiente.

La forma actual de leer las tasas de interés es la siguiente:

1. Leer los renglones utilizados para el cálculo de tasas de interés de la tabla de curvas de tasas *curvactu* y guardarlos en un arreglo en memoria con la misma estructura de la tabla.

2. Iniciar ciclo y obtener la tasa de interés directamente del arreglo en memoria.
3. Realizar operaciones correspondientes a la fórmula utilizada.
4. Se guarda el resultado en un vector para después guardarlo en la tabla correspondiente.

Regresar al paso 2 hasta completar el ciclo de 12000.

Con la nueva forma de accesar la base de datos se cambia de 12000 accesos a un máximo de 10 accesos a la base de datos, por lo que el tiempo de ejecución del programa disminuyo notablemente de 30 minutos a 3 minutos, pues teniendo los renglones en memoria es mucho más rápido que estar leyendo datos desde el disco.

Actualmente todos los programas que tienen cantidades considerables de accesos a la base de datos trabajan de la misma manera, procurando realizar todas las operaciones en la memoria de la computadora. Sin embargo, hay ocasiones en que el cambio no ayuda debido a que leer los renglones utilizados de la tabla en la memoria implicaría el mismo número de accesos a la base de datos, caso en el que no se cambiaron los programas aplicativos.

CONCLUSIONES

A través de este trabajo hemos podido observar que aunque la información se ha convertido en el motor que mueve al mundo y nadie puede dudar su importancia, no lo es todo. Tener toda la información en calidad, cantidad, oportunidad y significado incrementa la probabilidad del éxito en un porcentaje muy alto, pero no siempre lo garantiza. Por ejemplo el contar con un DBMS no significa que todos los problemas relativos a información están resueltos, o que se resolverán por el simple hecho de haber invertido en ello. Por otra parte, es importante entender que todos los avances en materia de ciencia y tecnología no tendrían trascendencia alguna si no tuvieran una aplicación práctica en nuestro mundo real, y esto implica que la ciencia y la tecnología también deben tomar cartas en materia de control como: reducir tiempos de proceso, evitar catástrofes, anticiparse a fenómenos naturales, reducir las barreras y distancias en comunicaciones. La combinación de avances científicos-tecnológicos con creatividad e imaginación es lo que realmente mueve al mundo. De lo que podemos deducir que el factor humano es el más importante dentro de esta fórmula, de hecho, la tecnología es creada por el hombre.

La globalización se ha hecho presente día con día en nuestro entorno y ha eliminado todas las barreras de tiempo y espacio, la competencia cada vez es más abierta y descarada por lo que se requiere ser cada día más competitivo para lograr el éxito. Lo anterior provoca una necesidad creciente de información, también en una tendencia creciente, la necesidad por estar informado es cada vez más importante para sobresalir dentro de esta era tecnológica. Actualmente las bases de datos son el principal factor que contribuye a resolver esta necesidad informativa, las bases de datos requieren manejar datos cada vez más complejos y abundantes, y proporcionar resultados más contundentes, por lo que se requiere gran capacidad de análisis en las aplicaciones.

Dentro del ciclo de vida de todo sistema de información, el volumen de los datos almacenados llega a ser un factor determinante en los resultados generados. En ocasiones se desea tener toda la información histórica disponible, pero también se requiere velocidad y precisión en los procesos; en algunos casos extremos llega el momento en que prácticamente es imposible el almacenamiento y la operación de los datos, por lo que las caídas del sistema y resultados de poca calidad se hacen presentes con mayor frecuencia. Por otro lado, las nuevas tendencias requieren volúmenes de información cada vez mayores sin sacrificar la calidad de la información obtenida, los ambientes datawarehouse (almacenes de datos basados en el tiempo) son un ejemplo donde se requiere volumen de

información y velocidad en los procesos. Por lo que se hace necesario su depuración sin afectar el curso de las operaciones cotidianas.

Hay que recordar también que un factor importante en la satisfacción del usuario de un sistema de base de datos es su rendimiento. Si el tiempo de respuesta a una solicitud es demasiado largo, disminuirá el valor del sistema. Como en todas las aplicaciones, debe llegarse a un equilibrio no sólo entre el espacio y el tiempo, sino también entre la información necesaria y las operaciones a realizar para alcanzarla.

El proceso de afinación de una base de datos prácticamente es una fase implícita dentro del ciclo de vida de todo sistema de información y no importa que tan bien se haya diseñado la aplicación, llegará el momento en que se requiera una afinación. El proceso de afinación es una actividad constante y periódica dentro de la vida del sistema, sus principales impulsadores son la retroalimentación y el control. Dentro del proceso de afinación hay una serie de factores a considerar:

- Analizar todos los elementos que intervienen en el resultado final (hardware y software).
- Tener presente que habrá ocasiones en que el proceso de afinación sea insuficiente. Lo que será un indicador de que la infraestructura actual del sistema es insuficiente y que se requiere aumentar la capacidad actual.
- Los objetivos del proceso de afinación deben ir de la mano con los objetivos del sistema de información. En caso contrario serían inalcanzables.

Finalmente el objetivo principal de todo proceso de afinación es el mismo que el de cualquier base de datos, es decir mantener todas las características requeridas en la información.

BIBLIOGRAFÍA

Adoracion de Miguel y Piattini Mario. Conceptos y diseño de bases de datos del modelo E/R al modelo relacional. Adison-Wesley Iberoamericana, S.A. 1993. 989p.

Henry F. Korth y Abraham Silberschatz. Fundamentos de bases de datos. Segunda edición. Mcgraw Hill. España 1993. 739p.

James A. Seen. Análisis y diseño de sistemas de información. Segunda edición. Ed. Mcgraw Hill. México 1994. 942p.

James Martin y Joe Leben. Client/Server Databases Enterprise Computing. Prentice Hall p t r. USA 1995. 352p.

Jeffrey I. Whitten, lonnie D. Bentley y Victor M. Barlow. Análisis y diseño de sistemas de información. Tercera edición. Ed. Irwin. España. 1996. 907p.

Kendall e. Kenneth y Kendall E. Julie. Análisis y diseño de sistemas. Tercera edición. Ed. Prentice Hall. México 1997. 913p.

Kruglinsk, David. Sistemas de administración de bases de datos. Ed. Mcgraw Hill. México 1984.

Performance Guide For Informix Dynamic Server. Versión 7.3. Febrero 1998
Informix Software Inc.

Informix Guide to SQL Sintax Versión 7.2. Abril 1996
Informix Software Inc.

Ramirez, Federico. "Administración de Riesgos garantía de salud financiera". -- BANCRECER CARÁCTER. México. 1997, p. 10-11

www.informix.com

http://mmedia4.fi-b.unam.mx/miembros/bases/apunte/clase1.html

www.genxus.com.br/white/spanish/csarti.htm

Software Engineer Institute. Carnegie Mellon University.
www.sei.cmu.edu