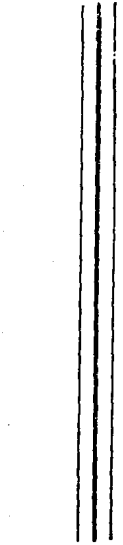
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA



DISEÑO E IMPLEMENTACION DE UN SISTEMA DE AFINACION DE BASES DE DATOS EN OPERACION

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A :

MONICA GUADALUPE VALDIVIA RIOS

DIRECTOR DE TESIS:

ING. RUBEN LIZARDI CERVERA

MEXICO, D. F.

1989

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

I N D I C E

CAPITULO I.	INTRODUCCION.	... 1
CAPITULO II.	CONCEPTOS GENERALES.	... 7
CAPITULO III.	CARACTERISTICAS DE LAS BASES DE DATOS.	... 22
CAPITULO IV.	BASES DE DATOS INCOMPLETAS.	... 44
CAPITULO V.	BASES DE DATOS IMPRECISAS.	... 63
CAPITULO VI.	AFINACION DE BASES DE DATOS	... 111
CAPITULO VII.	DISEÑO E IMPLEMENTACION DE UN SISTEMA DE AFINACION DE BASES DE DATOS.	... 145

I. INTRODUCCION.

Nuestra tesis tiene como objetivo primordial diseñar un sistema que sirva de apoyo al Administrador de la Base de Datos, durante su tarea de afinación de la Base de Datos.

Para implementar el sistema que fungirá como una herramienta de afinación, utilizaremos conceptos novedosos dentro del área de Bases de Datos: **Bases de Datos Incompletas, y Bases de Datos Imprecisas**.

Dedicaremos dos capítulos para definir cada uno de estos conceptos, así como para dar ejemplos que nos permitan una mejor comprensión de los mismos.

Como veremos en su oportunidad, afinar la Base de Datos significa efectuar ciertos cambios sobre la misma, para conseguir un adecuado aprovechamiento de los recursos del sistema, y obtener así un alto rendimiento (performance).

Dividimos el presente trabajo en ocho capítulos, a cuya breve descripción nos avocaremos a lo largo de este capítulo, para mostrar un panorama general y obtener una perspeciva más clara de los pasos que seguiremos para lograr desarrollar nuestro sistema de afinación.

En el capítulo II empezaremos por definir lo que es una Base de Datos, así como un Sistema Manejador de Bases de Datos. Acto

seguido pasamos a explicar las diferentes estructuras de archivos que suelen manejarse, para concluir con las características de rendimiento que ofrece cada estructura, con respectoa las diferentes actividades que se realizan en el archivo. Estas características son mostradas en una tabla.

En el capitulo III, titulado " Características de las Bases de Datos ", nos enfocamos a enunciar las características deseables de cualquier Sistema de Base de Datos en operación, destacando la característica de **AFINACION**, por ser el tema central sobre el que girará nuestro trabajo. En este punto sentamos los conceptos básicos que intervienen en el proceso de afinación, resaltando su importancia como un factor determinisico en la eficiencia y rentabilidad de cualquier Sistema de Base de Datos. En vista de que el encargado de afinar las Bases de Datos es el Administrador de las mismas, consideramos de interés incluir una breve descripción de las diferentes funciones que el Administrador realiza. Como veremos, muchas de estas funciones convergen en el proceso de afinación.

En este capitulo también trataremos superficialmente las herramientas de afinación con que cuenta el Administrador de la Base de Datos, a reserva de que sean tratadas con mayor profundidad en otro capitulo.

Como ya dijimos, para desarrollar un sistema de afinación que sirva de apoyo al DBA, emplearemos un concepto innovador: **Bases de Datos Incompletas**. Dedicamos el capitulo IV, titulado " Bases de Datos Incompletas ", a definir las caracteristicas fundamentales de estas Bases de Datos a través de definiciones y teoremas que nos proporcionan una concepción teórica formal de las

mismas. Incluimos además ejemplos que facilitan la comprensión de la forma en que se manejan las Bases de Datos Incompletas. Se definen así mismo las Bases de Datos Esparcidas como aquellas que nos permiten hacer consultas a Bases de Datos Incompletas.

El capitulo IV titulado " Bases de Datos Imprecisas " trata otro concepto innovador, que nos servirá de base para desarrollar la herramienta de afinacion. Se trata de las **Bases de Datos Imprecisas.** Estas Bases de Datos, como veremos en su oportunidad, son Bases de Datos Incompletas, pero en las que se incluye una interpretación probabilistica.

Estrictamente hablando, las Bases de Datos Incompletas difieren de las Bases de Datos Imprecisas, en que las primeras manejan atributos que pueden tener más de un posible valor para una tupla cualesquiera, mientras que las Bases de Datos Imprecisas son Bases de Datos Incompletas pero en las que se incluye una interpretación probabilisica. Con esta interpretación, a través de un **grado de pertenencia** podemos conocer la **probabilidad** de que un elemento tome un valor determinado dentro de cierto rango, o bien, la posibilidad de que, dependiendo del valor que para cada atributo tenga la tupla, pertenezca a un **conjunto impreciso** definido con anterioridad. Todo esto se verá en su oportunidad con más detalle .

En este capitulo incluimos también algunas definiciones y teoremas que nos proporcionan una descripción formal de los aspectos que manejaremos. Sin embargo, consideramos importante señalar que no se profundizó mucho en formulismos, en vista de que no se requerian para nuestros fines. Sin embargo, y como señalaremos en el capitulo correspondiente, si el lector está

interesado en ahondar en estos temas, puede hacer referencia a la bibliografía que aparece al final de nuestro trabajo.

En virtud de la flexibilidad que nos ofrecen las Bases de Datos Imprecisas para representar cierta clase de información, consideramos que es la manera idónea de representar, en una Base de Datos, los parámetros de afinación de una Base de Datos cualesquiera.

Estos parámetros, como veremos más adelante, por su naturaleza, y en vista de que trataremos de dar un enfoque general para cuestiones de afinación, resultarían imposibles de manejar en una Base de Datos clásica que no admitiese más que un valor para cada uno de los atributos de la tupla.

En vez de ésto, requerimos manejar rangos para cada uno de los valores de los parámetros de afinación, ya que de hecho, durante la afinación, se procede a definir rangos de valores aceptables, que redundan en un alto rendimiento del sistema, y que no encajarían dentro de la rigidez de un esquema clásico.

Por otro lado, nos movemos dentro de un esquema relacional, en vista de que el concepto de imprecisión parte de un enfoque hacia este tipo de Bases de datos, y la literatura existente hasta el momento hace referencia exclusivamente a este modelo. Sin embargo, y en vista del potencial que las Bases de Datos Imprecisas representan para un manejo más eficiente y amplio de la representación de información, no podemos excluir la posibilidad de que en un futuro su enfoque gire hacia los otros modelos conceptuales: modelo jerárquico y modelo de red.

El capítulo V, titulado " Afinación de Bases de Datos ", está dedicado al tema de afinación, definiendo sus niveles y los parámeros que, en forma general, intervienen en el proceso de afinación de la Base de Datos.

Se describen los síntomas que presenta una Base de Datos con un bajo rendimiento (performance), y se establece la secuencia de pasos a seguir para incrementar el rendimiento, consistente en obtener primero toda la información de los parámetros de afinación que el sistema arroja, y a continuación se procede a realizar un análisis de la misma a fin de determinar la estrategia de afinación a seguir.

Se describen las fuentes a través de las cuales el Administrador de la Base de Datos obtiene la información acerca del funcionamiento de la Base de Datos, así como la forma en que puede realizarse la afinación durante cada una de las etapas de vida de un Sistema de Base de Datos.

Es importante destacar que, en vista de la gran diversidad de manejadores de Bases de Datos existentes en el mercado, y a la variedad de configuraciones de equipos en los que funcionan, preferimos plantear los parámetros generales, que resultan comunes a varios manejadores, en vez de enfocarnos a un solo Sistema Manejador de Base de Datos. Así, el sistema que desarrollaremos funcionará para un prototipo de DBMS que funcione en equipos Mainframe. No obstante, esto no excluye de ningun modo la posibilidad de que el sistema pueda adaptarse fácilmente a cualquier manejador en particular, y a cualquier arquitectura. Bastará con modificar los valores comprendidos dentro del conjunto impreciso ALTO RENDIMIENTO de los parámetros de afinación, o

inclusive con modificar los parámetros , agregar nuevos parámetros o eliminar aquellos que se considere que no representan el grado de rendimiento del Sistema de Base de Datos.

Esto se verá mas ampliamente en el capitulo VII, donde se detalla el diseño y conceptos empleados en el desarrollo del **Sistema de afinación**. El programa se hizo en lenguaje Pascal, y como podremos apreciar en su momento, está estructurado de tal forma que resulta sencillo adecuarlo a las necesidades que se planteen. Incluímos en el capitulo VII un listado de este programa documentado así como algunos ejemplos de su ejecución.

Finalmente, en el capítulo VIII daremos las conclusiones derivadas a lo largo de nuestro trabajo, así como las perspectivas a futuro que tienen los **Sistemas de afinación de Bases de Datos**, como una de las múltiples aplicaciones potenciales de las Bases de Datos Incompletas.

II. CONCEPTOS GENERALES

II.1 DEFINICION DE BASES DE DATOS Y SISTEMA MANEJADOR DE BASES DE DATOS (DBMS).

Resulta conveniente definir algunos conceptos referentes a las Bases de Datos en operación, para posteriormente indicar cuales son las características deseables en cualquier sistema en operación.

Podemos definir una Base de Datos como una colección de uno o más archivos, que juntos describen la información total requerida para una aplicación especifica. Una organización puede mantener múltiples Bases de Datos -digamos, por ejemplo, una para mercado y una para investigación.- . En este caso, cada Base de Datos trata las consultas dentro de su dominio.

Los datos se componen de entidades y asociaciones. Una entidad es cualquier objeto distinguible que podemos representar en una base de datos, y una asociación es un vínculo entre dos entidades.

En forma más amplia decimos que una base de datos es una colección de datos interrelacionados almacenados en conjunto sin redundancias perjudiciales o innecesarias; su finalidad es la de servir a una aplicación o más de la mejor manera posible; los datos se almacenan de modo que resulten independientes de los programas que los usan; se emplean métodos bien determinados para incluir datos nuevos y para modificar o extraer los datos almacenados.

Por su parte, un Sistema Manejador de Bases de Datos (DBMS) es un paquete de programas diseñados para definir, crear, manipular y

resumir bases de datos. El objetivo primordial de un DBMS es crear un ambiente en que sea posible guardar y recuperar información de la base de datos en forma conveniente y eficiente.

Esencialmente, todos los DBMS actuales consisten en un conjunto de estructuras de datos integrados o archivos, que residen en algún medio de almacenamiento masivo, tales como discos o cintas, y son manejados por software especializado. Cabe hacer notar que en el presente trabajo al hablar de bases de datos, estamos haciendo referencia a bases de datos almacenadas en disco (pack), a no ser que se especifique algún otro medio de almacenamiento.

La principal contribución de los Sistemas de Manejo de Bases de Datos fue separar las cuestiones acerca de QUE información podía ser almacenada en el sistema, y COMO podría ser almacenada. Las cuestiones referentes al contenido, relaciones y estructura conceptual de la base de datos, comprenden lo que se conoce como nivel lógico o nivel conceptual. Este nivel contiene toda la base de datos en términos de unas cuantas estructuras relativamente sencillas. Aunque es posible que la implantación de las estructuras simples del nivel conceptual requiera estructuras complejas en el nivel físico, no es forzoso que el usuario del nivel conceptual se de cuenta de ello. El nivel conceptual de abstracción lo utilizan los administradores de bases de datos, quienes deciden que información se guarda en la misma.

Por su parte, las cuestiones referentes a la descripción física de la base de datos, como por ejemplo la forma en que los datos son almacenados en los medios electrónicos, corresponden al nivel físico. Este es el nivel mas bajo de abstracción, y en él se describen en detalle las estructuras de datos complejas del nivel más bajo.

La separación entre estos dos niveles hace que los programadores sean más productivos, ya que el desarrollo de aplicaciones se enfoca únicamente en la identificación de los

datos que serán almacenados, las relaciones entre éstos, y las cuestiones que tienen que contestar. El almacenamiento eficiente de esta información en el disco corresponde al DBMS.

Antes del establecimiento de DBMS, cada programa de aplicación debía contener enunciados que especificaran exactamente donde se encontraba un dato en particular en el disco. Por ejemplo, indicar que el nombre del cliente estaba del caracter 5 al 19 en un registro correspondiente al archivo de clientes. De esta forma, cualquier cambio en la base de datos resultaba muy costoso.

En la nueva era de DBMS, la descripción física de un archivo no está contenida en los programas de aplicación, sino en un archivo de disco llamado Definición de Datos Almacenados (SDD). Los programas de aplicación se refieren a los datos por sus nombres lógicos, y las rutinas de acceso de la base de datos usan el SDD para recuperar (o almacenar) los datos apropiados desde (o hacia) la localidad correspondiente en el archivo. Si se requieren cambios en la estructura de la base de datos, se procederá a cambiar el SDD y no los programas de aplicación que accesan la base de datos. En suma, el DBMS elimina mucho del trabajo de programación requerido para mover datos entre los programas de aplicación y los archivos de datos, o incluso entre el video y los programas de aplicación, al facilitar la elaboración de pantallas de captura, la emisión de reportes, y la edición.

De lo expuesto anteriormente se desprende que la conexión entre la definición lógica y la definición física de la base de datos es la función principal del DBMS.

Solo nos faltaría hablar de un tercer nivel presente en las bases de datos, que es el nivel de abstracción más alto, llamado nivel de visión. En este nivel se describe solamente una parte de la base de datos. Aunque en el nivel conceptual se utilizan estructuras mas simples, todavía queda una forma de complejidad que resulta del gran tamaño de la base de datos. Muchos usuarios

de la base de datos no tendrán que ocuparse de toda esta información. Mas bien, necesitarán solamente una parte de la base de datos. Para simplificar la interacción entre estos usuarios y el sistema, se define el nivel de abstracción de visión. El sistema puede proporcionar muchas vistas diferentes de la misma base de datos.

II.2 ESTRUCTURAS DE ALMACENAMIENTO.

La estructura de los archivos y los métodos de acceso constituyen el esqueleto de cualquier sistema de información. Así, el conocimiento de los diferentes tipos de estructuras de archivos nos permite una mejor comprensión de como trabaja el DBMS.

Los archivos almacenados en discos magnéticos pueden ser accesados directamente en cuestión de milisegundos. Una vez accesados, los registros dentro de los archivos del disco pueden ser leidos secuencialmente (como sucede en las cintas magnéticas), o en forma directa, a una razón de 10 a 30 milisegundos. Históricamente, algunos archivos del DBMS estaban limitados a lectura secuencial, pero los archivos de DBMS modernos están usualmente diseñados para acceso directo de registros individuales. Las técnicas de indexación, por ejemplo, proveen este acceso.

Los datos pueden estar organizados en diferentes tipos de estructuras de archivos. Estos tipos son, comúnmente:

a) Pila
b) Secuencial
c) Indexado secuencialmente
d) Indexado
e) Hash
f) Arbol balanceado.

Todos estos tipos pueden ser implementados en sistemas de

almacenamiento en disco en computadoras personales, pero los tipos de archivos indexados, hash y árbol balanceado son las estructuras físicas mas comúnmente usadas en mini y microcomputadoras.

A continuación daremos algunos de los rasgos principales de cada uno de los diferentes tipos de estructuras de archivos que mencionamos, para posteriormente hacer un análisis comparativo en relación a ciertas características de rendimiento (performance).

a) P I L A S.

En la terminología del DBMS, una pila es un archivo en el cual los elementos dentro de un registro no requieren orden o secuencia. Tanto el nombre del campo como su valor son datos en el registro de la pila. Una entrada típica podría ser "EDAD = 43", donde el nombre del atributo y su valor son incluídos. En esta estructura, cada dato y cada registro son de longitud variable. Las pilas son extremadamente flexibles -una ventaja especial cuando el número de elementos con valores varía-. Esta estructura es la más utilizada para transacciones o el logging de cambios hechos a los registros de la base de datos. Presenta varias ventajas, como por ejemplo que es fácil de actualizar, y fácil de agregar nuevos registros al final del archivo. Pueden cambiarse los registros existentes escribiendo un nuevo registro al final, e invalidando o borrando el registro viejo.

También presenta la desventaja de que como la longitud de cada registro es variable, el archivo con estructura de pila solo puede ser leído secuencialmente. Por otro lado, el espacio usado en el almacenamiento en pila es ineficiente, ya que debe almacenar tanto el nombre como el valor para cada entrada. Por otra parte, si la mayoría de los campos están vacíos, esta estructura puede ser más eficiente que otras alternativas. Finalmente, tenemos que toma más tiempo para procesar cada registro, que cuando se trata de registros con campos fijos, porque la computadora debe decodificar el nombre y el valor, mientras que en la mayoría de las otras

estructuras solo requiere decodificar el valor. Principalmente se usan para crear logs de actividades.

b) ARCHIVOS SECUENCIALES.

Tienen dos atributos principales que los distinguen de una pila. Primero, los campos y los registros son de longitud fija. Segundo, está ordenado de acuerdo a su llave primaria. Como la estructura es fija, no se graba el nombre del campo con su valor, puesto que el nombre del campo está implícito en su localidad. Esta estructura es usada ampliamente en sistemas mainframe, y constituyen un punto intermedio entre niveles sofisticados de indexación, y la desorganización total de las pilas. Los archivos secuenciales son eficientes para aplicaciones batch, pero lentos para aplicaciones on-line. Los registros de un archivo secuencial son almacenados en orden físico, de acuerdo a los contenidos de uno de sus campos, llamado **llave primaria**. Un archivo secuencial cuya llave primaria es el nombre del paciente, podría ser ordenado alfabéticamente, de acuerdo a este nombre. A causa de este orden, las búsquedas a través de la llave primaria son relativamente rápidas. Podría no tomarnos más de 15 accesos a disco encontrar un registro en un archivo de 30,000 usando como técnica una búsqueda binaria, mientras que una búsqueda serial a través de 30,000 registros podría requerir un promedio de 15,000 accesos.

Es fácil modificar un archivo con estructura secuencial. Los registros son de longitud fija y sus valores pueden ser sustituidos por otros sin necesidad de cambios estructurales en el archivo. Sin embargo, las inserciones son muy costosas, porque requieren reescribir todos los registros que siguen al registro insertado.

c) ARCHIVOS INDEXADOS SECUENCIALMENTE.

A diferencia de la pila y el archivo secuencial, los cuales están basados en un solo archivo, el archivo indexado

secuencialmente requiere de dos archivos -un archivo de datos y un archivo separado de indices-. El archivo de indices es mucho mas pequeño que el archivo de datos, de tal forma que casi siempre es posible cargarlo en la memoria y buscar rápidamente para encontrar el registro de interés. Estos archivos son ampliamente usados por IBM y otros sistemas mainframe.

Para entender como funciona el archivo secuencialmente indexado, debemos entender como se recuperan los archivos desde el disco de trabajo. Cuando un registro es leido desde disco, es colocado en un buffer de memoria, el cual es generalmente grande para permitir el acomodo de un bloque de disco fisico. Un bloque de disco contiene varios registros lógicos, tales que cuando un archivo es leido secuencialmente, el registro lógico está siempre en el buffer. La lectura directa desde el disco es necesaria solo cuando los registros en el buffer son agotados. El tamaño del bloque afecta severamente el rendimiento del sistema. Asi, en muchos sistemas este tamaño puede ser ajustado para determinar la velocidad de las operaciones de la base de datos.

El almacenamiento en una estructura indexada secuencialmente es similar en muchos aspectos al trabajo de un archivo secuencial simple, pero difiere en que el archivo indexado secuencialmente usa una estructura en bloque del disco. Se guarda el primer registro de cada bloque en un archivo, y se mantiene un indice separado de la llave primaria para el primer registro de cada bloque. Para evitar tener que reescribir el archivo cada vez que se inserte un nuevo registro, se dejan algunos bloques vacios. Estos bloques son conocidos como áreas de overflow, y los nuevos registros son agregados en la siguiente posición disponible en cada area de overflow. Cada registro contiene un campo del apuntador; el orden secuencial es mantenido no por la colocación fisica de registros en el bloque, sino ligando cada registro al siguiente a través del campo del apuntador. Cuando un registro es agregado, el registro precedente en la secuencia de la llave primaria es modificado agregando un valor que apunte al próximo

espacio disponible en el área de overflow, donde el nuevo registro es escrito.

Esta técnica tiene varias ventajas importantes sobre los archivos secuenciales; la mas importante es la habilidad para agregar una nueva entrada on-line sin tener que reescribir el archivo. Así, los archivos indexados secuencialmente siempre reflejan la información disponible más reciente. No tienen el problema de actualizar hasta que la actualización batch sea procesada, como en el caso de los archivos secuenciales.

La búsqueda para un registro dado es rápida si la búsqueda es sobre su llave primaria. Es mucho más rápida que en un archivo secuencial porque el índice reduce el número de accesos a disco necesarios para obtener el registro deseado. Aún cuando se estén accesando registros de overflow, la velocidad es mayor, a menos que existe un número exorbitante de registros de overflow. Los archivos indexados secuencialmente deben estar organizados en intervalos,y este proceso es más difícil que para archivos secuenciales.

La estructura indexada secuencialmente representa un buen balance entre la eficiencia de actualizaciones y la velocidad de acceso, siendo ampliamente usada en sistemas de archivos de computadoras mainframe, como por ejemplo en los sistemas de bases de datos escritos para la serie 370 de IBM.

d) ARCHIVOS INDEXADOS.

Muchas veces es deseable buscar un archivo rápidamente por más de un campo. Por ejemplo, puede desearse accesar al archivo de registros de pacientes por su número del Seguro Social, el número del hospital, y su nombre. Cada campo que se quiere buscar rápidamente, requerirá su propio índice. Alternativamente, puede desearse producir salidas en varios órdenes. Desde un archivo de listas de correo, por ejemplo, se puede desear producir una lista

en orden alfabótico por nombre, o por código postal. Esto podría hacerse en modo batch sorteando los archivos de manera adecuada antes de imprimir cada uno de estos reportes. Pero el sorteo puede tomar más tiempo que la impresión del reporte. En algunos procesamientos de grandes cantidades de datos, el 25 % del tiempo de la computadora es consumido en el sorteo. Los costos de cómputo pueden reducirse si el archivo de listas de correo estuviera indexado por nombre y por código postal.

Un archivo indexado difiere de un archivo indexado secuencialmente en que los registros de datos no tienen un orden particular. Esto significa que las actualizaciones en el archivo principal pueden hacerse rápidamente, puesto que los registros pueden ser colocados al final del archivo sin necesidad de registros de overflow.

Como un registro indexado consiste solo de la llave y el apuntador al archivo principal, los archivos de índices son más pequeños, fáciles de actualizar, y más rápidos para búsquedas que los archivos de datos. Más aún, tener múltiples índices significa que diferentes vistas del usuario pueden ser acomodadas por una estructura de archivo.

Un índice es, en suma, un archivo secuencial sorteado, apuntando a otro archivo de datos. Es algunas veces llamado "archivo invertido" porque es una inversión del archivo principal. Los archivos invertidos pueden proveer búsquedas muy eficientes sobre llaves múltiples.

Inicialmente, los archivos indexados requerían actualizaciones en modo batch, como los archivos secuenciales. Pero ahora la mayoría de los archivos indexados están estructurados como árboles B, y pueden ser actualizados rápidamente en línea. Como los costos de espacio y tiempo crecen para cada índice adicional, solo una pequeña proporción de los campos en una base de datos típica, pueden ser indexados.

e) ARCHIVOS HASH.

Dos estructuras de archivos son mucho más eficientes que los archivos estándares descritos anteriormente. La primera son los archivos hash. Hash es un método de calcular las direcciones de un registro basándonos en el valor de su llave primaria. En vez de buscar directamente la llave primaria, la computadora la transforma en la dirección de una "cubeta de archivo". Teniendo calculada esta dirección de cubeta, la computadora busca a través de los registros en esa cubeta para encontrar uno que contenga la llave primaria de interés.

Este método tiene algunas ventajas importantes sobre otros métodos. Primero, la actualización es mucho más rápida que para archivos indexados porque no se mantienen índices; el índice es calculado cada vez que se requiera una búsqueda o una actualización. Esto evita la necesidad de accesar dos archivos, salvando espacio valioso en disco. A pesar de que la cubeta contiene varios registros, se tienen métodos eficientes para buscar en una cubeta, de tal forma que el tiempo de recuperación es rápido.

Los sistemas operativos usan el método hash extensivamente para almacenar información en memoria RAM. También es usado para archivar registros en DBMS.

El método hash es una técnica poderosa; su uso eficiente requiere de estimaciones precisas del número y tamaño promedio de los registros que serán almacenados. El almacenamiento hash permite registros de longitud variable, pero las cubetas son generalmente de tamaño fijo. Cuando una cubeta no es lo suficientemente grande como para almacenar todos los registros, se crea una cubeta de overflow. Si se subestima el número y el tamaño de los registros, un número excesivo de overflows ocurrirán, y el acceso a la base de datos será lento, pero si se sobreestiman, se

tendrá mucho espacio desperdiciado.

Para minimizar el overflow, cuando los archivos crecen, la función de hash puede ser cambiada, y los archivos tendrán que ser reconstruidos.

Las funciones de hash son un mecanismo rapido para accesar un solo registro de acuerdo a su llave primaria. Las actualizaciones en funciones hash son también eficientes.

f) ARBOLES BALANCEADOS.

Esta estructura es la segunda para lograr un rápido acceso de registros, por llaves arbitrarias. Los árboles son más flexibles que el hash, ya que no requiere conocer el tamaño y el número de registros para un rendimiento óptimo.

La idea de un árbol es una parte importante de la noción de un archivo lineal de índices y apuntadores. Los árboles permiten expresar información en gran profundidad en un campo, sin requerir la misma profundidad de índices en otro.

Una idea más ayudará en una estructura tipo árbol, y es que los brincos en una parte del arbol tendrán la mayoría de las veces más elementos que en otra. Idealmente, sin embargo, un archivo en disco basado en una estructura de árbol, podría ser diseñado de tal forma que el número total de accesos a disco no excederá un número máximo uniforme para todo el archivo. Este objetivo puede alcanzarse si el árbol es "balanceado", de tal forma que ningún brinco es más largo (en profundidad) que otro.

Los árboles B son la estructura universalmente preferida para almacenar los archivos de índices. Los registros de datos pueden también ser almacenados en archivos con estructura de árbol B, y como los archivos hash, los archivos de árboles B pueden también acomodar registros de longitud variable. Los árboles B son la

estructura de almacenamiento usada para la mayoría de los sistemas operativos, que manejan registros con longitud variable.

Los árboles B se comportan en forma interesante, ya que crecen de acuerdo con el tamaño del archivo, pero manteniendo siempre una estructura balanceada, para lo cual se siguen ciertos algoritmos que no explicaremos aquí por rebasar los objetivos de este capítulo.

Con un árbol construido de esta forma, el acceso a cualquier elemento no excederá la profundidad del árbol, que está balanceado por todas sus entradas. En otras palabras, tal árbol puede accesar todos los registros con el mismo grado de eficiencia, y con la garantía consistente en que el peor caso es el mismo para todos los registros de la parte inferior del árbol.

Un área interesante de aplicación de estas estructuras consiste en utilizarlas para encontrar mejores técnicas para obtener una mayor eficiencia en el almacenamiento.

El borrado en un árbol B es mucho más complicado, porque es necesario mantener la estructura balanceada, cuando ocurre la eliminación de un registro determinado.

El borrado de llaves en archivos muy grandes es algunas veces muy complicado, y requiere de muchos accesos a disco. Sin embargo, pese a esto, los árboles B son ampliamente usados en DBMS comerciales, para computadoras personales, por lo que es útil entender como son construidos y mantenidos.

C O N C L U S I O N E S.

Hemos descrito los principales tipos de archivos que se utilizan en sistemas de bases de datos. Algunos tipos son más apropiados que otros para computadoras personales. Otros son apropiados en computadoras personales equipadas con más memoria

principal.

Varios DBMS usan combinaciones de estas estructuras para mejorar la eficiencia de operación.

La tabla siguiente resume las ventajas y desventajas en cuanto a rendimiento de los diferentes tipos de archivos que describimos anteriormente.

CARACTERISTICAS DE RENDIMIENTO (PERFORMANCE) DE DIFERENTES TIPOS DE ARCHIVOS.

	Pila	Secuencial	Indexado sec.	Indexado	Hash	Arbol B
Eficiencia de almacenamiento	Mala	Mejor	Buena	Media	Media	Media
Traer registro por llave	Mala	Media	Buena	Mejor	Buena	Buena
Traer registro siguiente por llave primaria	Mala	Mejor	Mejor	Buena	Mala	Media
Traer registro sin llave	Mala	Media	Media	Media	Media	Media
Traer registro con llave múltiple	Mala	Media	Media	Mejor	Mala	Media
Agregar nuevo registro	Buena	Mala	Media	Media	Buena	Media
Modificar registro	Mala	Buena	Buena	Buena	Buena	Buena
Borrar registro	Media	Mala	Media	Media	Buena	Media

Procesar registro	Mala	Buena	Buena	Buena	Buena	Buena
Leer archivo completo	Media	Buena	Buena	Buena	Mala	Media
Reorganizarse	No lo hace	Media	Media	Media	Mala	Media

III. CARACTERISTICAS DE LAS BASES DE DATOS.

III.1 CARACTERISTICAS DE LAS BASES DE DATOS.

En el contexto de una organización, los datos están relacionados en forma compleja. Generalmente un usuario individual se relaciona solo con pequeños subconjuntos de los datos y relaciones presentes en un momento dato. Sin embargo, la base de datos debe servir a todos los usuarios y a todos los aspectos de sus requerimientos de datos. La base de datos debe usar los recursos de cómputo tan eficientemente como sea posible, y ofrecer una máxima protección a los datos almacenados en ésta. La tarea de diseñar una base de datos es una tarea compleja y con muchas facetas.

Un sistema de bases de datos proporciona a la empresa un control centralizado de sus datos en operación, que constituyen uno de sus activos más valiosos. De ahí la necesidad de que la base de datos instalada y en uso cumpla con ciertas características básicas, que hacen deseable la instalación de estos sistemas en cualquier empresa. Estas características serán descritas a continuación:

a) Redundancia controlada y consistencia.

En la mayoría de las cintotecas anteriores al advenimiento de las bases de datos, hay una sorprendente cantidad de datos duplicados o redundantes. Muchos datos se hallan simultáneamente almacenados en varios volúmenes con distintas finalidades y también con distintas fechas de actualización. En la base de datos se pretende eliminar esta redundancia, aunque en realidad en muchas bases de datos se admite cierta redundancia con el objeto

de reducir tiempos de acceso o simplificar los métodos de direccionamiento. Algunos registros se duplican para facilitar la reconstrucción de la base de datos en caso de daño accidental. Hay pues necesidad de armonizar el grado de redundancia con otras caracteristicas deseables de la base de datos, de modo que es preferible hablar de redundancia controla o de redundancia minima, en lugar de no redundancia.

Por otro lado, la redundancia no controlada acarrea varios inconvenientes. Uno de ellos radica en que, para actualizar por lo menos una parte de las copias redundantes, es preciso recurrir a múltiples operaciones de actualización, y debido a que las distintas copias pueden encontrarse en diferentes estados de actualización, el sistema tiende a proporcionar informaciones contradictorias, cayendo así en una de las caracteristicas no deseables en ninguna base de datos: la inconsistencia. Para eliminarla, a través de una redundancia controlada podemos garantizar que cualquier cambio hecho a una copia se propague a todas las copias existentes de la misma información. Este proceso se denomina propagación de actualizaciones, y abarca las operaciones de creación, supresión y modificación.

b) I n t e g r i d a d .

Cuando una base de datos incluye información utilizada por muchos usuarios, es importante que no puedan destruirse los datos almacenados ni las relaciones existentes entre ellos. Ocasionalmente se producirán fallas de hardware y diversos tipos de accidentes. El almacenamiento de los datos y los procedimientos de actualización deben asegurar que el sistema pueda recuperarse de estas contingencias sin daño para los datos. Toda instalación debe garantizar la integridad de la información que almacena. En esencia, el problema de la integridad es garantizar que los datos de la base de datos sean exactos. La inconsistencia entre dos entradas que representan al mismo "hecho" es un ejemplo de falta de integridad.

c) Procedimientos de validación.

Además de proteger los datos contra posibles problemas sistémicos, el DBA debe incluir ciertos procedimientos de validación o chequeo que habrán de ejecutarse cada vez que se intente una operación de actualización. Así, estos procedimientos de chequeo aseguran que los valores de los datos se ajusten a ciertas reglas prescritas de antemano por la empresa en cuestión.

c) D e s e m p e ñ o .

Las bases de datos diseñadas para ser usadas por operadores de terminal deben asegurar un tiempo de respuesta adecuado para el diálogo entre el hombre y la terminal. Además, el sistema de bases de datos debe tener capacidad para manejar un adecuado caudal de transacciones. En los sistemas en que el volumen de tráfico es reducido, el caudal de transacciones (THROUGHPUT) no tiene por qué imponer muchas restricciones al diseño de la base de datos. En cambio, en los sistemas de alto volumen de tráfico, el caudal de transacciones tiene una gran influencia sobre la organización del almacenamiento físico.

En los sistemas diseñados solo para el procesamiento por lotes, el tiempo de respuesta no tiene una gran significación, y el método de organización física se elegirá teniendo a la vista la mayor eficiencia con ese tipo de procesamiento.

El tiempo de respuesta adecuado en el caso de un sistema con terminales depende de la naturaleza del diálogo entre hombre y terminal. Para ciertas aplicaciones, por ejemplo, se requiere un tiempo de respuesta del orden de dos segundos, entre el instante en que el operador completa su mensaje de entrada en la terminal, y el momento en que aparece en ésta la respuesta.

e) Evolución o reestructuración.

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

f) La interfaz con el pasado.

Los organismos que han estado usando el procesamiento de datos durante algún tiempo han invertido mucho dinero en los programas, procedimientos y datos existentes. Cuando un organismo decide instalar un nuevo software de base de datos, es importante que éste pueda trabajar con los programas y procedimientos existentes y que los datos ya almacenados puedan ser convertidos a las nuevas formas. Esta necesaria compatibilidad llega a menudo a tener el carácter de una severa restricción para pasar al nuevo sistema de base de datos. Es esencial, no obstante, que no se frene el avance de la tecnología de la informática por prestar una atención excesiva a esta compatibilidad con el pasado.

g) Dinamismo.

La base de datos de una empresa no es una entidad estática. Los detalles de los datos almacenados y la manera como estos datos se almacenan, se cambia continuamente. Si un sistema de computación pretende imponer a la empresa una estructura de datos inmutable, se verá condenado a dedicar la mayor parte de sus recursos de programación a la modificación de los programas existentes, más

que al desarrollo de nuevas aplicaciones.

Uno de los objetivos más importantes en el diseño de una base de datos consiste en planearla de tal manera que se pueda modificar sin necesidad de tener que alterar los programas de aplicación en uso. Para lograrlo, el diseño debe prestar especial atención a la siguiente característica de una base de datos: la independencia de los datos.

h) Independencia.

A menudo se habla de la independencia de datos como uno de los atributos más destacados de la base de datos. Esta idea implica que los datos y los programas de aplicación que de ellos se sirven son mutuamente independientes, de manera que unos u otros puedan ser modificados sin afectar a los restantes. Es posible definir la independencia de los datos como la inmunidad de las aplicaciones a los cambios de la estructura de almacenamiento y de la estrategia de acceso, que implica, desde luego, que las aplicaciones no dependen de ninguna estructura de almacenamiento o estrategia de acceso en especial.

En particular, el programador de aplicaciones no debe ser afectado por los cambios que se introduzcan en los datos, en su organización, o en los dispositivos físicos donde se almacenan.

Cuando los primeros sistemas de bases de datos hubieron estado en uso durante cierto tiempo, se patentizó la necesidad de un mayor grado de independencia de datos. La estructura lógica general de los datos se hizo más compleja en muchos casos, y al crecer en tamaño las bases de datos, se hizo inevitable el cambio de la estructura lógica general. Resultó importante que esa estructura general pudiese cambiar sin forzar el cambio de los muchos programas de aplicación que la utilizaban. En muchos sistemas, los cambios de la estructura lógica general de los datos son un modo de ser: la evolución es permanente. Por esta razón,

se necesitan dos niveles de independencia de datos: la independencia lógica y la independencia física.

Por independencia lógica de los datos se entiende que la modificación de la estructura lógica general no afecta a los programas de aplicación. El cambio, desde luego, no debe eliminar ninguno de los datos que el programa necesita.

Por independencia física de los datos se entiende que pueden modificarse la distribución y la organización física de los datos sin afectar ni la estructura lógica general ni a los programas de aplicación.

En un sistema de bases de datos es muy importante la independencia de los datos, entre otras cosas porque el DBA debe tener libertad de modificar la estructura de almacenamiento o la estrategia de acceso (o ambas) en respuesta al cambio de necesidades sin tener que alterar las aplicaciones existentes: por ejemplo, la empresa puede adoptar nuevas normas; las prioridades de las aplicaciones pueden cambiar; nuevos tipos de dispositivos de almacenamiento pueden aparecer en el mercado, etc. Si las aplicaciones dependen de los datos, estos cambios implican modificaciones correspondientes en los programas, lo que requiere esfuerzos de programación que, de lo contrario, podrían utilizarse para crear aplicaciones nuevas. Por ejemplo, en una instalación grande, cerca del 25% del esfuerzo de programación se dedica a actividades de mantenimiento, lo que implica un desperdicio de un recurso muy valioso.

En realidad, así como en la práctica los datos son pocas veces totalmente no redundantes, así también son pocas veces completamente independientes. Algunos sistemas actuales proveen cierta independencia física pero poca o ninguna independencia lógica.

Lo mucho o lo poco que el programador debe conocer acerca de

los datos para acceder a ellos varía de una base a otra. No obstante, la independencia de los datos es uno de los más valiosos argumentos en pro de las bases de datos.

i) Versatilidad para la representación de relaciones.

Diferentes programadores requieren diferentes archivos lógicos. Estos archivos deben derivarse de la misma colección de datos. Existen diversas relaciones entre los rubros de datos almacenados. Algunas bases de datos comprenden un complejo entretejido de relaciones. El método de organización debe tener capacidad para derivar, de datos y relaciones, los archivos lógicos que se requieran.

j) Migración de datos.

Algunos datos se usan con mucha frecuencia y otros solo raramente. Es deseable almacenar los datos de uso frecuente de manera que resulte fácil acceder a ellos. Los datos de uso ocasional se almacenarán, en cambio, de manera más económica (por ejemplo en cinta magnética). Para los datos de uso frecuente pueden usarse discos, de modo que se pueda accesar a ellos en una fracción de segundo. Toda base de datos más o menos compleja tendrá múltiples niveles de facilidad de acceso.

A medida que cambia la popularidad de un conjunto de datos, será conveniente mudarlos dentro del almacén a posiciones mas o menos accesibles, de acuerdo con su actividad. En algunos casos no se mudan los datos, pero sí se modifican los índices que se utilizan para direccionarlos, de manera que puedan ser alcanzados más rápidamente. Este proceso de ajustar el almacenamiento de los datos de acuerdo con su popularidad se llama migración de datos. En algunos sistemas se hace automáticamente. En otros, la operación está a cargo de los programadores del sistema o del administrador de datos. En ocasiones se le considera como parte del proceso de afinación de la base de datos.

k) Cumplimiento de normas establecidas.

Con un control central de la base de datos, el DBA puede garantizar que se cumplan todas las normas aplicables a la representación de los datos. Las normas aplicables pueden comprender las normas de la compañía, de instalación, departamentales, industriales, nacionales o internacionales. Es muy deseable unificar formatos de los datos almacenados como ayuda para el intercambio o migración de datos entre sistemas.

l) Interconectividad.

Cuando un determinado conjunto de datos sirve a una variedad de programas de aplicación, cada uno de estos percibe, en general, diferentes relaciones entre aquellos. En gran medida, es preocupación principal en la organización de la base de datos la representación de las relaciones que existen entre ítems de datos y registros, así como también el cómo y el dónde se almacenan los datos. En las bases de datos previstas para aplicaciones diversas pueden existir múltiples interconexiones entre los datos.

m) Acceso rápido y flexible a la información.

El usuario de una base de datos suele plantear muchos interrogantes acerca de los datos almacenados. En la mayoría de las aplicaciones comerciales actuales, se han anticipado los tipos de averiguación y los datos se han organizado físicamente a modo de poder satisfacerlas con adecuada prontitud. Hay una creciente demanda para sistemas que sean capaces de atender consultas o producir informes que no han sido previstos en la época de diseño. El usuario quiere presentar pedidos espontáneos de información. Las averiguaciones no anticipadas (y algunas anticipadas) hacen necesario explorar algunas partes de la base de datos. La capacidad de hacerlo rápidamente y con diferentes criterios de búsqueda depende mucho de la organización física de los datos. Uno

de los objetivos y característica que debe poseer la organización es lograr capacidad para la búsqueda rápida y flexible.

n) Reserva (Privacidad).

La reserva se refiere al derecho de los individuos y organismos para determinar por si mismos cuándo, cómo y en qué medida se permitirá la transmisión a terceros de la información que les concierne.

Aunque la tecnología necesaria para asegurar este secreto está estrechamente relacionada con la de la seguridad, la reserva es una cuestión que traspasa los límites del centro de cómputo. En gran medida es un problema social. Para preservar el secreto de los datos personales, se necesitan recursos que escapan a la tecnología. La sociedad del futuro, cada vez más dependiente del uso masivo de datos, requerirá nuevos cuerpos legales y nuevos controles sociales si quiere mantener el grado de reserva que se pretende sobre la información personal.

o) Seguridad.

La seguridad de los datos se refiere a la protección de estos contra el acceso accidental o intencional por parte de personas no autorizadas y contra su indebida destrucción o alteración.

La seguridad es un tema extremadamente complejo a causa de sus múltiples y variados aspectos. El analista de sistemas que se hace responsable de la seguridad debe estar familiarizado con todas las particularidades del sistema porque éste puede ser atacado con fines ilícitos desde muchos ángulos. A veces se presta mucha atención a alguno de los aspectos del problema, mientras se descuidan otros.

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

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

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

3) Los datos deben poder ser sometidos a procesos de auditoria. La falta de auditoría en los sistemas de computación ha permitido que se cometan grandes delitos.

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

5) Ningún sistema puede evitar de manera absoluta las intromisiones malintencionadas, pero es posible hacer que resulte muy difícil eludir los controles. Los usuarios de la base de datos deben ser sometidos a un proceso de identificación positiva antes de tener acceso a ella.

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

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

La protección contra accesos no autorizados puede realizarse mediante el uso de usercodes y passwords. La protección de la información es un esfuerzo en todos los niveles, desde los datos individuales hasta la base de datos completa.

p) Simplicidad.

La base de datos debe ser fácilmente comprensible por los usuarios, lo cual significa:

1) El diseño de la base de datos debe ser tan simple como sea posible, lo que ayuda a minimizar ocurrencias de daños accidentales a través de fallas lógicas del programa, y

2) Tener disponible para el programador una visión de la base de datos que se refiera a sus requerimientos particulares. Por ejemplo, solo los datos usados por el sistema sobre el que está trabajando.

Los medios que se utilizan para representar la vista general de los datos deben ser concebidos de manera simple y nítida. No se necesita, en realidad, tanta complejidad como la que se encuentra en ciertas estructuras lógicas de datos.

q) Uso económico de los recursos del sistema.

Se debe usar el mínimo de recursos del sistema de cómputo, en términos de almacenamiento en disco, memoria principal y procesador.

Con el fin de mantener bajo el costo hay que elegir técnicas que minimizan las necesidades totales de almacenamiento. Apelando a estas técnicas, la representación física de los datos en el almacén puede ser muy distinta de la representación que usa el programador. El costo por bit de almacenamiento está disminuyendo rápidamente gracias al progreso tecnológico, mientras que no ocurre lo mismo con el costo de la programación. Hay por tanto una necesidad creciente de mantener sencilla la programación de aplicación, y las organizaciones lógicas de datos deben diseñarse con este objetivo.

r) Respaldo y recuperación.

Un sistema de cómputo, como cualquier otro dispositivo mecánico o eléctrico, está sujeto a fallas. Existen muy diversas causas de

estas fallas, entre ellas la caida de las cabezas lectoras de disco, la interrupción del suministro de energia y los errores de software. En cada uno de estos casos se pierde información de la base de datos. Es por tanto una caracteristica importante que el manejador de la base de datos detecte estas fallas y restaure la base de datos al estado que existia antes de presentarse la falla. Esto se logra normalmente iniciando diversos procedimientos de respaldo y recuperación.

s) Control de concurrencia.

Cuando varios usuarios actualizan la base de datos en forma concurrente, es posible que no se conserve la consistencia de los datos. Así, es necesario que el sistema controle la interacción entre los usuarios concurrentes. Lograr dicho control es una de las tareas del DBMS.

También debe proporcionar un alto rendimiento a usuarios concurrentes múltiples.

t) Afinación.

Cuando es indispensable asegurar el acceso en tiempo real a los datos, los usuarios del sistema se interesan al extremo en el tiempo de respuesta del sistema. En los sistemas de procesamiento por lotes, el interés se traslada al caudal de transacciones que el sistema admite o al tiempo que se tarda en despachar la carga de trabajo. Estos factores dependen del tiempo necesario para acceder a los datos y de la organización física de éstos, así como de su localización en las unidades de almacenamiento. La diferencia entre una organización adecuada y una organización inadecuada se refleja como una gran diferencia en los tiempos de respuesta y en los caudales de transacciones.

Cuando el almacenamiento se prevé para un conjunto específico y entendido de operaciones, es posible optimizar, para esas

operaciones, la organización del almacén y la localización de los datos en él. En muchos casos ni siquiera se sabe como se utilizarán los archivos, cómo se interrogará la base o con qué frecuencia. Resulta por lo tanto ajustar, y hasta cambiar fundamentalmente, la organización del almacén después que el sistema ha entrado en servicio y se han aclarado suficientemente las pautas de uso. En muchos casos, el uso de la base de datos evoluciona continuamente, a medida que mas personas se van familiarizando con ella y se crean más programas de aplicación.

El ajuste de la organización del almacén con el objeto de mejorar su desempeño se convierte así en un proceso continuo.

Este proceso de ajuste de la base de datos se llama afinación (tuning), y es el que nos ocupará a lo largo de nuestra tesis.

En la práctica, la afinación ha conducido a menudo a importantes economías. A veces, éstas han sido tan importantes como para marcar la diferencia entre lo que es una aplicación rentable y lo que no lo es. El responsable de la afinación de la base de datos es el administrador o su grupo, y es importante que éste tenga libertad para introducir los cambios que estime necesarios, sin causar estragos en los programas de aplicación. Este es un aspecto de suma importancia en lo que al procedimiento de afinación se refiere, ya que el administrador de la base de datos, apoyado en un amplio conocimiento de la base de datos que maneja, debe ser sensible a los cambios que los diversos parámetros de afinación producen sobre el rendimiento (performance) del sistema. Asimismo, debe ser capaz de manejarlos de tal forma que se obtenga un máximo rendimiento, considerando los recursos con los que se cuenta, y la carga de trabajo del sistema. Posteriormente ahondaremos más en la función del administrador de la base de datos, dentro del proceso de afinación de la misma.

Por otro lado, tenemos que sin un software apropiado, la

afinación suele incurrir en costos inadmisibles en lo que se refiere al mantenimiento y la prueba de programas. Es por ello que una característica deseable de un DBMS es que cuente con herramientas que faciliten la labor de afinación del DBA.

La correcta afinación tiene dos requisitos. Primero, necesita la independencia física de los datos. Segundo, requiere medios para supervisar automáticamente el uso de la base de datos con el fin de que puedan hacerse los ajustes necesarios. En las futuras bases de datos se incorporarán posiblemente algunos medios para la afinación automática, por ejemplo, en lo que se refiere al aspecto de migración de datos. Se pretenda o no que la afinación sea automática, el sistema debe estar diseñado de modo tal que la facilite.

Por último, y como señalamos anteriormente, resulta importante resaltar las funciones generales del administrador de la base de datos, en vista de que muchas de estas funciones desembocan en el proceso de afinación que nos ocupa. También, en forma introductoria, indicaremos algunas herramientas que ayudan a la tarea de afinación del administrador de la base de datos, y que serán vistas con más detalle en un capítulo posterior.

III.2 ADMINISTRADOR DE BASES DE DATOS.

Como ya dijimos, una de las razones principales para contar con sistemas de manejo de base de datos es tener un control centralizado tanto de los datos como de los programas que tienen acceso a ellos. La persona que tiene este control centralizado sobre el sistema es el administrador de base de datos (DBA: Data Base Administrator).

Las funciones del administrador de base de datos son, entre otras:

a) Decidir el contenido de la información de la base de datos.

Es trabajo del DBA decidir con exactitud qué información se mantendrá en la base de datos, es decir, identificar las entidades de interés para la empresa y la información que debe registrarse acerca de esas entidades.

b) Definición de esquema, es decir la creación del esquema conceptual mediante el lenguaje de definición de datos conceptual. El DBMS emplea la forma objeto (compilada) de este esquema para responder a las solicitudes de acceso. La forma fuente sirve de documento de referencia para los usuarios del sistema.

c) Definición de la estructura de almacenamiento y la estrategia de acceso.

El DBA debe decidir de que manera habrán de representarse los datos en la base de datos y especificar la representación escribiendo la definición de la estructura de almacenamiento (mediante el lenguaje de definición de datos interno: DDL interno). Además, debe especificar la correspondencia asociada entre la definición de la estructura de almacenamiento y el esquema conceptual. En la práctica, el DDL interno o el DDL conceptual incluirán tal vez los medios para especificar esta correspondencia, pero las distintas definiciones deben apreciarse con claridad. Al igual que el esquema conceptual, el esquema interno y su respectiva correspondencia existirán tanto en forma fuente como en forma objeto.

d) Vincularse con los usuarios.

Es responsabilidad del DBA vincularse con los usuarios, garantizar que los datos que requieran estén disponibles, y escribir los esquemas externos necesarios (mediante el lenguaje de definición de datos externo adecuado). Además, debe especificarse la correspondencia entre cualquier esquema externo específico y el esquema conceptual. Cada esquema externo y su respectiva correspondencia existirán tanto en forma fuente como en forma objeto.

e) Definir los controles de autorización y los procedimientos de validación.

Es decir, conceder diferentes tipos de autorización para acceso a los datos a los distintos usuarios de la base de datos. Esto permite al administrador de la base de datos regular cuáles son las partes de la base de datos a las que van a tener acceso diversos usuarios.

Los controles de autorización y los procedimientos de

validación pueden considerarse extensiones lógicas del esquema conceptual. El DDL conceptual incluirá los recursos para especificar esos controles y procedimientos.

El DBA debe también especificar ciertas limitantes de integridad. Estas limitantes se conservan en una estructura especial del sistema que consulta el manejador de la base de datos cada vez que se lleva a cabo una actualización en el sistema.

f) Definir una estrategia de respaldo y recuperación.

Una vez que una empresa adopta un sistema de bases de datos, empieza a depender en forma decisiva de la operación exitosa del mismo. En el caso de que se dañe alguna parte de la base de datos -debido a un error humano, a una falla en el hardware o en el sistema operativo de apoyo -, es esencial poder reparar los datos pertinentes con la mayor brevedad y reduciendo al mínimo posible las repercusiones en el resto del sistema.

El DBA debe definir y poner en marcha una estrategia de recuperación adecuada que incluya, por ejemplo, vaciado periódico de la base de datos en una cinta de respaldo y procedimientos para reponer las partes pertinentes de la base de datos desde la cinta más reciente.

g) Especificación de las limitantes de integridad.

Estas limitantes se conservan en una estructura especial del sistema que consulta el DBMS cada vez que lleva a cabo una actualización en el sistema.

h) Controlar el rendimiento y responder a los cambios de requerimientos.

El DBA se encarga de organizar el sistema de tal manera que se logre un rendimiento que sea "el mejor para la empresa", así como de hacer los ajustes adecuados a medida que los requerimientos cambian. Cualquier cambio en los detalles de almacenamiento y de acceso debe ser acompañado por un cambio respectivo en la definición de la correspondencia con el almacenamiento, de modo que el esquema conceptual se mantenga inmutable.

Sin duda, el DBA necesitará varios programas de utilería para facilitar estas tareas. Tales programas formarían parte esencial de un sistema práctico de bases de datos. En seguida, citamos algunos ejemplos de la clase de programas de utilería que podrían requerirse.

. Rutinas de carga (para crear la versión inicial de la base de datos).

. Rutinas de reorganización (por ejemplo, reordenar la base de datos para recuperar el espacio ocupado por datos obsoletos).

. Rutinas de registro de eventos diarios (anotar cada operación contra la base de datos, junto con la identificación del usuario que realiza la operación y un registro de los estados anterior y posterior a la misma).

. Rutinas de recuperación (restaurar la base de datos a un estado previo después de una falla de hardware o de software).

. Rutinas de análisis estadístico (para ayudar a controlar el rendimiento).

Es posible concebir a los programas de utilería como aplicaciones especiales proporcionadas junto con el sistema (con excepción de las rutinas de registro de eventos diarios, las cuales deben formar parte del DBMS central en sí).

Uno de los recursos más importantes del DBA es el diccionario de datos. El diccionario de datos es una base de datos que contiene "datos acerca de datos", es decir descripciones de otros objetos del sistema. En particular, todos los diversos esquemas (externo, conceptual e interno) se almacenan físicamente en el diccionario, tanto en forma fuente como en forma objeto. Un diccionario amplio incluirá también referencias cruzadas que indican, por ejemplo, qué partes de datos utiliza cada programa, qué informes necesita cada departamento, etc. De hecho, el diccionario puede integrarse incluso en la base de datos que describe, y por tanto, incluir su propia descripción. Debe ser posible consultar el diccionario de la misma manera que cualquier otra base de datos, de modo que, por ejemplo, el DBA pueda describir con facilidad qué programas tienen probabilidad de ser afectados por algún cambio propuesto al sistema.

Como pudimos observar, algunas de las funciones del DBA convergen hacia un objetivo fundamental: AFINAR la base de datos, y por consiguiente obtener el rendimiento máximo del sistema, de acuerdo a los recursos y requerimientos del mismo.

A continuación, hablaremos un poco más acerca del papel que juega el DBA en el proceso de afinación.

Empezaremos por preguntarnos con qué herramientas de afinación cuenta.

En muchas organizaciones, el DBA invierte gran parte de su tiempo resolviendo problemas de emergencia, en vez de prevenirlos.

Para remediar ésto, surgen herramientas tales como el reporte estadístico, que permite automatizar el trabajo mas tedioso del DBA, y mejorar el proceso de toma de decisiones.

Para elaborar esta herramienta, se parte de un ambiente tipico para un DBA. Generalmente, el DBA tiene que dedicar la mayoria de sus recursos en checar el status de los archivos y sus requerimientos de almacenamiento, y no tiene tiempo para la planeación de la base de datos actual. Por consiguiente, la solución es una herramienta consistente en un sistema de monitoreo de archivos.

Después de revisar los requerimientos del usuario, se propusieron algunos componentes especificos del sistema, que junto con un componente de soporte consistente en un procedimiento on-line, permitan al DBA actualizar y mantener los registros de control. Estos registros se usan para controlar la apertura, identificación y cierre de todos los archivos desde los cuales los reportes son generados.

- REPORTE DE USO -

El primer componente del sistema produce un reporte del uso de tablas. Este reporte produce una linea de información por cada archivo en uso. Es desplegado el número de registros de extensión, los registros totales del archivo, el número de registros agregados, y los mensajes de acción. Mensajes con información estadistica como :

" TABLA B DE UTILIZACION MAS DE 75 % -- INCREMENTO TABLA B " son desplegados, y permiten al DBA detectar problemas potenciales antes de que causen un inconveniente mayor. También ayudan a verificar que los archivos tengan el rendimiento esperado. Cuando aparecen mensajes de advertencia para un archivo, se hace necesaria una reorganización del mismo.

- STATUS DE ARCHIVOS -

Esta parte del sistema lista el status de cada archivo, y los mensajes apropiados indicarán los problemas identificados en el mismo. Este reporte es usado por el DBA para verificar el status del sistema después de una caída del mismo, antes de correr tareas batch, y antes de usar el monitor on-line. Hay otro componente en el sistema de monitoreo de archivos, que despliega un análisis del impacto de los archivos. Por ejemplo, se despliega el uso del archivo actual y se estima su nuevo uso, basándose en parámetros de impacto opcionales (por ejemplo, un parámetro calculará el impacto en un archivo si un atributo llave deja de serlo, o viceversa).

También puede calcularse el impacto que se produce en un archivo si un número de registros son agregados o borrados hacia o desde el archivo.

Cuando un sistema de este tipo está corriendo, el DBA reporta una mayor eficiencia en muchas áreas. Por ejemplo, el reporte de uso de archivos produce información sobre el status del almacenamiento, lo cual puede ser usado para planear requerimientos de almacenamiento nuevos. Antes sólo cuando se presentaban problemas en el sistema, se procedía a agregar nuevos drives.

El reporte de status de archivos, que se produce cada mañana, permite al DBA participar en el manejo de negocios, alcanzando extensiones no imaginadas anteriormente.

En un capítulo posterior, nos enfocaremos a desarrollar un sistema de apoyo al DBA, para simplificar el procedimiento de afinación, basándonos en conceptos novedosos, tales como bases de datos incompletas, y bases de datos imprecisas.

Estos conceptos serán explicados en su oportunidad.

En este sistema, el DBA introduce los valores de los parámetros de afinación observados en el sistema, y que se obtienen a partir

de información estadística o del monitoreo de la base de datos, y se obtendrá como salida la evaluación del sistema, incluyendo indicaciones acerca de qué parámetros debe modificar para mejorar el rendimiento de la base de datos.

IV. BASES DE DATOS INCOMPLETAS

IV.1 ANTECEDENTES.

Una base de datos es incompleta si alguno de sus elementos tiene más de un posible valor en sus atributos; si tiene solo un valor por atributo la base de datos es completa. No todas las bases de datos son completas todo el tiempo durante sus ciclos de vida: algunas evolucionan para ser completas al final, mientras que otras permanecen incompletas siempre.

Aún una base de datos incompleta provee información (que aunque parcial es correcta). Por lo tanto, nos gustaría tener un lenguaje de consulta para accesar dicha base. Este lenguaje podía ser similar a uno utilizado en una base de datos completa porque el usuario puede no saber si la base que consultará es completa o no.

Sin embargo, el procesamiento de consultas en una base de datos incompleta resulta ser muy complejo: este tipo de base de datos puede dar muchas respuestas a una consulta, mientras que una base de datos completa dará solo una respuesta. Por lo tanto, la semántica de una consulta es un punto importante en el diseño de las bases de datos incompletas.

El objetivo de este capítulo es estructurar una base de datos incompleta para facilitar el procesamiento de las consultas, asumiendo la semántica de un lenguaje de consultas dado. Sin embargo, los principios que daremos pueden ser aplicados a extensiones de ese lenguaje.

IV.2 BASES DE DATOS CONCEPTUALES.

En una base de datos, los modelos lógicos o conceptuales sirven para dos propósitos:

1) Para que el usuario perciba y use la base de datos.
2) Para el procesamiento de consultas.

Por ejemplo, en el modelo jerárquico, un usuario podría percibir su mundo como árboles; en bases de datos basadas en CODASYL, como redes, y en bases de datos relacionales, como un conjunto de tablas o relaciones. A causa de las diferencias en estos modelos conceptuales, el procesamiento de consultas y el DML (Lenguaje para Manipulación de Datos) relacionado son también diferentes. Por ejemplo, en el modelo jerárquico, el procesamiento de consultas emplea un concepto llamado 'registros' de la base de datos; en el modelo de red, el procesamiento de consultas se basa en conjuntos de indicadores de circulación, y en el modelo relacional se utilizan JOIN, PROJECTION Y SELECT. Claramente, en el uso de un modelo conceptual particular hay una disyuntiva entre amistad para los usuarios, y eficiencia para el procesamiento de consultas.

En general, el modelo jerárquico es más eficiente pero es menos amigable para el usuario; el modelo relacional es más fácil de usar pero menos eficiente, y finalmente el modelo de red cae en medio de ambos. Esto es porque el modelo jerárquico permite controlar más (a través del Modo Acceso) la colocación física de los datos en el modelo conceptual, mientras el modelo relacional no. El remedio tradicional, a nivel conceptual, es proveer facilidades adicionales (por ejemplo apuntadores lógicos) en lo alto de modelos lógicos básicos: por ejemplo, el modo de localización es propuesto en el modelo de red, y segmento, imagen y liga son sugeridos para un sistema de almacenamiento relacional (RSS) de un modelo relacional.

La idea es traer juntos aquellos registros que son potencial y

frecuentemente requeridos. Por consiguiente, la búsqueda a través de la base de datos entera puede ser evitada y así se facilita el procesamiento de consultas. Las ideas anteriores trabajan solo si las consultas que requieren búsquedas extensas son conocidas en cierto grado cuando la base de datos es diseñada; de otro modo, la búsqueda a través de toda la base de datos es necesaria.

En una base de datos incompleta, la mayoría de los valores son rangos; las consultas por lo tanto son también hechas en rangos. Consideremos por ejemplo la consulta hecha a una base de datos personales de alumnos, en la que se desea saber quienes están entre 30 y 40 años de edad. Sin apuntadores que liguen aquellos registros con edades entre 30 y 40 años, la búsqueda a través de la base de datos es inevitable. Pero ¿cómo saber el rango de edades que se consultará? No podemos preveer todos los posibles rangos que serán requeridos; como resultado no podemos construir apuntadores lógicos efectivos para información incompleta.

El procesamiento de consultas en una base de datos incompleta es difícil: cada atributo en una consulta requiere buscar a través de toda la base de datos, y con muchos atributos en una consulta, son necesarios muchos 'backtrackings'.

A continuación, presentamos una formulación primero para bases de datos incompletas, y después discutimos diferentes respuestas posibles para una consulta. Usaremos una base de datos simple con dos atributos, y una consulta de un atributo para demostrar los problemas que surgen en una base de datos incompleta. También describimos cómo una base de datos puede ser reestructurada para facilitar el procesamiento de consultas.

IV.3 BASES DE DATOS INCOMPLETAS.

Sea DB = { x, I, D_i, $B_i(x)$ } una base de datos consistente en:

x = conjunto finito de elementos de la base de datos
I = conjunto finito de atributos
Además, para cada $i \in I$ existe Di:
Di = conjunto no vacío llamado dominio del atributo i;

y para cada $x \in X$ y cada $i \in I$ hay un conjunto $B_i(x) \subseteq D_i$ para representar los posibles valores del atributo i para el elemento x en la base de datos.

Decimos que Bi (x) es completo si tiene solo un elemento, que es un valor específico en el dominio del atributo; será incompleto si es un rango del dominio, y desconocido si puede tomar cualquier valor dentro del rango completo de su dominio. Un valor desconocido es también conocido como valor nulo, y será denotado por 'α'.

Una consulta Q, es una expresión de $Q = q_1 \circ q_2 \circ q_3, \ldots, \circ q_n$, donde $\{q_1, q_2, \ldots q_n\}$ son conocidos como descriptores, y o's son operadores. Un descriptor es un elemento básico en Q que cuestiona solo un atributo en la base de datos. Denotamos un descriptor q como $\langle i, \theta \rangle$, donde $i \in I$ y $\theta \subseteq Di$, lo cual significa:

'Encuentra todos los elementos x's que satisfacen la condición de θ en la base de datos' Por ejemplo, $\langle$edad, [31,40]$\rangle$ es un descriptor que significa: '¿Quiénes están entre 31 y 40 años de edad en la base de datos?'

Los operadores o i's son operadores lógicos tales como AND, OR, NOT. Por consiguiente, $Q = q_1$ AND q_2 OR (NOT q_3) es un ejemplo de una consulta.

Dependiendo de Bi(x) en la base de datos, una respuesta a un descriptor q puede también ser de tres tipos: $\| q \|_c$ es una respuesta completa, $\| q \|_i$ es una respuesta incompleta, y $\| q \|_u$ es una respuesta no conocida. A continuación definimos la semántica para las respuestas de un descriptor $\langle i, \theta \rangle$:

1.- $x \in \| q \|_c$ si $Bi(x) \subseteq \theta$

2.- $x \in \| q \|_i$ si $Bi(x) \cap \theta \neq \emptyset$

3.- $x \in \| q \|_u$ si $Bi(x) = a$

Note que es posible que $x \in \| q \|_c$ aún si $Bi(x)$ es incompleta.

Un operador o en q_1 o q_2 también tiene diferentes significados, dependiendo de si las respuestas para q_1 y q_2 son completas, incompletas o desconocidas. Como un ejemplo, lo siguiente puede ser la semántica para AND y OR:

$$\| q_1 \text{ AND } q_2 \|_c = \| q_1 \|_c \cap \| q_2 \|_c$$

mientras que:

$$\| q_1 \text{ AND } q_2 \|_i \subseteq \| q_1 \|_i \cap \| q_2 \|_i$$

Similarmente:

$$\| q_1 \text{ OR } q_2 \|_c \subseteq \| q_1 \|_c \cup \| q_2 \|_c$$
$$\| q_1 \text{ OR } q_2 \|_i = \| q_1 \|_i \cup \| q_2 \|_i$$

Evidentemente, la semántica de un lenguaje de consulta en una base de datos incompleta es muy compleja, considerando las posibilidades de los operadores: AND, OR, NOT, y muchos otros. Peor aún, un tipo de respuesta de un atributo puede ser combinado con otro tipo de respuesta de otro atributo: por ejemplo, <edad, [35,40]> c OR (<nombre in [x2,x3,x4]> i AND NOT <edad,[31,40]> u).

Todavía hay otras posibles complicaciones: anteriormente, solo manejamos tres tipos de respuestas a un descriptor. De hecho, cuando una base de datos es incompleta, muchas otras respuestas pueden surgir debido a la interpretación de lo que la base de datos incompleta puede significar en un contexto particular. Por ejemplo, en vez de una respuesta incompleta, podemos definir otros dos tipos de respuestas: una que represente una 'respuesta posible', y otra que represente una 'respuesta factible', si

tenemos conocimiento de las probabilidades asociadas con la información incompleta.

No definiremos formalmente la semántica de un lenguaje de consulta, porque esto traspasa el ámbito de este capítulo. Así, la mayoría de las discusiones serán en términos de un operador genérico o sin ningún significado específico. Además, cuando hagamos alusión a una respuesta de un descriptor q, denotada como ‖ q ‖ , nos referiremos al conjunto de los tres diferentes tipos de respuesta; así, ‖ q ‖ = {‖ q ‖ c, ‖ q ‖ i, ‖ q ‖ u } (excepto que se especifique el tipo de respuesta).

Sea Q = q_1 O_1 q_2 O_2 ...., O_n q_n. Si cada descriptor tiene tres respuestas, entonces Q tiene 3^n posibles respuestas. El número de posibles respuestas puede aún ser mas grande si un operador tiene más de un significado. Así, una consulta puede ser especificada como: Q = q_{1x} O_1 q_{2x} O_2 ...., O_n q_{nx}, donde x ∈ {c,i,u}. Esta consulta requiere, explícitamente, una respuesta completa, incompleta o desconocida para cada descriptor, mismas que al combinarse con los operadores nos arrojan solo una respuesta final.

A continuación discutiremos un ejemplo para revisar los conceptos expuestos hasta aquí. La tabla 1 muestra dos bases de datos simples, DB_1 y DB_2, las cuales contienen solo dos atributos: Nombre y Edad. En DB_1, la Edad es ya sea completa, representada por un valor definido, o bien desconocida, si puede ser cualquier valor en el dominio. En DB_2, la Edad puede ser incompleta, si sus valores están en algún rango del dominio, o desconocida. Por simplicidad, asumimos que Nombre es completo en ambas bases de datos. Note que en la tabla 1 los valores de Edad en DB_1 son un subconjunto de los valores que aparecen en DB_2. Así, las respuestas de DB_1 deben ser también subconjuntos de las respuestas de DB_2. Esto es importante porque como una base de datos evoluciona de incompleta a completa, las respuestas podrían tornarse menos vagas y más precisas.

TABLA 1

	DB_1(c+u)	DB_2(i+u)
Nombre	Edad	Edad
x_1	65	[61,70]
x_2	54	[51,55]
x_3	33	[31.34]
x_4	α	α
x_5	38	[36,40]
x_6	47	[45,50]
x_7	59	[31,60]

La Tabla 2 muestra las consultas, y para cada consulta, tres posibles respuestas diferentes de cada base de datos. DB_3 es una abstracción de DB_1 y DB_2, y será discutida más adelante.

TABLA 2

		DB_1	DB_2	DB_3
<edad,[31,40]>	$\|\| q \|\|_c$	$\{x_3, x_5\}$	$\{x_3, x_5\}$	$\{x_3, x_5\}$
	$\|\| q \|\|_i$	0	$\{x_7\}$	$\{x_7\}$
	$\|\| q \|\|_u$	$\{x_4\}$	$\{x_4\}$	$\{x_4\}$
<edad,[33,39]>	$\|\| q \|\|_c$	$\{x_3, x_5\}$	$\{x_3\}$	0
	$\|\| q \|\|_i$	0	$\{x_5, x_7\}$	$\{x_3, x_5, x_7\}$
	$\|\| q \|\|_u$	$\{x_4\}$	$\{x_4\}$	$\{x_4\}$
<edad,[35,45]>	$\|\| q \|\|_c$	$\{x_5\}$	$\{x_5\}$	0
	$\|\| q \|\|_i$	0	$\{x_6, x_7\}$	$\{x_3, x_5, x_6, x_7\}$
	$\|\| q \|\|$	$\{x_4\}$	$\{x_4\}$	$\{x_4\}$
<edad,[35,55]>	$\|\| q \|\|_c$	$\{x_2, x_5, x_6\}$	$\{x_2, x_5, x_6\}$	$\{x_6\}$
	$\|\| q \|\|_i$	0	$\{x_7\}$	$\{x_2, x_3, x_5, x_7\}$
	$\|\| q \|\|_u$	$\{x_4\}$	$\{x_4\}$	$\{x_4\}$

Intuitivamente, tenemos que una respuesta es completa si la base de datos está segura de la respuesta, incompleta si la base de datos tiene solo un conocimiento parcial acerca de la respuesta, y desconocida si no hay información disponible para resolver la consulta formulada.

Las siguientes definiciones son de utilidad:

1. $\|\| q \|\|^* = \|\| q \|\|_c \cup \|\| q \|\|_i \cup \|\| q \|\|_u$, y se conoce como límite superior;

2. $\| q \|_{\bullet} = \| q \|_{c}$, conocido como límite inferior.

Considerando que el propósito de una base de datos es modelar el mundo externo, tenemos que la respuesta de una consulta Q representa una interpretación de parte de ese mundo, representado por Q, a través de la base de datos. El límite superior de una consulta contiene todas las posibles interpretaciones que pueden surgir de la información que se tiene disponible en la base de datos. Por otra parte, el límite inferior contiene los objetos que no podríamos excluir para una interpretación de la consulta Q. Entre el límite inferior y el límite superior, puede existir un modelo para representar el mundo externo desde la base de datos. Sin embargo, no profundizaremos en ésto.

Cabe hacer notar que las consultas de la tabla 2 son muy simples porque se refieren a un solo atributo: la edad. Esto puede complicarse si más de un atributo es combinado por medio de varios operandos, para construir consultas más complejas. Si no se utiliza una estrategia, el procesamiento de consultas necesitaría buscar en todos los elementos de la base de datos, lo cual no es eficiente -aún para consultas simples sobre un solo atributo, y bases de datos pequeñas-.

IV.4 BASES DE DATOS ESPARCIDAS.

Las bases de datos esparcidas se construyen a partir de bases de datos incompletas, para facilitar el procesamiento de consultas involucrado con información incompleta.

La idea básica de una base de datos esparcida es simple. Consideremos una base de datos lógica DB_1 que es amigable para el usuario pero cuya forma de estructurarse no resulta eficiente para el procesamiento de consultas. Esto puede ser por la carencia de apuntadores lógicos, o porque la adición de los mismos podría complicar su simplicidad conceptual. Para facilitar el procesamiento de consultas, sugerimos construir una base de

datos front-end DB_2. Esta base de datos, llamada base de datos esparcida, no necesita ser amigable para el usuario, ni contener informacion tan completa como DB_1. Sin embargo, al usarse limita o restringe los elementos que serán procesados en una consulta. En otras palabras, el procesador de consultas de DB_1 necesita solo enfocarse en los elementos derivados de DB_2, durante el procesamiento de consultas.

DEFINICION IV.4.1.- Sea $\| q \|$ y $\| q' \|$ dos respuestas para un descriptor q desde dos diferentes bases de datos. Decimos que $\| q \|$ está contenido en $\| q' \|$ si $\| q \|_c \cup \| q \|_i \cup \| q \|_u \subseteq \| q' \|$.

DEFINICION IV.4.2.- Sean DB_1 y DB_2 dos bases de datos. Decimos que DB_1 es cubierta por DB_2, o que DB_2 es una base esparcida de DB_1, denotado como $DB_1 \ll DB_2$, si para cualquier descriptor q de una consulta Q, las respuestas derivadas desde DB_1 están contenidas en las respuestas derivadas desde DB_2.

Discutiremos ahora como construir una base de datos esparcida R, para una base de datos $DB = \{ X, I, D_i, B_i(x) \}$. Primero, el dominio de cada atributo D_i de DB es particionado en subconjuntos disjuntos ϕ_{ij}'s: así, $D_i = \phi_{i1} \cup \ldots \cup \phi_{im}$, y $\phi_{i1} \cap \ldots \cap \phi_{im} = \emptyset$.

Segundo, para cada subconjunto del dominio ϕ_{ij}, se construye un subconjunto del dominio esparcido $R(i, \phi_{ij})$, definido como sigue:

$$R(i, \phi_{ij}) = \{ x / B_i(x) = a \text{ o } B_i(x) \cap \phi_{ij} \neq \emptyset \text{ o } B_i(x) \subseteq \phi_{ij} \neq \emptyset \}.$$

Sea $R = \{ X, I, D_i, R(i, \phi_{ij}) \}$, un conjunto creado desde $DB = \{ X, I, D_i, B_i(x) \}$ utilizando el procedimiento arriba descrito. Entonces tenemos lo siguiente:

TEOREMA IV.4.1.

R = {X,I,Di,R(i,ϕij) } es una base de datos esparcida de DB = {X,I,Di,Bi(x) }.

PRUEBA.

Sea q = <i, θ > un descriptor; por definición:

$\| q \|_c = \{ x/Bi(x) \subseteq \theta \}$
$\| q \|_i = \{ x/Bi(x) \cap \theta \neq \emptyset \}$
$\| q \|_u = \{ x/Bi(x) = \alpha \}$

Sea $\| q' \| = \{ x/x \in R(i,\phi ij),\ \phi ij \cap \theta \neq \emptyset \}$

Entonces $\| q \|_c \cup \| q \|_i \cup \| q \|_u \subseteq \| q' \|$

Como un ejemplo,tenemos en la tabla 3 una base de datos incompleta con tres atributos: edad, peso y sangre. En esa tabla, {B,O} para el atributo sangre significa que el tipo se sangre es B u O.

TABLA 3

Nombre	Edad	Peso	Sangre
x1	(41,70)	(6',7')	B
x2	(0,65)	(6',7')	α
x3	(0,60)	α	{B,O}
x4	65	5'	{A,B}
x5	(0,30)	5'10	O

Usando particiones arbitrarias, tenemos una base de datos esparcida en la cual los subconjuntos del dominio esparcido son

los siguientes:

R (edad, [21,30]) = {x2,x3,x5}
R (edad, [31,40]) = {x2,x3}
R (edad, [41,50]) = {x1,x2,x3}
R (edad, [51,60]) = {x1,x2,x3}
R (edad, [61,70]) = {x1,x2,x4}
R (peso, [5',6']) = {x3,x4,x5}
R (peso, [6',7']) = {x1,x2,x3}
R (sangre, {A,O}) = {x2,x3,x4,x5}
R (sangre, {B,O}) = {x1,x2,x3,x4,x5}

IV.4.1 PROCESAMIENTO DE CONSULTAS.

El propósito de una base de datos esparcida es facilitar el procesamiento de consultas. Así, a continuación mostraremos un procesador de consultas 'front-end' basado en bases de datos esparcidas. Las salidas desde este procesador son un subconjunto de los elementos que serían procesados por un procesador de consultas que utilice la base de datos original. Cuando la base de datos es grande, usar la aproximación de bases de datos esparcidas, reduce grandemente el número de elementos que serán revisados en procesos de consulta.

DEFINICION IV.4.3.- Sea $R = \{X, I, D_i, RC_i, \phi_{ij}\}$ una base de datos esparcida, y $q = \langle i, \theta \rangle$ un descriptor. Un conjunto de elementos $\Delta \subseteq X$ es compatible a q si $\Delta = \{x / x \in RC_i, \phi_{ij} \rangle$ donde $\phi_{ij} \cap \theta \neq \emptyset \}$.

DEFINICION IV.4.4.- Sea $R = \{X, I, D_i, RC_i, \phi_{ij}\}$, y $(q_i \circ q_j)$ un par de descriptores relacionados por un operador $\circ$. Entonces $\Delta_{ij} \subseteq \Delta_i \circ \Delta_j$ es un conjunto de pares compatibles desde R si $\Delta_{ij} = \{(x_i, x_j) / (x_i, x_j) \in RC_{i \circ j}, \phi_{ik} \circ \phi_{jl}) \}$ donde $x_i \in \Delta_i$, $x_j \in \Delta_j$ y $RC_{i \circ j}, \phi_{ik} \circ \phi_{jl}) \subseteq RC_i, \phi_{ik} \rangle \circ RC_j, \phi_{jl})$.

Nuevamente, no especificamos el significado de $\circ$ en la definición IV.2.4.

Sea Q={q1,....,qn} un conjunto de descriptores de una consulta, y R una base de datos esparcida. Entendemos como asignación Φ = {A1,A2,....An} de Q desde R a un conjunto de elementos Ai $\subseteq$ Δ_i para cada qi $\in$ Q. Decimos que la asignación Φ está contenida en la asignación Φ' = {A'1, A'2,....A'n} si Ai $\subset$ A'i, 1 $\leq$ i $\leq$ n; en este caso escribimos $\Phi \subset \Phi'$.

La asignación Φ es llamada consistente si, para toda θ .j, tenemos:

$$(\{x\} \circ A_j) \cap \Delta_{ij} \neq \emptyset \text{ , para toda } x \in A_i.$$

Lo siguiente es un algoritmo para procesamiento de consultas 'front-end': iniciamos con la asignación inicial Φ_0 = { $\Delta_1,\ldots,\Delta_n$ }.

Sea Φ k la asignación en la k-ésima aplicación del algoritmo. Para obtener la asignación en el paso (k + 1), descartamos desde cada Ai k cualquier x tal que ({x} o Aj) $\cap$ Aj k = $\emptyset$ para alguna j. En otras palabras, conservamos el elemento x en qi si, para toda qj, existe una x'$\in$ Aj k en qj que es compatible con x; de otra forma, descartamos x. El algoritmo termina cuando: Φ k = Φ k + 1 .

Note que como el número de elementos en cada Ai es finito, este algoritmo siempre termina. También, hay siempre una asignación consistente; en particular, { $\emptyset,\ldots,\emptyset$ } es consistente. Más aún, puede mostrarse que existe mayor consistencia de asignaciones(por ejemplo, una asignación donde otras asignaciones consistentes son subconjuntos de ésta).

EJEMPLO 1..

Sea Q = (q1 AND q2) OR q3 una consulta para la base de datos mostrada en la tabla 3, donde q1 = <edad,[0,40]>, q2 = <peso,['6',ω]> y q3 = <sangre,{A,O}>. De la base de datos esparcida correspondiente, construída previamente, tenemos que

$\Delta_1 = \{x_2, x_3, x_5\}$, $\Delta_2 = \{x_1, x_2, x_3\}$, $\Delta_3 = \{x_2, x_3, x_4, x_5\}$. Asumimos que Δ_i AND $\Delta_j = \Delta_i \cap \Delta_j$ y Δ_i OR $\Delta_j = \Delta_i \cup \Delta_j$; entonces tendremos la asignación $\Phi = \{A_1, A_2, A_3\}$ en la cual $A_1 = \{x_2, x_3\}$, $A_2 = \{x_2, x_3\}$, y $A_3 = \{x_2, x_3, x_4, x_5\}$.

La asignación Φ limita la búsqueda de respuestas para cada descriptor a los elementos contenidos en A_1, A_2, y A_3. Usando esta asignación, procesamos a continuación las respuestas para cada descriptor desde la base de datos en la tabla 3:

$\| \text{edad},[0,40] \|_c = \emptyset$
$\| \text{edad},[0,40] \|_i = \{x_2, x_3\}$
$\| \text{edad},[0,40] \|_u = \emptyset$

$\| \text{peso},[6', \infty) \|_c = \{x_2\}$
$\| \text{peso},[6', \infty) \|_i = \emptyset$
$\| \text{peso},[6', \infty) \|_u = \{x_3\}$

$\| \text{sangre},\{A,O\} \|_c = \{x_5\}$
$\| \text{sangre},\{A,O\} \|_i = \{x_3, x_4\}$
$\| \text{sangre},\{A,O\} \|_u = \{x_2\}$

Note que sin la base de datos esparcida, tendríamos que haber buscado en la base de datos completa de la tabla 3, para cada descriptor. Así, podríamos haber obtenido $\| \text{edad},[0,40] \|_c = \{x_5\}$ y $\| \text{peso},[6', \infty) \|_c = \{x_1, x_2\}$. Como puede observarse, si la base de datos esparcida no se usara, entonces no solo la búsqueda a través de toda la base de datos sería necesaria, sino que el número de objetos obtenido para cada descriptor, y que serán procesados es también mayor.

A continuación están las tres respuestas para Q, usando la semántica antes definida para AND y OR:

$\| Q \|_c = \{x_5\}$
$\| Q \|_i = \{x_3, x_4\}$

$$\| Q \|_u = \{x_2\}$$

IV.4.2 BASES DE DATOS ESPARCIDAS ESTRICTAMENTE.

En la parte anterior, discutimos las ideas básicas de las bases de datos esparcidas y el procesamiento de consultas para información incompleta. Demostramos que una base de datos esparcida puede ser usada para restringir el número de elementos a ser procesados. Para obtener las tres respuestas finales, tuvimos que recurrir a la base de datos original. De hecho, podemos también mantener los tres tipos de elementos en una base de datos esparcida.

DEFINICION IV.4.5.- Sea $\| q \|$ y $\| q \|'$ dos respuestas para un descriptor q desde dos diferentes bases de datos. Decimos que $\| q \|$ está contenida estrictamente en $\| q \|'$ si se cumplen las condiciones siguientes:

1. $\| q \|_c \cup \| q \|_i \subseteq \| q \|'_c \cup \| q \|'_i$
2. $\| q \|_c - \| q \|'_c \subseteq \| q \|'_i$
3. $\| q \|_u = \| q \|'_u$

DEFINICION IV.4.6.- Sean DB_1 y DB_2 dos bases de datos. Decimos que DB_1 está cubierta estrictamente por DB_2, o que DB_2 es una base de datos esparcida estrictamente de DB_1, denotado como $DB_1 \ll DB_2$, si para cualquier descriptor q de una consulta Q, las respuestas derivadas desde DB_1 están contenidas estrictamente en las respuestas derivadas desde DB_2.

La manera de construir una base de datos esparcida estrictamente R para una base de datos DB = $\{X, I, D_i, B_i(x)\}$ es similar a lo que discutíamos anteriormente. Primero, el dominio de cada atributo D_i de DB es particionado en subconjuntos disjuntos ϕ_{ij}'s: así, $D_i = \phi_{i1} \cup \ldots, \cup \phi_{imi}$ y $\phi_{ik} \cap$

ϕ_{im} = Ø. Después, para cada subconjunto del dominio ϕ_{ij}, construimos un subconjunto del dominio esparcido estrictamente RCi, ϕ_{ij}), definido como sigue:

$$RCi, \phi_{ij}) = \{x, x-, x^ / x \in RCi, \phi_{ij})c,\ x^ \in RCi, \phi_{ij})u,\ x- \in RCi, \phi_{ij})i - RCi, \phi_{ij})c\}$$

donde:

$$RCi, \phi_{ij})u = \{x \in X / BiCx) = a\}$$

$$RCi, \phi_{ij})i = \{x \in X / BiCx) \cap \phi_{ij} \neq \emptyset\}$$

$$RCi, \phi_{ij})c = \{x \in X / BiCx) \leq \phi_{ij}\}$$

Note que cada partición mantiene tres tipos diferentes de elementos: completos, denotados por x; incompletos, por x-; y desconocidos, por x^.

EJEMPLO 2.-

El presente ejemplo muestra la construcción de una base de datos esparcida estrictamente desde DB_2, en la tabla 1. Esta base de datos, llamada DB_3 en la tabla 2, es usada para procesar cuatro consultas en dicha tabla. En la construcción de DB_3, usamos arbitrariamente como particiones de edad: [21,30],[31,40],...,[61,70], lo cual mostramps a continuación:

$$RCedad, [21,30]) = \{x_4\}$$
$$RCedad, [31,40]) = \{x_3, x_5, x_7, x_4\}$$
$$RCedad, [41,50]) = \{x_6, x_7, x_4\}$$
$$RCedad, [51,60]) = \{x_2, x_7, x_4\}$$
$$RCedad, [61,70]) = \{x_1, x_4\}$$

Las respuestas para las cuatro consultas basadas en DB_3 son mostradas en la tabla 2, usando el siguiente algoritmo:

ALGORITMO.

1. Encontrar R(i,ϕ_{ij}) desde R, donde $\phi_{ij} \cap \theta \neq \emptyset$

2. $\| q \|_c = \{ x / x \in R(i,\phi_{ij}) \text{ y } \phi_{ij} \subseteq \theta \}$

3. $\| q \|_i = \{x, x^- / x, x^- \in R(i,\phi_{ij}) \text{ y } \phi_{ij} \cap \theta \neq \emptyset \} - \| q \|_c$

4. $\| q \|_u = \{x^n / x^n \in R(i,\phi_{ij}) \text{ y } \phi_{ij} \cap \theta \neq \emptyset \}$

Note que las respuestas de las consultas desde DB_2 están contenidas estrictamente en las respuestas desde DB_3. Como resultado, solo los elementos en las respuestas incompletas de DB_3 necesitan examinar nuevamente DB_2 para obtener las respuestas de DB_2. Por ejemplo, en la tabla 2, desde que conocemos que x_6 está en la respuesta completa y x_4 en la respuesta desconocida, solo necesitamos checar x_2, x_3, x_5 y x_7, esto es, los elementos de la respuesta incompleta, nuevamente en DB_2.

C O N C L U S I O N

Puede parecer que las bases de datos esparcidas son similares a las estructuras que manejan apuntadores lógicos, y son usadas en otros modelos de bases de datos para facilitar el procesamiento de consultas. Después de todo, nosotros podemos usar apuntadores lógicos para establecer las particiones del dominio. Existen, sin embargo, diferencias fundamentales: Primero, los apuntadores usualmente son utilizados para conectar elementos de la base de datos, siguiendo ciertos modelos de aplicación del mundo real (por ejemplo, todos los estudiantes de cierta clase). En las bases de datos esparcidas, las particiones del dominio son hechas en forma arbitraria. Una implicación importante de esta arbitrariedad es que pueden ser introducidas

diferentes interpretaciones para el procesamiento de consultas. Por ejemplo, las particiones que elaboramos en el ejemplo anterior colocan a x_7 en tres diferentes particiones. Si decimos que x_7 está 70% en [31,40], 80% en [41,50] y 50% en [51,60], introducimos imprecisión en la base de datos esparcida. Por otra parte, si decimos que x_7 está 70% en [31,40], 20% en [41,50] y 10% en [51,60] (donde la suma de las probabilidades es igual a la unidad), entonces introducimos una interpretación probabilística a la base de datos esparcida. En contraste, nosotros normalmente no podríamos crear un apuntador con probabilidad incierta.

Note que otras particiones son posibles, y entonces las interpretaciones podrían ser diferentes. Como mencionamos anteriormente, muchas posibles respuestas caen entre el límite superior y el límite inferior. Así, una partición en particular hace la interpretación de una consulta más significativamente que las otras, y consecuentemente el procesamiento de consultas se vuelve más eficiente. Son muchas las interpretaciones posibles de una base de datos esparcida, lo que las hace particularmente atractivas para ayudar al procesamiento de consultas en bases de datos incompletas.

Actualmente se está investigando la posibilidad de agregar algunos modelos de interpretación, basados en lógica imprecisa y/o probabilidades, a las bases de datos esparcidas. Esto podría aligerar mucho el peso de interpretar una consulta desde la base de datos original, la cual está básicamente organizada para los usuarios, utilizando la base de datos esparcida, la cual está organizada para procesar información imcompleta.

Algunos autores han apuntado que en las bases de datos incompletas, no hay nada que hacer con el concepto de imprecisión. Esto puede ser visto desde DB_2, donde se conoce que x_2 tiene un valor dentro del rango [51,55]. Esta información es incompleta, pero no obstante es completamente correcta. En las bases de datos esparcidas, la imprecisión se vuelve natural porque no tenemos forma o límites claros para crear particiones

para información incompleta.

Otra diferencia es que las bases de datos esparcidas son estructuras lógicas adicionales, y son más que simples apuntadores a organizaciones físicas. Así, desde las bases de datos esparcidas podemos fácilmente inferir otras consultas, mientras que el hecho de añadir diferentes significados a los apuntadores lógicos podría desvirtuar la simplicidad conceptual de un modelo lógico.

Finalmente podemos decir que ya sea la base original DB_1, o la base de datos esparcida (estrictamente) DB_2, pueden ser usadas directamente para realizar una consulta. En otras palabras, estas dos bases de datos coexisten en una base de datos: DB_2 es simplemente un modelo auxiliar a DB_1. Por supuesto, podemos construir más bases de datos esparcidas $DB_2 \ll (\ll) DB_3 \ll (\ll) \ldots$, si es necesario.

V. BASES DE DATOS IMPRECISAS

V.I INTRODUCCION.

Desde que Codd propuso el modelo de datos relacional, los sistemas de bases de datos relacionales han sido extensivamente estudiados y comercializados. Este modelo de datos usualmente utiliza solo datos bien definidos y nada ambiguos. Sin embargo, los datos de aplicaciones del mundo real son muchas veces parcialmente conocidos (incompletos) o imprecisos. Por ejemplo, en vez de especificar que la altura de Juan es de 192 cm. se puede decir que mide alrededor de 190 cm. o simplemente que Juan es alto. Tales enunciados contienen información acerca de la altura de Juan y pueden ser útiles para responder ciertas consultas.

Para obtener más significado de los datos, han sido propuestas varias extensiones del modelo relacional clásico. En algunas de estas extensiones, una variedad de 'valores nulos' han sido introducidos a modelos desconocidos o valores de datos no aplicables. Se han hecho intentos para generalizar los operadores del álgebra relacional para manipular tales modelos de datos extendidos.

La teoría de conjuntos imprecisos, y la lógica imprecisa propuesta por Zadeh, provee un requisito matemático para tratar con valores de datos extendidos. Recientemente, algunos autores han estudiado bases de datos relacionales a la luz de la teoría de conjuntos imprecisos, con el objetivo de amoldarse a un rango más amplio de requerimientos del mundo real, y proveer interacciones hombre-máquina más estrechas. En algunos de estos propósitos, las operaciones del álgebra relacional clásica tales como JOIN, PROJECTION, etc., han sido apropiadamente extendidas, y los problemas referentes al diseño de un lenguaje de consulta y

a la evaluación de consultas han sido examinados.

Como se extendió el modelo de datos relacional clásico para manejar información imprecisa, podría ser necesario considerar los límites de integridad que pueden involucrar conceptos de imprecisión. De hecho, los límites de integridad imprecisos, tales como 'salarios de la mayoría de los empleados igualmente calificados podrían ser mas o menos iguales', surgieron naturalmente en las bases de datos imprecisas.

Podemos definir una Base de Datos imprecisa, como un conjunto de datos que pueden ser parcialmente conocidos para uno o más atributos de la Base de Datos.

Para tratar con límites de integridad imprecisos, Zadeh ha introducido el concepto de restricción de una relación imprecisa, debida a una proposición imprecisa. Así, los límites de integridad imprecisos pueden ser representados por proposiciones imprecisas apropiadas. Más aún, la restricción de una base de datos relacional imprecisa, debida a un conjunto de límites de integridad imprecisos puede ser calculada combinando las proposiciones imprecisas asociadas con estos límites de integridad, de acuerdo a las reglas del cálculo impreciso. De hecho, examinaremos diferentes tipos de relaciones imprecisas y restricciones de tales relaciones debidas a límites de integridad imprecisos.

Nuestro objetivo principal es extender la teoría de diseño de bases de datos relacionales, al dominio impreciso, por medio de la definición adecuada de dependencia funcional imprecisa (dfi). Para tal objeto, en el punto 2 trataremos algunas definiciones y conceptos básicos de la teoría de bases de datos relacionales clásica. En los puntos 3 y 4 introducimos algunas definiciones básicas de teoría de conjuntos imprecisos, y discutiremos las relaciones imprecisas y los límites de integridad imprecisos. En el punto 5, examinaremos la dependencia funcional imprecisa (dfi) y las reglas de inferencia asociadas. Por último, en el punto 6 nos referiremos a la pérdida de unión de relaciones imprecisas en

presencia de dependencias funcionales imprecisas.

V.2 MODELO RELACIONAL CLASICO.

Atributos son símbolos tomados desde un conjunto finito $U = \{A_1, A_2, \ldots, A_N\}$. Cada atributo A_i tiene asociado un dominio, denotado por $dom(A_i)$, el cual es el conjunto de posibles valores para ese atributo. Elementos del $dom(A)$, $dom(B)$, y $dom(C)$ son denotados usualmente por a, b, c, respectivamente, con posibles sufijos. Para un conjunto de atributos X, un valor X es una asignación de valores a los atributos de X desde su dominio. Usaremos las letras A, B,... para atributos simples y las letras X, Y, para conjuntos de atributos. La unión de dos conjuntos X e Y es escrita como X Y. No se hace distinción entre un atributo sencillo y el conjunto { A }.

Una relación r con atributos $\{A_1, A_2, \ldots, A_n\}$ es un subconjunto del producto cartesiano de $dom(A_1) \times \ldots \times dom(A_n)$. Un esquema de relación sobre $\{A_1, \ldots, A_n\}$ será denotado por $R(A_1 \ldots A_n)$ o R. Una relación r es considerada una instancia de un esquema de relación R. Los elementos de la relación son llamados tuplas o renglones. Una tupla es usualmente representada como una cadena de valores asociados con los atributos; por ejemplo, ac es una tupla de una relación r sobre $R(AC)$. Si t es una tupla en la relación r de $R(U)$, y A es un atributo en U, entonces $t[A]$ es la componente A de t. Similarmente, para el conjunto de atributos $X \subseteq U$, usamos la notación $t[X]$ para denotar la restricción de t a X; por ejemplo si $t = abc$, entonces $t[AC] = ac$.

Hay dos operaciones sobre las relaciones que resultan de interés: proyección y unión natural. La proyección de una relación r de $R(XYZ)$ sobre el conjunto de atributos X se obtiene tomando la restricción de las tuplas de r a los atributos en X y eliminando tuplas duplicadas que queden. Esta operación es usualmente denotada por:

$$\prod_X(r) = \{ t[X] / t \in r \}$$

Como el simbolo $\prod$ se usa en la literatura sobre conjuntos imprecisos para denotar la posibilidad de distribución, en lo sucesivo utilizaremos la notación $P_X(r)$ para la proyección de r sobre X.

Sean r_1 y r_2 dos relaciones de R(XY) y R(XZ), respectivamente. La unión natural $r_1 \omega r_2$ es una relación sobre R(XYZ) definida por:

$$r = r_1 \omega r_2 = \{ t / t[XY] \in r_1 \text{ y } t[YZ] \in r_2 \}$$

Entre los tipos diferentes de dependencia de datos, la dependencia funcional requiere una mención especial. De hecho, la importancia de la dependencia funcional en el diseño de sistemas de bases de datos relacionales ha sido reconocida desde que Codd introdujo el modelo de datos relacional. Formalmente, una dependencia funcional (df) es un enunciado, X --> Y, donde X e Y son conjuntos de atributos. Una relación r satisface la dependencia funcional (df) si para toda t_1 y t_2 en r, si $t_1[X] = t_2[X]$ implica que $t_1[Y] = t_2[Y]$.

En otras palabras, dada una relación R, el atributo Y de R es funcionalmente dependiente del atributo X de R si y sólo si cada valor de X en R tiene asociado a él exactamente un valor de Y en R (en cualquier instante).

Nótese que en la definición de dependencia funcional no existe el requisito de que un valor dado de X aparezca solo en una tupla de R. Esto aparece contemplado en la primera definición.

V.3 MODELO DE DATOS RELACIONAL IMPRECISO.

V.3.1 CONJUNTO IMPRECISO.

Sea U un conjunto clásico de objetos, llamado el universo. Un elemento de U es denotado por u.

DEFINICION V.3.1.

Un conjunto impreciso F en un universo U es caracterizado por una función de pertenencia:

$$\mu_F : U \dashrightarrow [0,1]$$

donde $\mu_F(u)$ para cada $u \in U$ denota el grado de pertenencia de u en el conjunto impreciso F.

Podemos definir el grado de pertenencia de un elemento u hacia un conjunto impreciso dado, como aquel parámetro que nos indica la probabilidad de que dicho elemento pertenezca al conjunto. Así, un grado de pertenencia igual a cero indica que el elemento no pertenece al conjunto, mientras que un grado de pertenencia igual a uno, indica que con certeza, el elemento u pertenece al conjunto.

Siguiendo las notaciones usadas en la teoría imprecisa de conjuntos, escribimos:

$$F = \{\mu(u_1)/u_1, \mu(u_2)/u_2, \ldots, \mu(u_n)/u_n\}$$

donde $u_i \in U$, $1 \leq i \leq n$. Note que un subconjunto clásico A de U puede ser visualizado como un subconjunto impreciso con función de pertenencia μ_A tomando valores binarios; por ejemplo

$$\mu_A(u) = 1 \quad \text{si } u \in A$$
$$= 0 \quad \text{si } u \notin A$$

Las operaciones usuales de teoria de conjuntos (tales como unión, intersección y complemento) han sido extendidas para tratar con conjuntos imprecisos. Sean A y B dos subconjuntos imprecisos en un universo, con funciones de pertenencia μ_A y μ_B, respectivamente. Entonces las funciones de pertenencia de $A \cup B$, $A \cap B$ y $\neg A$ (complemento de A), son dadas a continuación:

$$\mu_{A \cup B}(u) = \max(\mu_A(u), \mu_B(u))$$
$$\mu_{A \cap B}(u) = \min(\mu_A(u), \mu_B(u))$$
$$\mu_{\neg A}(u) = 1 - \mu_A(u)$$

Basados en estas definiciones, la mayoria de las propiedades de las operaciones de conjuntos clásicas, tales como las leyes de De Morgan, han sido mostradas para los conjuntos imprecisos. La única ley de la teoria ordinaria de conjuntos que no es siempre válida es la ley del medio excluido, por ejemplo:

$$A \cap \neg A \neq \emptyset \qquad y \qquad A \cup \neg A \neq U$$

donde $\emptyset$ es el conjunto nulo, y $\mu_\emptyset(u) = 0$ para toda $u \in U$.

Dados dos subconjuntos imprecisos A y B en U, B es un subconjunto impreciso de A, denotado por $B \subseteq A$, si:

$$\mu_B(u) \leq \mu_A(u), \quad \text{para toda } u \in U.$$

Dos conjuntos imprecisos A y B son iguales si $A \supseteq B$ y $A \subseteq B$.

Para definir el producto cartesiano de conjuntos imprecisos, sea $U = U_1 \times U_2 \times \ldots \times U_n$ el producto cartesiano de n universos y $A_1, A_2, \ldots, A_n$ conjuntos imprecisos en $U_1, U_2, \ldots U_n$, respectivamente. El producto cartesiano $A_1 \times A_2 \times \ldots \times A_n$ es definido como un subconjunto impreciso de $U_1 \times U_2 \times \ldots \times U_n$, donde:

$$\mu_{A1x...xAn}(u_1 \ldots u_n) = \min(\mu_{A1}(u_1),\ldots,\mu_{An}(u_n))$$

donde $u_i \in U_i$, $i = 1,\ldots,n$. Finalmente, dado un conjunto impreciso, es muchas veces necesario construir un conjunto clásico con elementos que tengan un valor de pertenencia mayor que $\alpha \in [0,1]$. Así, dado un conjunto impreciso A en U, el valor límite α del conjunto A está dado por:

$$A_\alpha = \{ u / u \in U \text{ y } \mu_A(u) \geq \alpha \}$$

La definición de conjunto impreciso que dimos inicialmente tiene que extenderse para definir una segunda categoría de conjuntos imprecisos cuyos grados de pertenencia son por sí mismos imprecisos; por ejemplo, de la expresión $\mu_F : U \rightarrow [0,1]$ es una correspondencia desde U a los conjuntos de conjuntos imprecisos por encima de [0,1].

V.3.2 DISTRIBUCION DE PROBABILIDAD Y CONJUNTOS IMPRECISOS.

En vez de tratar a $\mu_F(u)$ como el grado de pertenencia de u en F, otra manera de interpretarlo es como una medida de la posibilidad de que una variable X tenga un valor u, donde X toma valores en U. Por ejemplo, consideremos el conjunto impreciso SALARIO-ALTO mostrado a continuación:

SALARIO-ALTO = {0.1/20,000, 0.3/30,000, 0.5/40,000, 0.7/50,000, 0.9/70,000, 0.95/80,000, 1.0/90,000}

Supongamos que se conoce que Juan tiene un 'salario alto'. Entonces, de acuerdo a la interpretación de posibilidades, concluimos que la posibilidad de que Juan tenga un salario = 30,000 es 0.3.

Zadeh sugiere que una proposición imprecisa ' X es F ', donde F es un subconjunto impreciso de U y X es una variable que tomará valores de U, induce una distribución de posibilidad $\prod x$ que es igual a F:

$$\Pi x = F$$

La ecuación de asignación de posibilidad es interpretada como:

$$Pos(X = u) = \mu_F(u) \quad \text{para toda } u \in U$$

Así, la distribución de probabilidad de X es un conjunto impreciso, el cual sirve para definir la posibilidad de que X tome cualquier valor u en U. Podemos definir una función $\Pi_x : U \rightarrow [0,1]$, que es igual a μ_F y asocia con cada $u \in U$ la posibilidad de que X tome el valor de u.

$$\Pi_x(u) = Pos(X = u) \quad \text{para } u \in U$$

La función Π_x es llamada función de distribución de probabilidad de X.

La distribución de probabilidad Π_x puede también ser usada para definir una medida imprecisa Π sobre U, donde para cualquier $A \subseteq U$,

$$\Pi(A) = Pos(X \in A)$$

V.3.3 RELACIONES IMPRECISAS.

Matemáticamente, una relación imprecisa r es un subconjunto impreciso del producto cartesiano de algunos universos. Así, dados n universos: $U_1, U_2, \ldots, U_n$, una relación imprecisa r es un subconjunto impreciso de $U_1 \times U_2 \times \ldots \times U_n$ y es caracterizado por la función de pertenencia:

$$\mu_r : U_1 \times U_2 \times \ldots \times U_n \rightarrow [0,1]$$

Mientras aplicamos esta definición a bases de datos relacionales, es necesario proveer la interpretación apropiada para los elementos de U_i, $i=1,\ldots,n$ y μ_r.

En un modelo de datos relacionales que soporta información imprecisa, podemos distinguir dos tipos de imprecisión: imprecisión en los valores de los datos, e imprecisión en la

asociación entre los valores de datos. Como un ejemplo de imprecisión en valores de datos, consideremos la base de datos de Empleados, donde se sabe que el salario de Juan está entre $60,000 y $80,000, o simplemente que su salario es alto, estamos manejando imprecisiones en los valores de los datos. Para ejemplificar el segundo tipo de imprecisión, consideremos una base de datos Preferencias, con atributos Estudiante y Curso, en la que se representa cuanto le gusta a un estudiante un curso en particular. Aquí, los valores de los datos pueden ser precisamente conocidos, pero el grado en el cual un estudiante, digamos Ana, le gusta DBMS es impreciso. No resulta difícil imaginar ejemplos donde ambas ambigüedades en cuanto a valores y asociaciones entre éstos estén presentes.

En años recientes, se han realizado algunos intentos para usar la teoría de conjuntos imprecisos y conceptos relacionados, para hacer una interpretación adecuada de los diferentes tipos de imprecisiones en las bases de datos relacionales. Buckles y Petry han sugerido que los valores de un atributo sean reemplazados por conjuntos de valores. Una medida similarmente imprecisa ha sido también usada para identificar tuplas similares. Ruspini ha usado una organización en rejillas para dominios, donde los valores del dominio corresponden a uno o más puntos del enrejado, determinado ésto por una distribución de probabilidad. Umano, Prade y Testemale han propuesto modelos basados explícitamente en distribuciones de probabilidad, donde los valores del dominio son tomados desde conjuntos de distribuciones de probabilidad, y las asociaciones entre las entidades son también medidas por distribuciones de probabilidad. Prade y Testemale han mostrado que asi un modelo de datos extendido puede alojar diferentes tipos de 'valores nulos' usados en la literatura para bases de datos relacionales clásicas. Baldwin ha usado una aproximación mixta, donde los valores del dominio son conjuntos imprecisos, y la asociación entre las entidades se representa como un valor de verdad en [0,1]. Zemankova-Leech y Kandel se han avocado al uso de cuantificadores lingüísticos.

En lo sucesivo, trataremos de adherirnos a la notación usada en la teoría clásica de bases de datos relacionales tanto como sea posible. Así un esquema de relación R es un conjunto finito de nombres de atributos {A1, A2, ..., An} y se denota como R(A1 A2 ... An), o simplemente R. Para cada atributo Ai, con 1≤ i≤ n, tenemos un conjunto dom(Ai), llamado dominio de Ai. Sin embargo, a diferencia de las relaciones clásicas, en el modelo relacional impreciso, el dominio de Ai puede ser un conjunto impreciso o aún más, un conjunto de conjuntos imprecisos. Por consiguiente, a cada atributo Ai asociamos un conjunto Ui, llamado el universo para los valores del dominio de Ai.

DEFINICION V.3.2 Una relación imprecisa r sobre un esquema de relación R(A1 A2 ... An) es un subconjunto impreciso de dom(A1) x dom(A2) x ... x dom(An).

Dependiendo de la complejidad del dom(Ai), i=1,...n, podemos clasificar las relaciones imprecisas en dos categorías. En las relaciones imprecisas de tipo 1, el dom(Ai) puede ser solo un conjunto impreciso (o un conjunto clásico). Las relaciones imprecisas del tipo 1 pueden ser consideradas como una extensión en primer nivel de las relaciones clásicas, donde seremos capaces de capturar las imprecisiones en la asociación entre las entidades. Las relaciones imprecisas del tipo 2 brindan una mayor generalización al permitir que el dom(Ai) sea un conjunto de conjuntos imprecisos (o distribuciones de probabilidad). Al ampliar el dom(Ai), las relaciones imprecisas del tipo 2 nos permiten representar un tipo más amplio de imprecisiones en los valores de los datos. Tales relaciones pueden ser consideradas como una generalización en segundo nivel, de las relaciones clásicas.

Finalmente, como las relaciones clásicas, una relación imprecisa r será representada como una tabla con una columna adicional para $\mu_r(t)$, la cual denota el valor de pertenencia de la tupla t en r. Mas aún, esta tabla contendrá solo aquellas tuplas para las cuales $\mu_r(t) > 0$; así, para cualquier

tupla no presente en la tabla podemos asumir que $\mu_r(t) = 0$. Como la ley del medio excluido no se aplica a conjuntos imprecisos, si $\mu_{\neg r}(t) > 0$, donde $\neg r$ es el complemento de r, no podemos concluir que $\mu_r(t) = 0$. Solo cuando $\mu_{\neg r}(t) = 1$, por ejemplo, t está definitivamente en r, y tenemos que $\mu_r(t) = 0$.

a) MODELO DE DATOS RELACIONAL IMPRECISO, TIPO 1

Como mencionamos anteriormente, en una relación imprecisa de tipo 1, el $dom(A_i)$ puede ser un subconjunto clásico o un subconjunto impreciso de U_i. Sea la función de pertenencia del $dom(A_i)$ denotada por μ_{A_i}, para $i = 1,\ldots,n$. Entonces de la definición de producto cartesiano de conjuntos incompletos, $dom(A_1) \times dom(a_2) \times \ldots \times dom(A_n)$ es un subconjunto impreciso de $U = U_1 \times U_2 \times \ldots \times U_n$. Por lo tanto, una relación imprecisa r de tipo 1 es también un subconjunto impreciso de U con función de pertenencia μ_r. También, para toda $(u_1\ u_2\ \ldots\ u_n) \in U$, μ_r debe satisfacer:

$$\mu_r(u_1\ u_2\ \ldots\ u_n) \leq \min(\mu_{A_1}(u_1), \mu_{A_2}(u_2), \ldots, \mu_{A_n}(u_n))$$

De acuerdo a la interpretación probabilística de conjuntos imprecisos, μ_r puede ser tratado como una función de distribución de probabilidad en U. Así, $\mu_r(u_1\ u_2\ \ldots\ u_n)$ determina la probabilidad de que una tupla $t \in U$ tenga $t[A_i]=u_i$, para $i=1,\ldots,n$. En otras palabras, $\mu_r(u_1\ u_2\ \ldots\ u_n)$ es una medida imprecisa de asociación entre un conjunto de valores del dominio $\{u_1, u_2, \ldots, u_n\}$.

EJEMPLO 1.

Consideremos un esquema de relación PREFERENCIAS(Estudiante, Curso), donde dom(Estudiante) y dom(Curso) son conjuntos ordinarios. En la relación imprecisa r mostrada en la tabla I, $\mu_r(t)$ puede ser interpretada como una medida de la posibilidad de que a un estudiante le guste un curso en particular. Así, la posibilidad de que a Juan le guste DBMS es 0.90. Entonces, μ_r es una medida imprecisa de la asociación entre Estudiante

y Curso. Podemos dar una interpretación alternativa de μ_r como un valor de verdad impreciso, perteneciente al intervalo [0,1]. De acuerdo a esta interpretación, para una tupla t, $\mu_r(t)$ es el valor de verdad de un predicado impreciso asociado con r, cuando las variables en el predicado son reemplazadas por $t[A_i]$, $i=1,\ldots,n$.

TABLA 1

Estudiante	Curso	μ
Juan	DBMS	0.90
Maria	DBMS	0.70
Juan	AI	0.80
Pedro	AI	0.95

EJEMPLO 2.-

Consideremos un esquema de relación R(N, T, E, S) de empleados altamente experimentados y con altos salarios, donde N = nombre del empleado, T = trabajo, E = experiencia, y S = salario. Aquí, dom(N) y dom(T) son conjuntos ordinarios, pero dom(E) y dom(S) son los conjuntos imprecisos ALTA-EXPERIENCIA y ALTO-SALARIO, en los universos apropiados. Supongamos que el universo U_e para el atributo Experiencia es el conjunto de enteros positivos en el rango de 0 a 30. Similarmente, U_s, el universo para Salario, es el conjunto de numeros enteros en el rango de 10,000 a 100,000.

La función de pertenencia μ_{AE} y μ_{AS} de los conjuntos imprecisos ALTA-EXPERIENCIA y ALTO-SALARIO se dan a continuación:

$$\mu_{AE}(e) = (1 + |e-10|/4)^{-1} \quad \text{para } e \leq 10$$
$$= 1 \quad \text{para } e > 10$$

$$\mu_{AS}(s) = (1 + |s-60,000|/20,000)^{-1} \quad \text{para } s \leq 60,000$$
$$= 1 \quad \text{para } s > 60,000$$

Note que la función de pertenencia asociada con el descriptor ALTO es dependiente del dominio. Una instancia típica r de R se muestra en la tabla II.

En este ejemplo, μ_r (t) puede ser interpretado como el valor de verdad de la proposición imprecisa 'Y tiene alta experiencia y alto salario' para la tupla t. Así, el valor de verdad de la proposición imprecisa 'Juan tiene alta experiencia y alto salario' es 0.67.

TABLA II

Nombre	Trabajo	Experiencia	Salario	μ
Juan	Ingeniero	8	60,000	0.67
Manuel	Gerente	9	70,000	0.80
Maria	Secretaria	8	40,000	0.50
Javier	Ingeniero	12	80,000	1.00
Roberto	Ingeniero	9	60,000	0.80

b) MODELO DE DATOS RELACIONAL IMPRECISO, TIPO 2.

A pesar de que las relaciones del tipo 1 nos permiten representar imprecisiones en las asociaciones entre los valores de datos, su papel para capturar incertidumbre en valores de datos es más limitado. Por ejemplo, en un modelo relacional de tipo 1 para Empleados (Nombre,Salario), no está permitido especificar que el salario de Juan está en el rango de $40,000 a $50,000, y que el salario de María es un conjunto impreciso 'bajo'. Para poder acomodar una clase más amplia de ambigüedades en los datos, consideraremos ahora una generalización del modelo

de datos relacional impreciso donde para cualquier atributo A_i, su $dom(A_i)$ puede ser un conjunto de conjuntos imprecisos en U_i.

Como consecuencia de esta generalización, una tupla $t=(a_1\ a_2 \ldots\ a_n)$ en $D = dom(A_1) \times dom(A_2) \times \ldots \times dom(A_n)$ se convierte en un subconjunto impreciso de $U = U_1 \times U_2 \times \ldots \times U_n$, con:

$$\mu_t\ (u_1\ u_2 \ldots\ u_n) = \min\ [\mu_{a_1}(u_1), \mu_{a_2}(u_2), \ldots, \mu_{a_n}(u_n)\ ]$$

donde $u_i \in U_i$, para $i=1,\ldots,n$.

Una relación imprecisa r del tipo 2 es un subconjunto impreciso de D. La función de pertenencia:

$$\mu_r : D \dashrightarrow [0,1]$$

debe satisfacer la siguiente condición:

$$\mu_r\ (t) \leq \max_{(u_1\ u_2 \ldots\ u_n) \in U} [\min \{\mu_{a_1}(u_1), \mu_{a_2}(u_2), \ldots, \mu_{a_n}(u_n)\}]$$

donde $t = (a_1\ a_2 \ldots\ a_n) \in D$.

Para una tupla $t = (a_1\ a_2 \ldots\ a_n) \in D$, la posibilidad de $t[A_i] = u_i$ es igual a $\mu_{a_i}(u_i)$.

Por ejemplo, supongamos que una instancia de la relación Empleados (Nombre, Salario) contiene una tupla (Juan, S), donde S = {0.3/10,000, 0.6/20,000, 0.8/30,000}. Aquí, S representa la distribución de probabilidad para el salario de Juan; por ejemplo, Pos(Salario de Juan = 20,000) = 0.6. Basado en la interpretación de probabilidad, para una tupla t de r, obtenemos:

$$Pos\ (\ t[A_1]=u_1,\ t[A_2]=u_2 \ldots,\ t[A_n]=u_n\ ) = \min\ \{\ \mu_r(t)\ ,\ \mu_t(u_1\ u_2 \ldots u_n)\ \}\ \}\qquad \ldots \text{ ec. (1)}$$

donde $u_i \in U_i$, $i=1,\ldots,n$ y μ_t quedó definido anteriormente.

EJEMPLO 3.-

Consideremos la relación Empleados (N, D, T, E, S, I), donde N=Nombre del empleado, D=Departamento, T=Trabajo, E=Experiencia, S=Salario e I=Impuesto sobre la renta. El dom(N), dom(D) y dom(T) son conjuntos ordinarios, pero dom(E), dom(S) y dom(I) son conjuntos de conjuntos imprecisos en los universos Ue, Us y Ui, respectivamente. Ue, Us y Ui son conjuntos de enteros positivos en el rango 0-30, 10,000-100,000 y 1-10,000, respectivamente.

Una instancia tipica r de Empleados se muestra en la tabla III, donde los descriptores de conjuntos imprecisos ALTO, BAJO y MODERADO han sido usados para representar valores imprecisos sobre sus dominios respectivos. Como todos los elementos de un conjunto clásico tienen un valor de pertenencia de 1.0, éstos son representados sin sus valores de pertenencia. Asi, usamos {10,14,19} en vez de {1.0/10, 1.0/14, 1.0/19}. También, conjuntos clásicos de enteros consecutivos como {10, 11, 12} se denotan como 10-12.

TABLA III

Nombre	Depto.	Trabajo	Experien-cia	Salario	Impues-to	μ
Javier	Mecánica	Ingeniero	10	50,000	5,000	0.70
Manuel	Eléctrica	Gerente	15-20	ALTO	ALTO	0.90
Pedro	Finanzas	Contador	PEQUEÑO	BAJO	BAJO	0.60
Juan	Ventas	Gerente	MODERADO	40,000-60,000	4,000-7,000	0.80

Las funciones de pertenencia de los descriptores de conjuntos imprecisos tales como ALTO, BAJO, etc., son dependientes del

dominio, y se dan a continuación:

Para e ∈ Ue :

$$\mu_{\text{moderado}(e)} = (1 + |e - 8|)^{-1} \quad \text{para } e > 1$$
$$= 0 \quad \text{en cualquier otro caso}$$

$$\mu_{\text{pequeño}(e)} = (1 + 12\,e)^{-1} \quad \text{para } e > 0$$
$$= 0 \quad \text{en cualquier otro caso}$$

Para y ∈ Us o y ∈ Ur: $\mu_{\text{alto}(y)} = (1+a\,|y-c|)^{-1}$ para $y \leq c$

$= 1$ para $y > c$

donde: $a = 1/20{,}000$ y $c=60{,}000$ para $y \in$ Us;

$a = 1/1{,}000$ y $c=5{,}000$ para $y \in$ Ur.

También: $\mu_{\text{bajo}(y)} = 1 - \mu_{\text{alto}(y)}$

Aplicando la ecuación (1) a la segunda tupla en r, podemos concluir que la posibilidad de que Manuel tenga una experiencia = 18, un salario = 50,000 y un impuesto sobre la renta = 5,000 es de 0.67. El valor de posibilidad así obtenido podría ser útil durante una evaluación de consultas para identificar las tuplas que tienen una posibilidad diferente de cero (o mayor que un umbral dado) de satisfacer el predicado de consulta.

V.3.4 OPERACIONES RELACIONALES IMPRECISAS.

El álgebra relacional introducida por Codd consiste en operaciones tradicionales de conjuntos tales como: unión, intersección, producto cruz, etc. Como una relación imprecisa es, por definición, un subconjunto impreciso del producto cartesiano de los dominios de sus atributos, las definiciones de unión, intersección y producto cruz discutidas en el punto V.3.1 pueden también ser aplicadas a relaciones imprecisas.

Sea la relación imprecisa r una instancia del esquema

relacional R (A1 A2 ... An). Consideremos un subconjunto Ri (Ai1...Aik) de R. Para una tupla t de r (por ejemplo, $\mu_r(t) > 0$), t [Ri] denota la restricción de t sobre los atributos de Ri. Así, para t = (a1 a2 ... an), t[Ri] = (ai1...aik).

De acuerdo con Zadeh, la proyección $r_i = P_{Ri}(r)$ es una relación imprecisa con k atributos en dom(Ai1) x ... x dom(Aik). También la función de pertenencia μ_{ri} está dada por:

$$\mu_{ri}(t) = \max_{tr} \{\mu_r(tr) / tr[Ri] = t\}$$

donde tr es una tupla de r y $t \in$ dom(Ai1) x ... x dom(Aik). Así, las tuplas de ri son las restricciones de las tuplas de r, como en el caso de las relaciones clásicas. El operador max de la ecuación anterior asegura que si más de una tupla en r, digamos $S_t \subseteq r$, tiene la misma restricción t sobre Ri, entonces la proyección ri contendrá solo una tupla y su valor de pertenencia es el máximo de los grados de las tuplas en S_t.

EJEMPLO 4.-

La proyección de la relación imprecisa r en la tabla II, sobre R js = {Trabajo, Salario} se muestra en la tabla IV.

TABLA IV

Trabajo	Salario	μ
Ingeniero	60,000	0.8
Ingeniero	80,000	1.0
Gerente	70,000	0.8
Secretaria	40,000	0.5

En la literatura de conjuntos imprecisos, la proyección ha sido también llamada restricción imprecisa marginal. Como una operación de conversión, Zadeh ha definido la 'extensión cilíndrica' de una relación imprecisa.

Sea la relación imprecisa r_i una instancia del esquema relacional R_i $(A_{i1} \ldots A_{ik})$. También consideremos un esquema relacional R $(A_1\ A_2 \ldots A_n)$ donde $R_i \subseteq R$. La extensión cilíndrica de r_i sobre R es denotada por $C_R(r_i)$. De acuerdo a Zadeh, la extensión cilíndrica $\hat{r}_i = C_R(r_i)$, es una relación imprecisa formada por n atributos, en $D = dom(A_1) \times dom(A_2) \times \ldots \times dom(A_n)$. La función de pertenencia $\mu_{\hat{r}i}$ de $\hat{r}i$ está dada por:

$$\mu_{\hat{r}i}(t) = \mu_{r_i}(t[R_i]) \quad ; \text{ para } t \in D$$

De las definiciones de proyección y extensión cilíndrica, se deduce que para cualquier instancia r de un esquema relacional R, donde $R_i \subseteq R$:

$$\mu_{\hat{r}i}(t) \geq \mu_r(t)$$

donde $\hat{r}_i = C_R(P_{R_i}(r))$ y $t \in D$. En otras palabras,

$$r \subseteq C_R(P_{R_i}(r))$$

En relación a los ejemplos 2 y 4, para la tupla $t = (\text{Juan Ingeniero } 8\ 60,000)$, $\mu_r(t) = 0.67$, mientras que en la extensión cilíndrica $\hat{r}_{JS}$ de P_{RJS}, $\mu_{\hat{r}JS}(t) = 0.80$. De hecho, podemos llegar a situaciones donde $\mu_r(t) = 0$; por ejemplo, la tupla no pertenece a r. Así, de acuerdo a la tabla 2, $\mu_r(\text{Juan Ingeniero } 10\ 80,000) = 0$, pero esta tupla definitivamente pertenece a $\hat{r}_{JS}$, donde $\mu_{\hat{r}JS}(\text{Juan Ingeniero } 10\ 80,000) = 1.0$.

Podemos definir que para cualquier instancia r_i de R_i, donde $R \supseteq R_i$:

$$r_i = P_{R_i}(C_R(r_i))$$

Pasaremos a definir la unión de relaciones imprecisas. Sea ρ = R_1, R_2,...,R_s un conjunto de esquemas de relaciones y R (A_1 A_2 ... A_n) = R_1 R_2 ... R_s. Consideremos un conjunto de relaciones imprecisas r_1, r_2, ..., r_s donde r_i es una instancia de R_i; i=1,...,s. La unión natural de estas relaciones imprecisas se escribe como: $r = r_1 \infty r_2 \infty \ldots \infty r_s$, y es una relación imprecisa del esquema de relación R.

La función de pertenencia de r está dada por:

$$\mu_r\ (a_1\ a_2 \ldots a_n) = \min\ [\mu_{\hat{}r_1}\ (a_1\ a_2\ \ldots\ a_n),\ \mu_{\hat{}r_2}\ (a_1\ a_2\ \ldots\ a_n),\ldots,\mu_r\ (a_1\ \ a_2 \ldots a_n)]$$

donde $a_i \in dom(A_i)$, y $\hat{}r_j$ es la extensión cilíndrica de r_j sobre R, para i=1,...,n y j=1,...,s.

EJEMPLO 5.-

Consideremos la unión de la instancia r de la relación PREFERENCIAS mostrada en la tabla I, con una instancia r_1 de la relación ENSEÑANZA (Maestro, Curso) mostrada en la tabla V. La función de pertenencia μ_{r1} es interpretada como una medida de la habilidad del maestro para impartir el curso.

TABLA V

Maestro	Curso	μ
Martinez	DBMS	0.80
Martinez	AI	0.60
González	DBMS	0.60
González	AI	0.90

La relación r obtenida realizando la unión natural de estas

dos relaciones se muestra en la tabla VI.

TABLA VI

Estudiante	Maestro	Curso	μ
Juan	Martínez	DBMS	0.80
Juan	González	DBMS	0.60
Juan	Martínez	AI	0.60
Juan	González	AI	0.80
María	Martínez	DBMS	0.70
María	González	DBMS	0.60
Pedro	Martínez	AI	0.60
Pedro	González	AI	0.90

Los valores μ_r (t) de las tuplas en r pueden ser interpretados como una medida de la asociación entre estudiantes y maestros, basada en la preferencia de los estudiantes hacia un curso determinado, y la habilidad del maestro para enseñar ese curso.

Cuando las relaciones imprecisas r_i, con i=1,...,s se obtienen haciendo las proyecciones de una relación imprecisa r sobre R_i, con i=1,...,s, tenemos que:

$$r \subseteq P_{R1}(r) \bowtie P_{R2}(r) \bowtie \ldots \bowtie P_{Rs}(r)$$

esto es:

$$r \subseteq \bowtie_{\mu r}(r)$$

donde $\bowtie_{\mu r}(r) = P_{R1}(r) \bowtie P_{R2}(r) \bowtie \ldots \bowtie P_{Rs}(r)$ es la correspondencia entre unión y proyección.

V.4 LIMITES DE INTEGRIDAD IMPRECISOS.

Los límites de integridad en sistemas de bases de datos relacionales pueden ser clasificados en dos grupos:

1) Dependencia del dominio, la cual restringe los valores del dominio admisibles para los atributos; por ejemplo, 'la edad de un empleado es menor a 65 años', o 'nadie tiene más de 10 pies de altura'.

2) Dependencia de datos, la cual requiere que si algunas tuplas en la base de datos cumplen ciertas igualdades, entonces ya sea que existan también otras tuplas en la base de datos, o que algunos valores de las tuplas dadas sean iguales.

Entre estos dos tipos de dependencias, la dependencia de datos ha recibido una atención más amplia por su impacto en el diseño de sistemas de bases de datos. Varios tipos de dependencias de datos, tales como dependencia funcional, dependencia de enlace, etc., se han identificado y el problema de implicación asociado ha sido examinado. Este problema consiste en decidir si un conjunto dado de dependencias lógicamente implica otra dependencia y tiene un efecto importante sobre la síntesis automatizada de esquemas de bases de datos.

Como hemos generalizado sistemas de bases de datos relacionales para manejar información incompleta, será necesario considerar los límites de integridad que involucran construcciones imprecisas. Así, en una relación JUGADORES (Nombre, Edad, Estatura, Deporte, Ingreso), un límite de integridad puede ser definido como ' La mayoría de los jugadores de basketball son altos' o ' muchos jugadores de tenis perciben altos ingresos '. Estos límites de integridad imponen restricciones a los valores admisibles de altura o ingreso de los jugadores de basketball o tenis, respectivamente. Similarmente, como un ejemplo de una dependencia de datos imprecisa, consideremos el esquema relacional EMPLEADOS (Nombre,

Departamento, Trabajo, Experiencia, Salario), donde un límite de integridad puede ser enunciado como 'en cualquier departamento, los empleados con trabajos y experiencia similares deben tener casi igual salario '.

Zadeh ha introducido el concepto de particularización para manejar límites de datos. La particularización de una relación imprecisa r, dentro de un esquema relacional R (A1 A2 ... An), es el efecto de especificar la distribución de posibilidad de uno o más atributos Y ⊆ R. La relación resultante r es por lo tanto una restricción de la relación original.

La particularización de una relación imprecisa r de R(A1 A2 ... An) debida a una proposición imprecisa Y es G, donde Y = Ai1 ... Aik, produce la relación r con función de pertenencia dada por:

$$\mu_r(a_1, a_2, \ldots, a_n) = \min[\mu_r(a_1, a_2, \ldots, a_n), \mu_{\hat{G}}(a_1, a_2, \ldots, a_n)]$$

donde ai ∈ dom(Ai) y Ĝ es la extensión cilíndrica de G. La proposición imprecisa Y es G puede ya sea representar un límite de integridad, como en el ejemplo que daremos a continuación, o puede ser el predicado de una operación seleccionada sobre la base de datos.

EJEMPLO 6.- En la tabla siguiente mostramos una instancia de estudiantes jóvenes e inteligentes, con el esquema relacional R(Nombre, Edad, Estatura, Curso, Calificación).

TABLA VII

Nombre	Edad	Estatura	Curso	Calificación	μ
Alfredo	25	175	DBMS	75	0.75
Alfredo	25	175	AI	90	0.80
Juan	23	170	DBMS	92	0.90
Juan	23	170	AI	85	0.85
Carlos	20	160	DBMS	70	0.70

Ahora supongamos que esta relación debe satisfacer la dependencia del dominio impreciso 'Estudiantes son altos',' donde el conjunto impreciso ALTO se define como sigue:

ALTO = {0.40/150, 0.50/160, 0.65/165, 0.80/170, 0.90/175, 1.00/180}

La particularización de la relación imprecisa en la tabla anterior debida a este proposición imprecisa produce una instancia de la relación imprecisa de estudiantes jóvenes, inteligentes y altos, que se muestra a continuación.

TABLA VIII

Nombre	Edad	Estatura	Curso	Calificación	μ
Alfredo	25	175	DBMS	75	0.75
Alfredo	25	175	AI	90	0.80
Juan	23	170	DBMS	92	0.80
Juan	23	170	AI	85	0.80
Carlos	20	160	DBMS	70	0.50

V.4.1 REGLAS DE TRANSLACION DEL CALCULO IMPRECISO.

Para evaluar la particularización de una relación imprecisa debida a una proposición imprecisa compuesta, es necesario examinar cómo la distribución de posibilidad de una proposición imprecisa compuesta puede ser obtenida de las distribuciones de probabilidad de sus proposiciones constitutivas. Para este propósito, se utilizan las siguientes reglas desarrolladas por Zadeh, llamadas reglas de translación.

a) REGLA DEL MODIFICADOR.

Consideremos la proposición modificada X es σ F, donde σ es un modificador, tal como 'NO', 'MUY', o 'MAS O MENOS'. Cada modificador se relaciona con una función $f\sigma$: [0,1] --> [0,1]. La distribución de posibilidad $\prod_{X\,M}$ de la proposición modificada X es σ F está dada por:

$$\prod_{X\,M} = F_M$$

Aquí F_M es un subconjunto impreciso de U con la función de pertenencia:

$$\mu_{F_M}(u) = f_\sigma(\mu_F(u)) \qquad \text{para } u \in U$$

Así, el efecto del modificador es generar una nueva distribución de posibilidad sobre U, que está únicamente determinada por la función de modificación f_σ, asociada con el modificador, y la función de distribución de posibilidad μ_F de la proposición constitutiva.

Las siguientes funciones de modificación se recomiendan en la literatura de conjuntos imprecisos para algunos modificadores usados comunmente.

σ = NO,	$f_\sigma(x) = 1 - x$
σ = MUY,	$f_\sigma(x) = x^2$
σ = MAS O MENOS,	$f_\sigma(x) = \sqrt{x}$

b) REGLAS DE COMPOSICION.

Estas reglas pueden ser usadas para encontrar la distribución de probabilidad asociada a una proposición compuesta del tipo **X es F o Y es G**, donde el operador de composición o puede corresponder a 'AND', 'OR', etc. La distribución de probabilidad $\prod (X \circ Y)$ asociada con la proposición compuesta está dada por:

$$\prod (X \circ Y) = F \circ G$$

donde $F \circ G$ es un subconjunto impreciso de $U \times V$. Dependiendo del operador de composición, el subconjunto impreciso $F \circ G$ estará definido como sigue:

$$\circ = \text{AND}, \qquad F \circ G = {}^{\wedge}F \cap {}^{\wedge}G$$
$$\circ = \text{OR}, \qquad F \circ G = {}^{\wedge}F \cup {}^{\wedge}G$$

donde ${}^{\wedge}F$ y ${}^{\wedge}G$ son extensiones cilíndricas de F y G en $U \times V$, respectivamente, y la unión e intersección de conjuntos imprecisos ya fue definida en puntos anteriores.

DEFINICION V.4.1 Sean F y G subconjuntos imprecisos de U y V, respectivamente. La distribución de posibilidad $\prod (X \text{--}> Y)$ asociada con la proposición imprecisa condicional: **SI X ES F ENTONCES Y ES G**, está dada por:

$$\prod (X \text{--}> Y) = R_S$$

donde R_S es un subconjunto impreciso de $U \times V$ con la función de pertenencia:

$$\mu_{R_S} (u , v) = 1 \qquad \text{si} \quad \mu_F (u) \leq \mu_G (v)$$
$$= 0 \qquad \text{en otro caso}$$

Note que μ_{R_S} define una partición de $U \times V$.

Además de las reglas de translación discutidas anteriormente, Zadeh ha propuesto también reglas de translación para proposiciones imprecisas cuantificadas, tales como Q X ES F, donde el cuantificador Q puede representar 'LA MAYORIA', 'MUCHOS', 'POCOS', etc. Similarmente, las reglas de translación para determinar la distribución de probabilidad de la proposición imprecisa calificada X ES F ES τ, donde τ involucra una calificación de verdad, tal como 'CIERTO' o 'MUY CIERTO', ha sido también sugerida.

En una base de datos relacional, si los límites de integridad involucran proposiciones imprecisas compuestas, las reglas de translación discutidas arriba pueden ser aplicadas para determinar la distribución de probabilidad inducida por tales límites de integridad.

V.5 DEPENDENCIA FUNCIONAL IMPRECISA Y REGLAS DE INFERENCIA.

Nuestro próximo objetivo es extender la teoría de diseño de sistemas de bases de datos relacionales al dominio impreciso, para lo cual es necesario estudiar las dependencias de datos imprecisos y sus problemas de implicación asociados. Nos concentraremos en dependencias funcionales debido a su importancia entre los diferentes tipos de dependencias de datos, y a su facilidad para identificarlas desde las especificaciones de diseño de la base de datos.

Como ya mencionamos, una instancia r de un esquema relacional R(A1 A2 ... An) satisface una dependencia funcional f: X --> Y si, para cada par de tuplas t_1 y t_2 de r, tales que $t_1[X]=t_2[X]$, tenemos que $t_1[Y] = t_2[Y]$. En el dominio impreciso, igualdad de valores del dominio definen a una proposición imprecisa, y puede ser especificado como 'APROXIMADAMENTE IGUAL', 'MAS O MENOS IGUAL', etc. Por lo tanto, una dependencia de datos imprecisos en la relación EMPLEADOS(Nombre, Trabajo, Experiencia, Salario) puede ser enunciada como 'TRABAJO Y EXPERIENCIA MAS O MENOS IGUAL DETERMINAN SALARIO'.

V.5.1 LA IGUALDAD COMO UNA RELACION IMPRECISA.

La relación imprecisa IGUAL(EQ) que definimos abajo puede ser usada como una medida imprecisa para comparar elementos de un dominio dado.

DEFINICION V.5.1 Una relación imprecisa EQUAL(EQ) sobre un universo U se define como un subconjunto impreciso de U x U, donde μ_{EQ} satisface las siguientes condiciones.

Para toda a, b $\in$ U.

μ_{EQ} (a,a) = 1 (reflexividad)

μ_{EQ} (a,b) = μ_{EQ} (b,a) (simetria)

En términos de la teoria de probabilidad, μ_{EQ} (a,b) puede ser interpretado como la posibilidad de tratar a y b como 'IGUALES'. La función de pertenencia μ_{EQ} podria ser seleccionada apropiadamente durante la creación de la base de datos para capturar el significado de IGUAL/APROXIMADAMENTE IGUAL de los valores del dominio, tal como son percibidos por el diseñador de la base de datos.

Cabe hacer notar que a diferencia de la igualdad clásica, si manejamos conjuntos imprecisos la igualdad no es transitiva. Esto es con el objeto de manejar límites de integridad provenientes de valores del dominio que son aproximadamente iguales. De hecho, con medidas próximas/más distantes usadas para comparar valores del dominio, no está presente la transitividad.

Podemos extender la definición de IGUAL sobre dominios compuestos, como sigue. Sea D = dom(A1) x dom(A2) x...x dom(An), y sean t1 y t2 dos tuplas en D. Supongamos que en cada dom(Ai) ha sido definida una función de pertenencia para la relación de IGUALDAD, denotada como μ_{EQ} . Entonces la relación imprecisa

IGUAL, extendida sobre D, define un subconjunto impreciso de D x D, con la función de pertenencia siguiente:

$$\mu_{EQ}(t_1,t_2) = \min\{\mu_{EQ\ 1}(t_1[A_1],t_2[A_1]), \mu_{EQ\ 2}(t_1[A_2],t_2[A_2]),\ldots,\mu_{EQ\ n}(t_1[A_n],t_2[A_n])\}$$

ec. (A)

En relaciones imprecisas del tipo 1, dom(Ai) puede ser un subconjunto impreciso del universo Ui. Podemos definir la semejanza de los elementos del dom(Ai) como:

$$\mu_{EQ\ i}(a,b) = \min\{\mu_\alpha(a,b), \mu_\theta(\mu_{Ai}(a),\mu_{Ai}(b))\}$$

donde a, b $\in$ Ui, $\mu\alpha$ y $\mu\theta$ son las funciones de pertenencia de las relaciones de semejanza sobre Ui y [0,1] respectivamente.

Similarmente, en relaciones del tipo 2, podemos definir la igualdad sobre dom(Ai) por:

$$\mu_{EQ\ i}(a_{i1},a_{i2}) = \min_{ui \in Ui} \psi(\mu_{ai1}(ui),\mu_{ai2}(ui))$$

donde a_{i1}, a_{i2} $\in$ dom(Ai) son subconjuntos imprecisos de Ui y ψ es una relación de semejanza sobre [0,1].

EJEMPLO 7. Consideremos el esquema relacional EMPLEADO(N,D,T,E,S,I) y su instancia r discutida en el ejemplo 3. En este caso, algunos de los valores del dominio son precisos, mientras que dom(E), dom(S), dom(I) son conjuntos de conjuntos imprecisos. Podemos seleccionar las siguientes funciones de pertenencia como medidas de igualdad sobre diferentes dominios.

(1) $\mu_{EQ}(a,b) = 0$ para $a \neq b$, $a,b \in$ dom(A), $A \in \{N,D,T\}$.

En otras palabras, los nombres de los empleados, departamentos o trabajos deben coincidir exactamente para cumplir con el criterio de IGUALDAD.

(2) Para a,b ∈ dom(A), A ∈ {E,S,I}, surgen las siguientes situaciones.

i) Los valores del dominio a, b son elementos únicos del conjunto impreciso, con valores de pertenencia binarios. En este caso,

$$\mu_{EQ}(a,b) = 1 / (1 + \beta \mid a - b \mid),$$

donde β = 1 para a, b ∈ dom(E),
= 1/2000 para a,b ∈ dom(S),
= 1/500 para a,b ∈ dom(I).

ii) Si a es preciso y b es un subconjunto impreciso, entonces $\mu_{EQ}(a,b) = \mu_b(a)$. Similarmente, si b es preciso y a es un subconjunto impreciso, entonces $\mu_{EQ}(a,b) = \mu_a(b)$. Nótese que un subconjunto ordinario tal como 15-20 puede ser considerado como un subconjunto impreciso con valores de pertenencia binarios.

iii) Si tanto a como b son subconjuntos imprecisos, entonces:

$$\mu_{EQ}(a,b) = \max \{ c / card(a), c / card(b) \},$$

donde *card* es la cardinalidad del conjunto impreciso, y c = card(a ∩ b).

Cabe hacer notar que sobre el dom(A), A ∈ {E,X,I}, la relación de semejanza imprecisa IGUAL puede ser usada para proveer información acerca de la igualdad aproximada de los valores del dominio. En el ejemplo 2 vimos como estas relaciones de semejanza imprecisas pueden ser usadas para representar una dependencia de datos imprecisa 'PARA CUALQUIER TRABAJO, EXPERIENCIA DETERMINA SALARIO'.

Una vez que el significado de igualdad en los valores del

dominio es determinado, las reglas modificadoras del cálculo impreciso pueden ser aplicadas para obtener la interpretación para relaciones imprecisas 'MAS O MENOS IGUAL' o 'MUY SIMILARES'. Por lo tanto, la función de pertenencia de la relación imprecisa 'MAS O MENOS IGUAL' podría estar dada por:

$$\mu_{\text{MASOMENOSEQ}}(a,b) = \sqrt{\mu_{\text{EQ}}(a,b)}$$

En general, la función de pertenencia asociada con la relación modificada ρ IGUAL en dom(A) estará dada por:

$$\mu_{\sigma\text{EQ}}(a,b) = f_{\sigma}\ (\mu_{\text{EQ}}(a,b))$$

donde f_{σ} es la función modificadora asociada con σ y $a,b \in \text{dom}(A)$.

Esta función de pertenencia modificada puede ahora ser aplicada para determinar la posibilidad de que, por ejemplo, dos salarios sean mas o menos iguales.

Finalmente, puede mencionarse que a pesar de que la idea de tratar la igualdad como una relación imprecisa es motivada por nuestro intento de generalizar los límites de integridad, una aproximación similar puede seguirse en el procesamiento de consultas. Por ejemplo, una consulta sobre una base de datos puede estar especificada como 'DAME EL NOMBRE DE LOS EMPLEADOS QUE TIENEN MAS O MENOS IGUAL SALARIO QUE SU GERENTE'. En este caso, 'IGUAL' podría ser tratado como una relación imprecisa de semejanza, y 'MAS O MENOS' como un modificador de 'IGUAL'. También, la relación imprecisa 'IGUAL' durante el procesamiento de una consulta puede ser descrita por una función de pertenencia diferente que la que uno utiliza para comparar valores del dominio en dependencias de datos imprecisas. La evaluación de esta consulta recuperará los nombres de empleados que tienen una posibilidad diferente de cero (o por encima de cierto valor de umbral dado) de tener un salario 'MAS O MENOS IGUAL' al salario de su gerente. El valor de la posibilidad de la tupla de ser seleccionada estará determinado por la función de pertenencia de

la relación imprecisa 'MAS O MENOS IGUAL'.

V.5.2 DEPENDENCIA FUNCIONAL IMPRECISA.

Sean X = A_{i1} A_{i2} ... A_{ik}, y Y = A_{j1} A_{j2} ... A_{jp}, subconjuntos de un esquema relacional R (A_1 A_2 ... A_n). Basados en la interpretación de IGUALDAD, una proposición imprecisa 'X ES IGUAL' define a un subconjunto impreciso de dom(A_{i1}) x dom(A_{i2}) x .... x dom(A_{ik}) con la función de pertenencia determinada por la expresión (A).

Una generalización de una fd: X-->Y en R, llamada ffd: X ==> Y, puede ser definida como una particularización de una relación imprecisa sobre R, debida a una proposición condicional imprecisa 'SI X ES IGUAL ENTONCES Y ES IGUAL'. La particularización impuesta por una ffd está, por lo tanto, ligada con la regla de translación usada para proposiciones imprecisas condicionales.

Con una proposición imprecisa 'SI X ES IGUAL ENTONCES Y ES IGUAL', la distribución de probabilidad produce una partición del dom(A_1) x dom(A_2) x ... x dom(A_n). Combinando esta observación con el concepto de particularización, llegamos a la siguiente definición de **dependencia funcional imprecisa**.

DEFINICION V.5.2 Una **dependencia funcional imprecisa** (ffd) X==>Y, con X,Y $\subseteq$ R, existe en una relación imprecisa r sobre R, si para cualquier par de tuplas t_1, t_2 de r (para i=1,2), tenemos:

$$\mu_{EQ}(t_1[X], t_2[X]) \leq \mu_{EQ}(t_1[Y], t_2[Y])$$

NOTA.- Una fd en una base de datos relacional clásica puede ser vista como un caso especial de una ffd. Para ésto, consideremos ffd: X ==> Y y supongamos que la relación de semejanza IGUAL sobre dom(A), A $\in$ X Y, satisface la propiedad adicional de que

$\mu_{EQ}(a,b) = 0$ para $a \neq b$, $a,b \in dom(A)$. Para una base de datos relacional clásica r, $\mu_r(t) = 1$ para $t \in r$. La definición V.5.2 implica que no pueden existir dos tuplas de r que coincidan en los valores correspondientes al atributo X, y que difieran en los valores correspondientes al atributo Y.

EJEMPLO 8. Consideremos el esquema relacional EMPLEADOS(Nombre, Departamento, Trabajo, Experiencia, Salario, Impuesto) manejado en el ejemplo 3 , así como las relaciones de semejanza planteadas en el mismo.

Puede verificarse que con las definiciones hechas para μ_{EQ} , la relación imprecisa r en la Tabla III satisface las siguientes dependencias funcionales imprecisas.

N D ==> T : Nombre y departamento determinan trabajo.
T E ==> S : Para cualquier trabajo, empleados con 'igual' experiencia deberían tener 'igual' salario.
S ==> I : Para salarios 'iguales', impuestos 'iguales'.

Aquí, la dependencia funcional imprecisa: N D ==> T es, de hecho, una dependencia funcional clásica, debido a la naturaleza especial de la relación de IGUALDAD sobre el dominio de estos atributos.

Cabe hacer notar que la dependencia funcional imprecisa ffd: T E ==> S no permite que la tupla t_2 = (Jaime Eléctrica Ingeniero 11 55,000 5700) sea insertada en la base de datos, debido a que ésta ya contiene a la tupla t_1 = (Javier Mecánica Ingeniero 10 50,000 5,000), y :

$$\mu_{EQ}(t_1[TE], t_2[TE]) > \mu_{EQ}(t_1[S], t_2[S])$$

Sin embargo, la inserción de esta tupla no violaría una dependencia funcional clásica fd: T E --> S. Así, el límite de integridad 'para cualquier trabajo, la experiencia determina el salario' no está adecuadamente representado por una dependencia funcional clásica, cuando la interpretación de este límite de

integridad requiere que 'para cualquier trabajo, los empleados que tengan aproximadamente igual experiencia deben tener aproximadamente igual salario.' Así, al seleccionar adecuadamente μ_{EQ}, la dependencia funcional imprecisa ffd : T E ==> S provee un modelo más aceptable para tales límites de integridad.

Podemos manejar límites de integridad que involucren modificadores imprecisos. Así, tenemos que dado un modificador impreciso σ con $f_\sigma(1) = 1$, la relación modificada σ IGUAL es también reflexiva y simétrica. Con tales modificadores, un límite de integridad definido por una proposición imprecisa ' Si X es σ_1 igual entonces Y es σ_2 igual ' puede ser expresado como una ffd, donde las funciones de pertenencia modificadas de IGUAL se definen como:

$$f_{\sigma_1}(\mu_{EQ}(t_1[X], t_2[X])) \leq f_{\sigma_2}(\mu_{EQ}(t_1[Y], t_2[Y])) \qquad \text{ec. (B)}$$

Así, con el límite de integridad 'X más o menos determina Y', asociamos una relación de igualdad modificada MLEQ sobre los dominios de los atributos en Y, donde la función de pertenencia de MLEQ es calculada usando la siguiente expresión:

$$\mu_{MLEQ}(a,b) = \forall \mu_{EQ}(a,b)$$

Podemos decir que la dependencia funcional imprecisa ffd: ' X MAS O MENOS determina Y ', se presenta en una relación imprecisa r, si para dos tuplas cualesquiera t_1 y t_2 se cumple que:

$$\mu_{EQ}(t_1[X], t_2[X]) \leq \forall \mu_{EQ}(t_1[Y], t_2[Y])$$

EJEMPLO 9.

Supongamos que una relación ABASTECIMIENTO (Número de pedido, Pedido, Fecha de Orden, Fecha de Entrega) satisface un límite de integridad definido como: 'Para cualquier pedido, la Fecha de Orden más o menos determina la Fecha de Entrega'. Para representar este límite de integridad consideremos las siguientes

relaciones de semejanza sobre los dominios de Pedido, Fecha de Orden y Fecha de Entrega.

i) $\mu_{EQ}(a,b) = 0$ para $a \neq b$, donde $a,b \in$ dom(Pedido)

ii) $\mu_{EQ}(a,b) = 1 / (1 + |a - b|)$, para $a,b \in$ dom(Fecha de Orden), o dom(Fecha de Entrega),

donde $|a - b|$ representa la diferencia en el número de días entre dos fechas a y b.

Entonces, la sentencia 'MAS O MENOS IGUAL Fecha de Entrega' define una relación de semejanza imprecisa sobre el dominio Fecha de Entrega, con función de pertenencia $1 / (1 + |a-b|)$. La ffd que representa el límite de integridad enunciado previamente requiere que en cualquier instancia de la relación ABASTECIMIENTO, dos tuplas cualesquiera t_1 y t_2 con t_1[Pedido] = t_2[Pedido] satisfagan la siguiente condición:

$$1 / (1 + |t_1[\text{Fecha de Orden}] - t_2[\text{Fecha de Orden}]|) \leq \forall (1 / (1 + |t_1[\text{Fecha de Entrega}] - t_2[\text{Fecha de Entrega}]|)$$

ec. (C)

La relación clásica mostrada en la Tabla IX es una instancia típica de ABASTECIMIENTO, que satisface la ecuación (C).

TABLA IX

Número de Pedido	Pedido	Fecha de Orden	Fecha de Entrega
100	nuez	15-10-86	18-10-86
102	pasas	12-10-86	18-10-86
100	nuez	20-10-86	24-10-86
104	clavo	15-10-86	18-10-86

Esta dependencia funcional imprecisa **ffd** no permite que una tupla (102 pasas 14-10-86 29-10-86) sea insertada en la base de datos, puesto que ya está presente la tupla (102 pasas 12-10-86 18-10-86), y se violaría el límite de integridad representado con la ffd definida en la ecuación (C).

La inserción de esta tupla, sin embargo, no violaría una dependencia funcional clásica **fd** : Pedido Fecha de Orden --> Fecha de Entrega, lo que implica que la dependencia funcional clásica ha fallado al interpretar el significado del límite de integridad dado.

V.5.3 DEDUCCION DE AXIOMAS PARA DEPENDENCIA FUNCIONAL IMPRECISA.

Dado un conjunto de dependencias de datos, presentes en una base de datos, la mayoría de las veces es posible derivar otras dependencias de datos para la misma base de datos. Para derivar estas nuevas dependencias se utiliza un conjunto de axiomas , que presentaremos a continuación.

Consideremos un esquema relacional R (A1, A2, ..., An) y un conjunto de dependencias funcionales imprecisas (ffd) F. Una instancia r de R es llamada una instancia legal si r satisface todas las dependencias funcionales imprecisas en F. La validez de una ffd en una relación imprecisa r, se asume que tendrá un valor de verdad binario. En los axiomas siguientes, X, Y y Z son subconjuntos del esquema relacional R.

AXIOMAS.

1. REFLEXIVIDAD: Si $Y \subseteq X$, entonces $X \Longrightarrow Y$.
   Por consiguiente, tenemos que:
   $\mu_{EQ}(t_1[X], t_2[X]) \leq \mu_{EQ}(t_1[Y], t_2[Y])$.

2. AGREGACION: Si se tiene que $X \Longrightarrow Y$, entonces también se tendrá que $XZ \Longrightarrow YZ$, donde:

$$\min \{\mu_{EQ}(t_1[X], t_2[X]), \mu_{EQ}(t_1[Z], t_2[Z])\} \leq \min \{\mu_{EQ}(t_1[Y], t_2[Y]), \mu_{EQ}(t_1[Z], t_2[Z])\}$$

3. TRANSITIVIDAD: Si X ==> Y y Y ==> Z , entonces X ==> Z. Esto es, como:

$$\mu_{EQ}(t_1[X], t_2[X]) \leq \mu_{EQ}(t_1[Y], t_2[Y]), \quad y$$
$$\mu_{EQ}(t_1[Y], t_2[Y]) \leq \mu_{EQ}(t_1[Z], t_2[Z])$$

entonces se infiere que:

$$\mu_{EQ}(t_1[X], t_2[X]) \leq \mu_{EQ}(t_1[Z], t_2[Z])$$

De estos axiomas podemos inferir los siguientes:

4. UNION: Si X ==> Y y X ==> Z, entonces X ==> YZ.

5. DESCOMPOSICION: Si X ==> YZ, entonces X ==> Y y X ==> Z.

6. PSEUDOTRANSITIVIDAD: Si X ==> Y y YW ==> Z, entonces XW ==> Z.

DEFINICION V.5.3

Supongamos que F es un conjunto de dependencias funcionales imprecisas dentro de un esquema relacional R, y sea W $\subseteq$ R. Entonces W+, la cerradura de W con respecto a F, se define como el conjunto de atributos A $\in$ R, tales que W==>A puede ser obtenido desde F usando los axiomas vistos anteriormente.

LEMA 5.1 W ==> V se desprende de los axiomas de inferencia de dependencias funcionales imprecisas, si y solo si V $\subseteq$ W+.

LEMA 5.2 La unión natural de relaciones imprecisas conserva las dependencias funcionales imprecisas.

En vista de que el objetivo del presente capitulo es mostrar un panorama general y los conceptos básicos para el manejo de bases

de datos imprecisas, no incluímos las demostraciones de los lemas y teoremas mencionados. Además, como en aplicación práctica no se requerirán estos teoremas en forma estricta, preferimos omitir los detalles demostrativos. Sin embargo, el lector que esté interesado en profundizar en el tema, puede referirse a la bibliografía que incluímos al final del capítulo.

A continuación examinaremos lo completo de los axiomas de ffd. Un conjunto de axiomas de inferencia es considerado completo para una familia de restricciones o limitantes, si para cada conjunto F de la familia, las restricciones implicadas por F son exactamente aquellas que pueden ser derivadas de éste, usando los axiomas de inferencia. El siguiente ejemplo muestra que los axiomas de inferencia vistos anteriormente no son siempre completos.

EJEMPLO 10. Seleccionaremos las relaciones de semejanza sobre dom(Ai), con i=1,2,3, planteadas en el ejemplo 7 , y agregaremos la siguiente condición:

Para toda a_{ri}, $a_{sj} \in$ dom(A$_i$) y para toda a_{rk}, $a_{rp} \in$ dom(A$_r$), r = 2,3, si $a_{rk} \neq a_{rp}$, entonces $\mu_{EQ}(a_{ri}, a_{sj}) > \mu_{EQ}(a_{rk}, a_{rp})$

Con esta definición de EQUAL, consideremos la dependencia funcional imprecisa ffd A$_1$ A$_2$ ==> A$_3$ sobre R(A$_1$ A$_2$ A$_3$). Sea r una instancia legal de R. Dado que r satisface A$_1$ A$_2$ ==> A$_3$, para dos tuplas cualesquiera t_1 y t_2:

$$\mu_{EQ}(t_1[A_1\ A_2],\ t_2[A_1\ A_2]) \leq \mu_{EQ}(t_1[A_3],\ t_2[A_3])$$

Mostraremos ahora que r también satisface A$_2$ ==> A$_3$. Para probar ésto, notamos que cuando la proposición $p \equiv$ ((t$_1$[A$_2$] $\neq$ t$_2$[A$_2$]) AND (t$_1$[A$_2$] = t$_2$[A$_2$])) es verdadera, se cumple que:

$$\mu_{EQ}(t_1[A_1\ A_2], t_2[A_1\ A_2]) = \mu_{EQ}(t_1[A_1],\ t_2[A_1])$$

Así también tenemos que:

$t_1[A_3] = t_2[A_3]$

Por consiguiente, y por reflexividad de las relaciones de semejanza, r satisface que: $A_2 \Longrightarrow A_3$.

Similarmente, cuando φ es falsa (ya sea que $t_1[A_3] = t_2[A_3]$, o que $t_1[A_2] \neq t_2[A_2]$), tendremos que $\mu_{EQ}(t_1[A_1\ A_2], t_2[A_1\ A_2]) = \mu_{EQ}(t_1[A_2], t_2[A_2])$. Entonces, r satisface que $A_2 \Longrightarrow A_3$.

Por lo tanto, para la definición de EQUAL que estamos manejando, la ffd $A_1\ A_2 \Longrightarrow A_3$ implica $A_2 \Longrightarrow A_3$, a pesar de que tal implicación no puede desprenderse del uso de los axiomas de inferencia. Si restringimos EQUAL aún más por la condición que $\mu_{EQ}(a_{1i}, a_{1j}) = 1.0$ para toda a_{1i}, $a_{1j} \in dom(A_3)$, r satisfacerá las dependencias funcionales imprecisas $A_2 \Longrightarrow A_1$ y $A_3 \Longrightarrow A_1$. Nuevamente, ninguna de estas dos ffds puede inferirse desde $A_1\ A_2 \Longrightarrow A_3$ usando los axiomas de inferencia.

El ejemplo 10 indica que, dependiendo del tipo de relaciones de semejanza usadas para definir las ffds, es posible implicar nuevas ffds que no pueden ser inferidas usando los axiomas de inferencia. Para inferir tales ffds, tenemos que considerar axiomas de inferencia adicionales que dependen de las relaciones de semejanza usadas para la comparación de los valores del dominio. Como el número de relaciones de semejanza que podemos definir sobre cualquier dominio es infinito, un conjunto completo de axiomas de inferencia puede ser obtenido para las ffds apropiadas, donde las restricciones adicionales son impuestas en la definición de EQUAL. Por consiguiente, será de utilidad encontrar una clase de ffds para las cuales los seis axiomas de inferencia planteados anteriormente constituyan un conjunto completo. El siguiente teorema establece la perfección de estos axiomas cuando cada dominio tiene al menos dos elementos que no son semejantes entre sí.

TEOREMA 5.1 Los axiomas de inferencia 1, 2 y 3 forman un conjunto completo de axiomas de inferencia para dependencias funcionales imprecisas de un esquema relacional $R(A_1\ A_2\ \ldots\ A_n)$ cuando se cumple la siguiente condición:

Para cada Ai ∈ R, existe al menos un par de elementos ai, bi ∈ dom(Ai) tales que $\mu_{EQ}(ai, bi) = 0$.

En la mayoría de las aplicaciones del mundo real no es difícil seleccionar las relaciones de semejanza apropiadas que satisfagan el teorema 5.3.1; las ffds pertenecientes a esta clase pueden no obstante representar límites de integridad imprecisos que involucren la igualdad aproximada de los valores del dominio. Por ejemplo, podemos definir las relaciones de semejanza sobre los dominios dom(Fecha-de-orden), y dom(Fecha-de-entrega), tales que:

$$\mu_{EQ}(a,b) = 1/(1 + |a - b|) \quad \text{para } |a - b| \leq 30$$
$$= 0 \quad \text{en otro caso.}$$

La dependencia funcional imprecisa resultante captura la semántica del límite de integridad 'para cualquier pedido, la fecha de orden MAS O MENOS determina la fecha de entrega', para dos órdenes cualesquiera efectuadas en el lapso de 30 días.

V.6 UNION CON MENOS PERDIDA EN RELACIONES IMPRECISAS.

Para responder las consultas de los usuarios en una base de datos relacional imprecisa, muchas veces es necesario unir dos o más relaciones imprecisas. Sin embargo, la unión natural puede no recuperar la relación imprecisa original. El problema de minimizar la pérdida de unión de esquemas relacionales es de central importancia en la teoría de diseño de bases de datos relacionales. De hecho, el concepto de la descomposición adecuada de bases de datos relacionales requiere que el esquema relacional sintetizado tenga la propiedad de menos pérdida de unión. A continuación examinaremos la minimización de pérdidas de unión de relaciones imprecisas, en la presencia de dependencias funcionales imprecisas.

DEFINICION V.6.1 Sean R un esquema y $\rho = \{ R_1, R_2, \ldots, R_s \}$ una descomposición de R, con $R = R_1 R_2 \ldots R_s$. Esta

descomposición es una unión con menos pérdida con respecto a un conjunto de ffds F, si para toda relación imprecisa r de R que satisfaga estas ffds, se cumple la siguiente condición:

$$r = \bowtie \rho (r) \qquad \text{ec. (D)}$$

Una descomposición ρ de R es una unión con menos pérdida para un conjunto dado de límites de integridad, si y solo si, para cualquier tupla $(a_1\ a_2\ \ldots\ a_n) \in dom(A_1) \times dom(A_2) \times \ldots \times dom(A_n)$, existe al menos una $\hat{r}_i = C_R(P_{R_i}(r))$, $i \in \{1,2,\ldots,s\}$, tal que:

$$\mu_{\hat{r}_i}(a_1\ a_2\ \ldots\ a_n) = \mu_r(a_1\ a_2\ \ldots\ a_n) \qquad \text{ec. (E)}$$

Note que la validez de la condición es requerida para cualquier tupla $t \in D = dom(A_1) \times dom(A_2) \times \ldots \times dom(A_n)$, y no solo para aquellas tuplas que estén presentes en r, con $\mu_r(t)>0$. El siguiente ejemplo, con una relación clásica, ilustra que una tupla t puede no estar presente en una instancia r, y aún así conducir a una pérdida de unión.

EJEMPLO 11. Consideremos un esquema relacional R(ABC) con fds A --> B y C --> B, y una instancia r de R conteniendo dos tuplas $t_1 = (a_1\ b\ c_1)$ y $t_2 = (a_2\ b\ c_2)$. Sea $\rho = \{AB, BC\}$ una descomposición de R. Es bien conocido que esta descomposición no tiene menos pérdida de unión. Esta conclusión puede también ser verificada observando que para una tupla $t = (a_1\ b\ c_2)$, $\mu_r(t) = 0$; no obstante $\mu_{\hat{r}_1}(t) = 1$ y $\mu_{\hat{r}_2}(t) = 1$, donde $\hat{r}_i$, con $i = 1,2$, son las extensiones cilíndricas de las proyecciones de r sobre AB y BC, respectivamente.

En el siguiente ejemplo, mostramos una descomposición con menos pérdida de una relación imprecisa basada en una dependencia funcional clásica.

EJEMPLO 12. Sea R(N, E, T, Ex) un esquema relacional de maestros experimentados, donde N es el nombre del maestro, E es

la edad, T es el tema enseñado, y Ex es la experiencia en la enseñanza de un tema en particular. Aquù, los valores de pertenencia pueden ser interpretados como la posibilidad de que un maestro enseñe un tema particular. Supongamos que en R se tienen las dependencias funcionales clásicas N --› E y N T --› Ex. Una instancia legal típica *r* de R se muestra en la tabla siguiente.

TABLA X

Nombre	Edad	Tema	Experiencia	μ
Rao	30	DBMS	4	0.8
Rao	30	AI	2	0.6
Johnson	35	AI	5	0.9
Sen	28	OS	3	0.7

Sea ρ = NE, NSE una descomposición de R. Las proyecciones de *r* sobre ρ se muestran en las tablas siguientes.

TABLA XI

Nombre	Edad	μ
Rao	30	0.8
Johnson	35	0.9
Sen	28	0.7

TABLA XII

Nombre	Tema	Experiencia	μ
Rao	DBMS	4	0.8
Rao	AI	2	0.6
Johnson	AI	5	0.9
Sen	OS	3	0.7

Puede verificarse fácilmente que $r = P_{NE}(r) \bowtie P_{NTEx}(r)$. También, la expresión E puede ser verificada mostrando que $\mu_{\hat{}rz}(t) = \mu_r(t)$ para todas las tuplas $t \in dom(N) \times dom(E) \times dom(T) \times dom(Ex)$, donde $\hat{}rz$ es la extensión cilíndrica de la proyección de r sobre los atributos N, T y Ex.

A pesar de que la expresión D es una condición necesaria y suficiente para una unión con menos pérdida de relaciones imprecisas, no puede ser aplicada en la práctica, para probar la unión con menos pérdida de una descomposición dada, como una prueba exhaustiva con todas las posibles combinaciones de los valores del dominio que se requieran. En esta sección utilizaremos esta condición para probar algunos teoremas concernientes a la menor pérdida en la descomposición-unión de relaciones imprecisas, en presencia de dependencias funcionales imprecisas.

V.6.1 DEPENDENCIA FUNCIONAL IMPRECISA Y MENOS PERDIDA EN LA DESCOMPOSICION-UNION.

En la teoría de bases de datos relacional clásica, es bien conocido que si una fd: X --> Y se presenta en una base de datos relacional R(X Y Z), entonces la descomposición $\rho = \{XY, YZ\}$ de R tendrá menos pérdida. Esta importante propiedad de fds forma la base de los procedimientos de normalización de Codd. A continuación mostraremos que a menos que la relación de

semejanza imprecisa EQUAL esté apropiadamente restringida, no se tendrá una propiedad similar para dependencias funcionales imprecisas ffds.

EJEMPLO 13. Consideremos el esquema relacional R(A B C) con una **ffd** : A ==› B. También supongamos que las funciones de pertenencia de la relación de semejanza imprecisa EQUAL usadas para definir esta ffd no distinguen los elementos de los dominios de los atributos A y B; por ejemplo, $\mu_{EQ}(a,a_1) = 1$ y $\mu_{EQ}(b,b_1) = 1$, para toda a, $a_1 \in dom(A)$ y b, $b_1 \in dom(B)$. Una instancia r de R que satisface esta ffd se muestra en la tabla siguiente.

TABLA XIII

A	B	C	μ
a	b	c	0.8
a	b_1	c	0.9
a	b	c_1	0.9
a	b_1	c_1	0.6

Sean R_1 (A B) y R_2 (A C) descomposiciones de R. Las proyecciones r_1 y r_2 sobre R_1 y R_2 se muestran a continuación.

TABLA XIV

A	B	μ
a	b	0.9
a	b_1	0.9

TABLA XV

A		Cμ
a	c	0.9
a	c1	0.9

Sea $r = r_1 \infty r_2$. Entonces podemos checar que μ_r (abc) = μ_r (a b_1 c_1) = 0.9, mientras que μ_r (a b c) = 0.8 y μ_r (a b_1 c_1) = 0.6. Como podemos observar, la descomposición de R basada en **ffd** : A ==› B no tiene menos pérdida.

Más aún, podemos mostrar que cuando la relación de semejanza EQUAL sobre dom(B), tiene μ_{EQ} (b, b_1) = 1 para cualquier b ≠ b_1, se puede encontrar una instancia r de R que satisfaga la **ffd**: A ==› B , y así la unión de las proyecciones de r sobre R_1 y R_2 no será igual a r.

A pesar de que la suposición que EQUAL es una relación de semejanza, es suficiente para la generalización de fds y para los axiomas de inferencia asociados, este ejemplo sugiere que para una menor pérdida en la síntesis de relaciones imprecisas con ffds, EQUAL necesita ser restringida aún más. En vista de ésto, en desarrollos subsecuentes la relacion de semejanza imprecisa EQUAL es restringida de tal forma que también satisfaga:

μ_{EQ} (a, b) < 1 para a ≠ b; a, b ∈ U ... EC. (F)

Así, para X ⊆ R(A_1 A_2 ... An), la expresión anterior podría implicar.

μ_{EQ} ($t_1[X]$, $t_2[X]$) < 1, si $t_1[X] \neq t_2[X]$ ... EC. (G)

donde t_1, t_2 ∈ D = dom(A_1) x dom(A_2) x ... x dom(An).

A partir de aquí, la clase de ffds donde EQUAL está

restringido por la expresión F , será referida como dependencia funcionál imprecisa restringida (**rffd**).

Note que si una relación imprecisa *r* satisface una rffd: A ==> B, entonces *r* no puede tener dos tuplas las cuales coincidan en el valor del atributo A pero difieran en el valor del atributo B, como sucede en el caso de dependencias funcionales clásicas. Sin embargo, a diferencia de las dependencias funcionales clásicas, esta rffd no permite que *r* tenga dos tuplas t_1 y t_2 con $t_1[A] \neq t_2[A]$, si:

$$\mu_{EQ}(t_1[A], t_2[A]) > \mu_{EQ}(t_1[B], t_2[B])$$

Asi, cuando en una relación imprecisa *r* se tiene una **rffd** *f*: X ==> Y, la dependencia funcional **fd** ~*f*: X --> Y también aparecerá en la misma relación.

La restricción impuesta por la expresión F , sin embargo, convierte una rffd en un limite de integridad más fuerte que en una dependencia funcional clásica. Sin embargo, aún con esta restricción adicional para EQUAL, las dependencias funcionales imprecisas resultan muy útiles en el modelado de límites de integridad que manejan la igualdad aproximada de los valores del dominio. Por ejemplo, las relaciones de semejanza usadas para modelar el límite de integridad 'Para cualquier artículo, la fecha del pedido MAS O MENOS determina la fecha de entrega' en el ejemplo 9 , satisfacen la expresión F . Sin embargo, algunas de las relaciones de semejanza en los ejemplos 7 y 8 no la satisfacen. Para la relación R(N,D,T,E,S,I) del ejemplo 8 , tenemos menos pérdida en la descomposición-unión ρ = NDT,TES,SI , podríamos seleccionar nuevas relaciones de semejanza sobre dom(E), dom(S) y dom(I), que satisfagan la expresión F.

El siguiente teorema muestra que bajo esta restricción adicional, la dependencia funcional imprecisa restringida rffd: X ==> Y nos conducirá a una menor pérdida en la descomposición-unión de R(X Y Z).

TEOREMA 6.1 Dado un esquema relacional R (A_1 A_2 ... A_n) con una rffd : X ==› Y, donde X, Y ⊆ R. El esquema relacional R tiene una menor pérdida en la descomposición-unión en dos componentes R_1 (X Y) y R_2 (X Z), donde Z = R - X Y.

Con dependencias funcionales clásicas, Rissanen ha establecido una versión del Teorema 6.1, de acuerdo a la cual, si R_1(X Y) y R_2(X Z) son una descomposición-unión con menos pérdida de R(X Y Z), con un conjunto de dependencias funcionales F, entonces ya sea X --› Y o X --› Z pueden ser inferidos desde F usando los axiomas de Armstrong. Sin embargo, el siguiente ejemplo muestra que un resultado similar no siempre se tiene con rffds.

EJEMPLO 14.- Consideremos un esquema relacional R(A B C) con dom(A) = a_1, a_2 , dom(B) = b_1, b_2 y dom(C) = c_1, c_2 . Definimos las relaciones de semejanza sobre estos dominios de la forma siguiente:

$$\mu_{EQ}(a_1, a_2) = 0.8 \ ,\mu_{EQ}(b_1, b_2) = 0.6, \ \mu_{EQ}(c_1, c_2) = 0.7$$

Note que la expresión F es satisfecha. Con esta selección de EQUAL, consideremos la rffd A B ==› C y una descomposición R_1(A B) y R_2(B C) de R. Aplicando la expresión E a cualquier instancia legal r de R, puede mostrarse que esta descomposición tiene menos pérdida de unión. Sin embargo, ni B ==› A ni B ==› C pueden ser inferidas desde A B ==› C, usando los axiomas de inferencia. Un examen exhaustivo revela que como en el ejemplo 10 , si una instancia r de R satisface A B ==› C, entonces r también satisface la rffd B ==› C. En otras palabras, para la selección de EQUAL actual, los axiomas de inferencia para ffd no están completos.

CONCLUSIONES.

Este capitulo trata con modelos de datos relacionales imprecisos, con el objeto de proveer una aproximación general para el tratamiento preciso e impreciso de los datos. Seleccionando las interpretaciones adecuadas para los valores de pertenencia, un modelo relacional impreciso es capaz de representar ambig"edades en los valores de los datos. Como uno de los principales objetivos de la lógica imprecisa es representar aproximadamente razonamientos empleados en lenguajes naturales, se espera que en el medio ambiente de las bases de datos, combinaciones apropiadas de un modelo de datos relacional y lógica imprecisa amplie las capacidades de los sistemas de bases de datos existentes. Un breve examen de algunas de las propuestas existentes para usar lógica imprecisa en un medio ambiente de bases de datos relacionales ha sido presentado.

Para una combinación exitosa de la teoría de conjuntos imprecisa y bases de datos relacionales; es, sin embargo, esencial desarrollar una técnica de diseño apropiada para tales sistemas. En suma, sería ideal si pudiéramos extender algunos de los resultados investigados ampliamente en la literatura de bases de datos relacionales clásica. En este punto, hemos examinado las propiedades de los operadores relacionales, especialmente relaciones imprecisas de unión y proyección.

Mientras intentamos extender las dependencias de datos en relaciones clásicas, se ha observado que para comparar valores del dominio, una medida imprecisa apropiada, tal como IGUAL puede resultarnos útil. Así, tratando 'IGUALDAD' como una relación de **semejanza** imprecisa, una **extensión simple** y natural de una dependencia funcional clásica, llamada dependencia funcional

imprecisa, ha sido propuesta. Se ha mostrado que una dependencia funcional imprecisa puede representar exitosamente límites de integridad que representan una igualdad aproximada de los valores del dominio. Así, los axiomas de inferencia para dependencias funcionales imprecisas (ffds) son similares a los axiomas de Armstrong para dependencias funcionales clásicas. Así mismo, hemos establecido la certidumbre de estos axiomas para una clase de ffds, por la imposición de una restricción simple en las relaciones de semejanza. Esta aparente similitud de los axiomas de inferencia es especialmente útil porque muchos algoritmos bien conocidos en la literatura clásica de dependencias funcionales, tales como los algoritmos de Beeri y Bernstein, pueden ser aplicados a las dependencias funcionales imprecisas para realizar tareas similares.

Para obtener la descomposición adecuada de esquemas relacionales, examinamos el problema de menos pérdida en unión de relaciones imprecisas, en presencia de ffds. Se observó que para alcanzar menos pérdida en la síntesis de esquemas relacionales, la relación de semejanza IGUAL necesita ser restringida aún más. Despues de introducir las restricciones apropiadas a IGUAL, se muestra que una clase de ffds se comporta exactamente como dependencias funcionales en relaciones clásicas. De hecho, la teoría de diseño de relaciones clásicas con dependencias funcionales se vuelve aplicable a relaciones imprecisas con esta clase de dependencias funcionales imprecisas.

Cabe mencionar que en este capitulo no pretendimos dar una exposición completa ó concluyente de las capacidades de las relaciones imprecisas en la captura de la semántica de los datos. Mas bien nos restringimos a una subclase de cálculos imprecisos donde los valores de verdad de una proposición imprecisa toman valores binarios.

VI. AFINACION DE BASES DE DATOS.

VI.1 AFINACION.

La afinación o "tuning" es una técnica en la cual pequeños cambios en la utilización de recursos pueden conseguirse haciendo cambios a parámetros de software.

Los objetivos de una afinación de Bases de Datos se enfocan a los siguientes aspectos:

- Tiempo de respuesta.
- Recursos.
- Uso del procesador.

Podemos calificar los objetivos de afinación por medio de los indicadores siguientes:

- Seguridad.
- Integridad.
- Flexibilidad.
- Productividad.

VI.2 PRINCIPALES CARACTERISTICAS DE RENDIMIENTO (PERFORMANCE)

Las principales características que nos describen el rendimiento del sistema son:

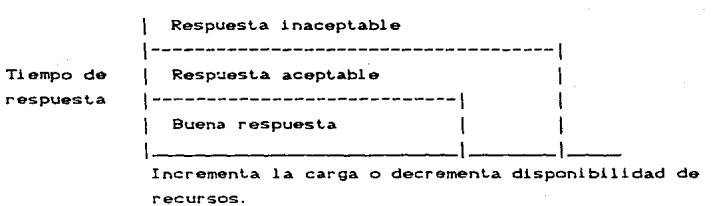
- Tiempo de respuesta.
- Carga máxima.
- Utilización de almacenamiento real y virtual.
- Uso del procesador.
- Utilización de almacenamiento físico.
- Utilización de redes.

El objetivo básico es alcanzar la carga deseada manteniendo la respuesta requerida. Así, la degradación del sistema implicará que

existe algún recurso limitado o algún "embotellamiento".

En la siguiente gráfica se ilustra la curva de rendimiento (performance) de un sistema. Puede apreciarse la relación que guarda el tiempo de respuesta del sistema, y el incremento de la carga o bien la disminución de la disponibilidad de recursos. Cuando la carga del sistema excede cierto límite, y los recursos disponibles son mínimos, se presenta una degradación del sistema, que se manifiesta como un crecimiento exponencial del tiempo de respuesta, obteniéndose así una respuesta inaceptable del sistema.

Curva de rendimiento (performance)



Cabe destacar que en circunstancias normales, la afinación no es una respuesta para un mal diseño y recursos inadecuados, y no permitirá un incremento significativo para una carga máxima.

Si pretendemos analizar el rendimiento (performance) de una Base de Datos, podemos tomar en cuenta los aspectos siguientes, que ampliaremos más adelante.

- Componentes del programa. Esto se refiere, por ejemplo, a evitar el exceso en el número de comandos.
- Medio ambiente de operación.
- Diseño de archivos y contenido.

- Lógica del programa de aplicación.

VI.3 NIVELES DE AFINACION.

Se puede hablar de cuatro niveles de afinación:

a) Afinación gruesa.
b) Afinación burda.
c) Afinación normal.
d) Afinación fina.

a) AFINACION GRUESA.- Se analizan las horas de tiempo de ejecución, así como cuantos registros fueron leídos, y cuantas llamadas se hicieron por transacción.

Las tareas por realizar son:

- Revisar el sistema.
- Aplicar los objetivos de la organización.

b) AFINACION BURDA.- Analizamos los minutos de tiempo de ejecución, y cuantas llamadas de I/O se manejaron. Las tareas por realizar en este nivel de afinación son:

- Prueba del sistema.
- Analizar la información de afinación.
- Aplicar los objetivos de la organización.

c) AFINACION NORMAL.- En la afinación normal analizamos los segundos de tiempo de ejecución, y un desempeño o funcionamiento tolerable. Para tal efecto, realizamos las tareas siguientes:

- Monitoreo de la Base de Datos.
- Analizar la información de afinación.
- Identificar los "embotellamientos".
- Eliminar o reducir los "embotellamientos".

d) AFINACION FINA.- En este nivel analizamos los milisegundos de tiempo de ejecución, así como aplicaciones en línea críticas.

Para tal efecto se procede a realizar un monitoreo del sistema, así como analizar la información de afinación, y afinar apropiadamente los componentes del sistema.

VI.4 PROBLEMAS DE RENDIMIENTO (PERFORMANCE) Y AFINACION.

I. S I N T O M A S.

Los síntomas de que una Base de Datos está funcionando en forma ineficiente, incluyen, entre otros, los aspectos siguientes:

- Tiempo de procesador excesivo (OVERHEAD), siendo acumulado a los programas del usuario.

- Alto promedio de tiempos de espera para lectura ó escritura, lo que se refleja en las estadísticas manejadas por el DBMS.

- Gran número de overlays normales ó forzados, mostrado en las estadísticas del DBMS.

- Tiempo excesivo requerido para efectuar SYNCPOINT/CONTROLPOINT. Cabe señalar que este concepto se emplea solo en algunos manejadores, como por ejemplo DMSII.

- Mala proporción de I/O lógicas a I/O físicas, calculada a partir de las estadísticas del DBMS.

- Buffers (SAVEMEMORY) excesivos requeridos por la Base de Datos.

- Baja proporción de transacciones contra la Base de Datos, a causa de una sola cola de espera (THREADING).

- Baja proporción de transacciones contra la Base de Datos, a causa de un subsistema de I/O saturado, o una carga no

balanceada.

- Rendimiento inferior contra algunos conjuntos de datos, debido a un gran número de conjuntos automáticamente asociados.

De aquí puede desprenderse la importancia de las estadísticas que se manejan, y la información que pueden arrojarnos.

Si cualquiera de los síntomas listados anteriormente, o cualquier otro problema se traduce en un bajo THROUGHPUT de la Base de Datos, o bien se detecta el uso excesivo de los recursos del sistema, puede recurrirse a los siguientes procedimientos para diagnosticar el problema.

- **P A S O 1 .**

a) ACOPIO DE INFORMACION.

Las principales fuentes de información acerca del rendimiento (performance) de la Base de Datos, se obtienen de las estadísticas emitidas por el DBMS, de las estadísticas para sistemas on-line, del análisis de la utilización de los recursos de la máquina, y a través del "debugging" de los programas del sistema.

Otras posibles fuentes de información son los archivos LOG del sistema, y en el caso de los programas batch, el sumario de tareas. Una técnica sugerida en el caso de sistemas on-line es tomar las estadísticas cada hora, mientras el sistema está activo.

Dentro de la información que suele ser de utilidad para proporcionarnos una orientación durante el proceso de afinación, podemos señalar:

- Conteo de las etapas de trabajo.
  . Cantidad de memoria principal alojada y utilizada.
  . Conteo de I/O a drives individuales.

. Conteo total de I/O a disco.
. Tiempo de etapas del CPU.

- Conteo de TP del sistema.
  . Resumen de cuentas de transacciones, almacenamiento, llamadas y tiempos de respuesta

Algunos puntos a considerar dentro del reporte de la Base de Datos son los siguientes:

- Fragmentación y áreas de espacio contiguo no utilizado.

- Chequeo de archivos que se hayan perdido en múltiples extensiones.

- Cuenta de registros.

- Examinar la relación entre el máximo ISN (Número secuencial de identificación) esperado y el número de registros cargados.

- Verificar las opciones especiales.

- Verificar la FDT (Tabla de Descriptores de Archivos), especialmente los descriptores (llaves).

Por sesión, es conveniente manejar las siguientes estadísticas:

- Llamadas al buffer.
- Sobreescritura de formatos.
- Llamadas por THREAD.

El histograma de valores descriptores y conteos se forma a partir de la información siguiente:

- Distribución de valores de datos.
- Longitud de la lista de ISN por valor.
- Número de valores.
- Valores no típicos ó rangos de valores.
- Precisión de valores (nomenclatura estándar o variaciones).

Resulta conveniente manejar las siguientes estadísticas de aplicación, para Bases de Datos:

- Usuario designado.
- Estadísticas generadas por el programa de aplicación.
- Conteo de actividad (registros leídos/actualizados).
- Información de transacciones on-line (usuario, terminal, fecha, tiempo, tránsito).
- Resultados (Número de corridas, errores).
- CPU total.
- Historia del funcionamiento.
- Estadísticas por sesión.
- Comandos totales.
- I/O totales por conjunto de datos.

b) MONITOREO DEL CPU

Se enfoca básicamente a dos aspectos:

- Medir el porcentaje de tiempo que cada partición/región está usando o esperando el CPU durante un periodo muestra.

- Identificar los rangos de direcciones dentro de un programa en los que se usan cantidades relativamente altas de CPU, y localizar estas direcciones, de ser posible, en una memoria más rápida.

c) MONITOREO DE I/O

Se enfoca a los siguientes aspectos:

- Balance de la actividad de I/O dentro de la base de datos, y entre la base de datos y otros sistemas.

- Conteo de I/O por drive.

- Promedio de la distancia de movimientos del brazo, por cada I/O.

- Porcentaje de tiempo ocupado.

- Promedio de tiempos de servicio de I/O.

Hasta aquí solo hemos mencionado algunos de los aspectos que deben tomarse en cuenta para la afinación del sistema. Más adelante trataremos más ampliamente algunos puntos que se desprenden de las estadísticas manejadas por el DBMS, su interpretación y valores óptimos.

P A S O 2..

ANALISIS DE INFORMACION.

Una vez que se ha recopilado la información del sistema de Base de Datos que nos ocupa, procedemos a su análisis. Es importante mencionar que tratamos de dar las pautas generales a seguir. Por esta razón, no nos referimos exclusivamente a un DBMS específico, y solo en algunos casos consideramos a un DBMS en particular, pero a modo de ejemplo. Además, aunque cada manejador tiene sus peculiaridades, en general poseen muchas similitudes, mismas que aprovechamos en este capítulo para que el análisis de la

información se plantee en forma general. La infomación por analizar previamente se clasifica y agrupa en la forma siguiente:

a) RECURSOS DEL PROCESADOR

El análisis de los recursos del procesador sobre el periodo monitoreado indicará donde está siendo usado el procesador. Idealmente, para tiempos de respuesta aceptables en sistemas on-line, el tiempo de procesador podría promediar 30% o más sobre un periodo extendido, digamos 3 horas o más. Si la utilización del procesador es excesivamente alta (digamos de un 70% total sobre un periodo extendido), y el procesador para tareas del usuario se contabiliza para una proporción medible de ésta, será necesaria una investigación más profunda para descubrir si la Base de Datos está causando OVERHEAD excesivo del procesador.

Para hacer ésto, debe obtenerse el tiempo de procesador promedio por transacción, para cada programa activo. Las estadísticas muestran cuántas transacciones fueron procesadas por cada programa, para una hora determinada, así como el tiempo de procesador usado por el programa en esa hora. Estos datos nos proporcionan el tiempo por transacción promedio. Si resulta mucho mayor que 100 milisegundos (por ejemplo en un equipo B6800/B6900), entonces puede deberse a varias causas:

- El tipo de transacción que fue procesado es muy complejo.
- Existe un código ineficiente en el programa.
- Las rutinas para manejo del buffer están siendo forzadas para hacer una cantidad excesiva de trabajo.

El "debugging" del programa podría indicarnos cual de las causas anteriores es verdadera. La complejidad del programa podría ser indicada si muchas secciones están siendo insertadas, pero poco tiempo del procesador está siendo acumulado para cualquiera de ellas.

Una excesiva manipulación de rutinas para manejo de buffer se

refleja como altos tiempos de procesador, para secciones que contienen verbos tales como FIND, LOCK, DELETE y otros, los cuales inician una búsqueda para un registro del conjunto de datos o un conjunto de entrada. La causa de esta situación es comúnmente una inadecuada memoria asignada para buffers, o tamaños de bloques innecesariamente grandes para conjuntos de datos.

En el histograma número 1 mostramos la relación entre el tiempo de procesador para tareas del usuario, el tamaño del bloque del conjunto de datos, y la memoria asignada.

La Base de Datos considerada consiste en 10 conjuntos de datos estándares, de 10,000 registros cada uno, y con una longitud por registro de 400 bytes, siendo ejercitada aleatoriamente por 10 programas a través de conjuntos indexados secuencialmente. Cada programa invoca y ejercita los diez conjuntos de datos. Puede observarse que la diferencia en la utilización del procesador, del peor caso al mejor caso, es un factor de 4.

En el histograma 2 mostramos la relación existente entre el tiempo de espera para lectura y el factor de bloqueo utilizado, para diferentes valores de memoria asignada, que van de 40,000 a 60,000 palabras. Como podemos observar, a mayor factor de bloqueo mayor será el rango de tiempos de espera. Por otro lado, a mayor memoria asignada, el tiempo de espera será menor.

Un indicador seguro de que la memoria asignada es demasiado pequeña es un alto porcentaje de OVERLAYS forzados. El mejor valor para ésto es cero, sin embargo un rango de overlays forzados de hasta 1 por segundo para todas las Bases de Datos activas concurrentemente es aceptable.

El histograma 3 nos indica la relación entre overlays forzados, y la memoria asignada, para un factor de bloqueo igual a 10.

b) TIEMPOS DE ESPERA PARA LECTURA / ESCRITURA

Un valor aceptable para un tiempo de espera promedio de lectura es de 60 milisegundos o menos. Para tiempos de espera de escritura podría ser de 40 milisegundos o menos. Si estos valores de rendimiento nominal son excedidos, la causa es comúnmente una de las siguientes:

- Memoria asignada muy pequeña.
- Tamaño del bloque del conjunto de datos muy grande.
- Colas excesivas de unidades (packs) a canales.
- Muy alta utilización del procesador, pero bajo porcentaje de tiempo de procesador.

El porcentaje de packs encolados puede ser determinado a partir de las estadísticas. Una cola promedio de hasta 0.4 es aceptable; hasta 0.6 es marginal, y arriba de 0.6 debe tomarse alguna acción correctiva. Algunas acciones podrían ser:

- Reducir la actividad de I/O a los packs de la Base de Datos, minimizando la actividad de OVERLAY, y los tamaños de los bloques del conjunto de datos.

- Distribuir la Base de Datos en más packs. Donde se utiliza más de un pack, éstos podrían ser configurados como una familia de un solo multipack.

Si por cualquier razon se desea usar un número de familias de packs individuales, con estructuras asignadas a packs particulares, debemos asegurarnos que los conjuntos de datos no tienen sus conjuntos de expansión en el mismo pack. Esto es para reducir el número de packs individuales encolados. Alguna experimentación probablemente será requerida, para distribuir sus estructuras de tal forma que la carga de I/O para cada unidad sea aproximadamente la misma.

c) PROPORCION DE BUFFERS EN MEMORIA TOTAL

En general, en un sistema Burroughs grande, la cantidad de buffers (SAVEMEM), no podría exceder el 40% de la memoria total (TOTALMEM). Si SAVEMEM se incrementa, el tamaño de cada área de trabajo por tarea disminuye, y se producirá un incremento en el código y en overlays de datos. Esto causa un incremento en I/O de packs, particularmente a la unidad de overlay. Al aumentar la cola en esta unidad disminuye la eficiencia de la función de overlay. Eventualmente los recursos del sistema se aplicarán primariamente al manejo de overlays. El histograma 4 ilustra ésto.

En el caso de un sistema on-line, cuando SAVEMEM ocupa el 40% de la memoria total, se inicia la degradación del sistema. La relación entre el tiempo de respuesta y SAVEMEM para un sistema on-line típico se ilustra en el histograma 5.

Recordemos que el stack de la Base de Datos y los buffers están en SAVEMEM, y no son sujetos a overlays.

Si el throughput del sistema total es afectado adversamente por una SAVEMEM excesivamente alta, puede mejorarse el rendimiento reduciendo los tamaños del bloque del conjunto de datos, permitiendo un decremento en la memoria asignada. Eliminar cualquier estructura redundante de la Base de Datos, y asegurar que los programas solo invoquen las estructuras que necesiten, también ayudará. Si con los procedimientos anteriores solo se consiguiesen ganancias insignificantes, debe considerarse la posibilidad de agregar más memoria física.

d) TIEMPO REQUERIDO POR SYNCPOINT / CONTROLPOINT

Los términos SYNCPOINT y CONTROLPOINT no son manejados por todos los DBMS. Entre otros, DMSII maneja el término SYNCPOINT para referirse al tiempo que transcurrirá para hacer un respaldo

de la información, y CONTROLPOINT para verificar que se hizo la copia de la información, y puede procederse a sobreescribir en el buffer.

Si las estadísticas de la Base de Datos reflejan un alto promedio de espera para SYNCPOINT / CONTROLPOINT, por ejemplo arriba de 1 segundo, entonces es aparente que uno o más programas están permaneciendo en el estado de transacción por periodos excesivamente largos.

e) MALA PROPORCION DE I/O LOGICAS A I/O FISICAS

De las estadísticas de la Base de Datos, es posible obtener el número total de lecturas físicas y el número total de lecturas lógicas. Para un sistema on-line típico, esto podría ser aproximadamente 1.8 - 2.5 : 1 . Si la proporción es mucho mayor que este modelo, conviene checar a través de estructura por estructura, para aislar aquellas estructuras que hacen I/O físicas excesivas.

Las dos causas más probables de ésto son:

1. Conjuntos con gran número de llaves duplicadas.
2. Programas que usan expresiones de selección parcial sobre conjuntos con llaves compuestas.

Ambas causas pueden ocasionar una gran distribución de I/O innecesarias, y uso de búsquedas lineales, lo cual consume más tiempo.

Dentro de DMSII, en el caso de programas batch, los cuales hacen recorrido secuencial a través de conjuntos de datos, ganancias muy sustanciales en la actividad de I/O física y lógica, y consecuentemente en el THROUGHPUT del sistema, pueden conseguirse usando el atributo físico "REBLOCK". Con ésto, modificamos el valor del factor de bloqueo. El histograma 6 muestra una ganancia del THROUGHPUT en un factor de 4 o más, que puede alcanzarse modificando el factor de bloqueo.

f) GRAN NUMERO DE CONJUNTOS DE EXTENSION SOBRE EL CONJUNTO DE DATOS.

Si un conjunto de datos tiene muchos conjuntos extendidos, un OVERHEAD está ocurriendo cada vez que un registro es agregado, borrado, o que su llave es modificada, ya que el DBMS debe localizar todas las entradas al conjunto para ese registro, y actualizarlas. Como una teoría general, los diseños que usan más de tres conjuntos de extensión deberían evitarse.

g) LOGICA DE DISEÑO MALA

Algunas consideraciones de diseño, referentes al rendimiento son:

- Evitar situaciones en las cuales se producirán cadenas de registros unidos durante la ejecución. Esto puede pasar con el uso extensivo de ligas, o bien con diseños jerárquicos de excesiva profundidad.

- Selección de los tipos de estructura apropiados. El histograma 7 nos da información relativa a las propiedades de conjuntos de datos estándar, conjuntos de datos random y conjuntos de datos directos. La selección de los tipos de estructura correctos mejorará el rendimiento.

- En general, el apego a una metodología de diseño relacional producirá una Base de Datos que satisfaga los criterios de rendimiento anteriores.

VI.5 CONSIDERACIONES DE AFINACION EN LAS DIFERENTES ETAPAS DE UN

SISTEMA DE BASE DE DATOS.

Podemos enfocar la afinación partiendo de las diferentes etapas que se siguen dentro del ciclo de vida de una Base de Datos, y que abarca desde aspectos de diseño e instalación, hasta la puesta en marcha de los programas de aplicación. Así, para cada etapa deben tomarse en cuenta ciertas consideraciones generales que contribuyen a afinar el sistema, y de este modo mejorar su rendimiento.

Aunque el concepto de afinación de Bases de Datos solo puede concebirse en un sistema de Bases de Datos en operación, cuando nos enfrentamos a un problema de afinación, es necesario analizar el sistema desde sus orígenes, e incluso desde la instalación del manejador de la Base de Datos (DBMS), y los parámetros iniciales de éste.

También suele ser necesario analizar el contenido y diseño de los archivos que se manejarán, contemplando aspectos tales como:

- Campos y número de éstos.
- Ubicación de los campos en los archivos.
- Colocación de los archivos en la Base de Datos.

La tarea de instalar el DBMS, y diseñar los archivos y su contenido, corre a cargo del administrador de la Base de Datos (DBA).

El usuario, por su parte, se ocupa de diseñar el programa de aplicación, mismo que deberá ser revisado durante el proceso de afinación de la Base de Datos, repasando la lógica de programación y las llamadas al manejador.

Otra etapa de afinación se ocupa de las pruebas sobre las aplicaciones de la Base de Datos. En este punto pueden considerarse las entradas y salidas físicas, y las llamadas directas.

También tenemos el monitoreo del funcionamiento del sistema, en el cual, como ya mencionamos, se identifican "embotellamientos" o "embotellamientos potenciales", así como procesos "en marcha".

Un aspecto que resulta de sumo interés en el proceso de afinación es la colocación y configuración del conjunto de datos que se manejará. Al respecto, existen ciertas consideraciones generales, como por ejemplo:

- Evitar la colocación de cualquier componente de la Base de Datos con cualquier conjunto de datos de alta actividad.
- Igualar I/O para tantos drives como canales se tengan disponibles.
- Cuando se usen dispositivos de diferente tipo, colocar los archivos accesados con más frecuencia, en los dispositivos más rápidos.

Por lo que se refiere al arranque del manejador de Bases de Datos, en esta etapa se siguen los pasos siguientes:

- Lectura y validación de todos los parámetros de control.
- Construcción del medio ambiente deseado.
- Carga de los módulos requeridos.
- Transferencia del control al módulo apropiado.

Al respecto, tenemos que algunas medidas de afinación que pueden hacerse en el arranque, serían:

- Cambiar un parámetro a la vez, y medir el efecto de ese cambio.
- Revisar y monitorear continuamente, especialmente después de la adición de una nueva aplicación.

Cabe señalar que los valores de la mayoría de los parámetros

afectarán los requerimientos centrales del DBMS.

Por ejemplo, en ADABAS existe un conjunto de parámetros de arranque, que pueden ser modificados durante el proceso de afinación. Tal es el caso del parámetro LBP, cuya función es invalidar el tamaño del buffer que por default se maneja, y que es de 80,000 bytes, e instalarlo nuevamente. Dentro de la afinación debemos considerar que LBP es el más importante parámetro de afinación durante el arranque, debido a su efecto sobre la I/O física. Si se presentara un gran número de interrupciones, o una excesiva paginación si LBP es muy grande, entonces serían indicadores de que existe un problema y es necesario afinar modificando este parámetro.

Algunas consideraciones pertinentes, a nivel de archivos, que deben tomarse en cuenta son:

- Determinar cuáles programas son los más críticos en cuanto a rendimiento (performance).
- Optimizar el rendimiento de cada función crítica.
- Usar supresión nula (NU).
- Usar opción fija (FI) si el contenido de los datos lo justifica.
- Colocar frecuentemente campos de lectura al principio del registro.
- Colocar los campos que no se lean, y solo se usen para criterios de búsqueda, al final del registro.
- Usar grupos de nombres para campos adyacentes, los cuales frecuentemente se lean juntos.
- Definir campos numéricos en el formato en el cual los registros serán más utilizados.
- No almacenar datos que puedan derivarse sin costo.

En lo que se refiere a los descriptores, conviene hacer algunas indicaciones:

- Un descriptor podría ser justificado en términos de su mantenimiento y espacio en disco.
- Un descriptor podría seleccionar un pequeño porcentaje de los registros en un archivo.
- Conviene usar un número pequeño de descriptores en un archivo que contenga un gran número de registros agregados y borrados.
- No deben crearse descriptores que serán frecuentemente actualizados.

Ahora analizaremos lo referente a la afinación de los programas de aplicación.

Primero definimos una transacción lógica como el tiempo para realizar los comandos necesarios del DBMS, y que se inicia con la colocación del primer registro en espera, y termina hasta que se indica la completación exitosa de la transacción, por medio de un comando de terminación.

La conveniencia de transacciones lógicas cortas se aprecia básicamente en cuando al uso compartido de recursos, para hacer más cortos los periodos de tiempo. Así, tenemos que "una transacción lógica debe ser la unidad aceptable más pequeña, para asegurar la integridad de los datos."

Una Base de Datos afinada debe evitar la redundancia, siempre y cuando con ésto no se pierda la integridad de los datos. Para tal efecto, pueden seguirse las recomendaciones siguientes:

- Solo leer los campos que sean actualmente requeridos por el programa. Usar un comando de búsqueda para leer el primer registro encontrado.
- Usar una lectura secuencial física cuando la I/O física

requerida para leer el archivo completo sea menor que la que se requeriría para una lectura secuencial lógica, o una opción "lee y encuentra".
- Evitar loops tales como "PERFORM READ -NEXT -RECORD UNTIL DBMS-END-OF-FILE".

Finalmente, damos algunos puntos que podrían resultarnos de gran utilidad al momento de afinar la Base de Datos:

- Minimizar el número de llamadas al DBMS.
- Usar nombres en grupo.
- Leer un número probable de ocurrencias periódicas en grupo.
- Solo leer los campos actualmente necesitados por el programa.
- Usar una lectura secuencial lógica más que una opción lee/encuentra para un rango de valores de un solo descriptor.
- Usar algoritmos de búsqueda simples.
- No usar la opción "GET NEXT" cuando se tenga un gran número de ISN's (Números secuenciales de identificación de registros).
- Omitir el comando "OPEN" a menos que sea necesario para establecer el tipo de usuario, su identificación, o una prioridad de procesamiento especial.
- Considerar prioridad de procesamiento especial para tareas críticas.
- Utilizar utilerías de actualización masivas, más que comandos de adición y eliminación de registros individuales, cuando los registros por agregar o eliminar representan más del 12% de los registros totales en un archivo.

VI.6 E S T A D I S T I C A S.

Volviendo nuevamente a las estadísticas del DBMS, convendría

profundizar un poco más en cada una de ellas, indicando algunos de los parámetros principales que se manejan, y la forma en que pueden tratarse para la afinación de la Base de Datos, así como la interrelación que puede presentarse entre algunos de estos parámetros.

Las estadísticas arrojadas por el DBMS pueden ser clasificadas como sigue:

- Estadísticas del buffer.
- Estadísticas de Entrada / Salida.
- Estadísticas de uso de la Base de Datos.
- Estadísticas de auditoria.
- Estadísticas de transacción.

A continuación, trataremos mas a fondo cada una de ellas, incluyendo algunos conceptos que se manejan y son comunes a varios manejadores, aunque la terminología empleada por cada uno pueda ser diferente.

a) ESTADISTICAS DEL BUFFER

Dentro de estas estadísticas suelen manejarse conceptos tales como:

ALLOWEDCORE.- Es el ambiente comúnmente en uso por las rutinas de acceso, y representa la memoria permitida, o autorizada para ser usada.

La variable ALLOWEDCORE puede ser utilizada para ayudar a conservar el promedio de OVERLAY bajo.

La experimentación es la mejor técnica para seleccionar un buen valor de ALLOWEDCORE.

Si fijamos ALLOWEDCORE a un valor absurdamente alto (100 K o más), después de que la Base de Datos ha estado corriendo por algún tiempo, checamos el máximo almacenamiento de buffer usado. A continuación, cambiamos el valor de ALLOWEDCORE a un valor

ligeramente menor que el máximo almacenamiento de buffer usado. Después de que la Base de Datos ha estado corriendo por un periodo más largo, buscar el número de overlays ocurridos durante ese lapso. Si el valor es razonable, el ambiente de ALLOWEDCORE es apropiado. De otra forma se requerirá un valor mayor.

Cabe señalar que, como su nombre lo indica, el máximo almacenamiento del buffer usado, es la cantidad de memoria máxima usada para satisfacer los requerimientos de buffer para la Base de Datos, durante su intervalo de tiempo estadístico.

El número de OVERLAYS forzados indica el número de veces que las rutinas de acceso desasignan un buffer a causa de que la cantidad de memoria usada para los buffers excede la cantidad de memoria permitida (ALLOWEDCORE).

Debe tenerse cuidado para prevenir un promedio de OVERLAY excesivo; sin embargo, esto puede resultar en el uso más memoria. Un promedio de OVERLAY de aproximadamente 1 por segundo para todas las Bases de datos activas concurrentemente es considerado razonable, únicamente para medidas de transacción bajas. Un promedio de OVERLAY cercano a cero sería el ideal.

Cuando un buffer es superpuesto (OVERLAYED), o sea escrito en disco y desasignado, o quizá solo desasignado, todos los stacks de proceso son buscados para copiar los descriptores al buffer. Esto puede impactar al sistema completo, si muchos stacks de proceso están presentes.

Por otra parte, el número de OVEPLAYS normales indica el número de veces que las rutinas de acceso desasignan un buffer, por las siguientes razones:

- Una estructura está cerrada.
- Un usuario cambia de modo serial a modo random.
- Un usuario cambia de modo random a modo serial.

b) ESTADISTICAS DE ENTRADA / SALIDA

En esta parte del reporte se incluye información acerca de I/O físicas contra la Base de Datos, para cada estructura que ha sido abierta al menos una vez.

Esta sección está dividida en dos partes: información perteneciente a lecturas, seguida por información perteneciente a escrituras.

Así, tendremos información como tiempo de I/O en segundos, que representa el tiempo de I/O total, usado para accesar los registros de la estructura, excluyendo tiempos de búsqueda, pero incluyendo latencia y tiempos de transferencia de datos. También se maneja el número total de lecturas, que es incrementado para cada lectura física, y el tiempo de espera total para lectura (en segundos), que es el tiempo acumulado por todos los programas que esperan por lecturas físicas completadas.

El procedimiento es el siguiente: después de que ha sido determinado por las rutinas de acceso que el registro deseado no está en un buffer de memoria, una I/O para la lectura es iniciada. La solicitud es procesada por el sistema operativo. Más que tener que esperar las rutinas de acceso para que la I/O se complete, el buffer anterior es examinado, y si este buffer ha sido modificado, y los cambios han sido escritos en el archivo de auditoría (LOG), entonces las rutinas de acceso inician la escritura de esta buffer. Este proceso es conocido como WRITEAHEAD. Si la I/O para la lectura original no ha sido completada aún, el usuario será puesto "a dormir", y el tiempo de espera para lecturas empieza a acumularse. Si la I/O para la lectura original ha sido completada antes de la terminación de estas actividades, entonces no se acumulará tiempo de espera.

Otro parámetro manejado es el promedio de tiempo de espera para lecturas (en milisegundos). Un promedio razonable es aproximadamente de 50 a 60 milisegundos por lectura. Un factor que

puede contribuir a un promedio mas alto de tiempo de espera para lectura es un bloque de datos excesivamente grande, para ser transferido.

La siguiente información se refiere a escrituras físicas a la Base de Datos.

- Número de escrituras para la estructura, y se incrementa para cada escritura física.
- Número de WRITEAHEADS para la estructura.
- Tiempo de espera total para escrituras (en segundos), que es el tiempo acumulado mientras una tarea está esperando que una escritura sea completada.
- Tiempo de espera promedio para escrituras (en milisegundos). Un promedio de tiempo de espera para escrituras de aproximadamente 15 a 25 milisegundos es considerado razonable.

Algunos de los factores que podrían influir para un tiempo promedio de escritura mayor, son:

-Muchos buffers están siendo escritos por el procedimiento de CONTROLPOINT.
- El procedimiento de OVERLAY está ocurriendo también muy frecuentemente.
- Mucho tiempo de I/O está siendo consumido en la transferencia de bloques de datos muy grandes.

c) ESTADISTICAS DE USO DE LA BASE DE DATOS

Estas estadísticas, junto con las estadísticas de I/O, pueden ayudar al DBA y al programador, a determinar posibles problemas de diseño y programación.

En esta sección se indica el número de enunciados del DBMS, realizados contra cada estructura, y en forma similar a las estadísticas de I/O, el nombre de cada estructura es listado en

una columna, y los encabezados indican la actividad de los enunciados que son reportados.

d) ESTADISTICAS DE AUDITORIA (LOG)

En esta sección, el reporte de las estadisticas no aparecerá para bases de datos no auditadas.

El tamaño del bloque promedio (en palabras) indica el número de palabras promedio en un bloque auditado, comparado con el número de palabras máximo que pueden tenerse en un bloque auditado.

Un bloque LOG es escrito de uno de dos buffers de inspección de memoria. Un bloque es escrito cuando el buffer de inspección está lleno. Un bloque de inspección parcialmente lleno es escrito en el tiempo del SYNCPOINT, o bien cuando la cantidad de memoria para buffers presentes excede la memoria asignada (ALLOWEDCORE), y las rutinas de acceso necesitan reusar un buffer modificado que no puede ser escrito en disco (debido al hecho de que la información inspeccionada para el buffer no ha sido fisicamente escrita el medio de auditoria).

Cuando inspeccionamos al disco, el promedio se referirá al tamaño del bloque actual, porque los bloques LOG parcialmente llenos son reusados.

El último bloque LOG puede estar parcialmente lleno, contando para el tamaño promedio del bloque, siendo ligeramente mas pequeño que el tamaño del bloque actual.

Cuando auditamos cinta, los bloques parcialmente llenos no son reusados, contando para el tamaño promedio del bloque, siendo mas pequeño que el tamaño del bloque actual. Si el tamaño promedio del bloque LOG es mucho mas pequeño que el tamaño del bloque actual, ya sea que se reduzca el tamaño del bloque LOG, o se incremente el

número de transacciones que ocurren por SYNCPOINT.

TIEMPO PROMEDIO DE ESPERA DE I/O, indica el tiempo de espera promedio para escrituras al archivo de auditoría (LOG). Cuando auditamos una cinta, un tiempo de espera razonable es de aproximadamente 3 a 8 milisegundos. Cuando auditamos un pack, de 10 a 15 milisegundos o menos, es bueno. Si el tiempo de espera promedio es alto, es posible que el archivo de auditoría esté siendo escrito muy frecuentemente. El tamaño del bloque LOG puede ser también pequeño, originando que se llene muy rápidamente; por consiguiente, se requerirá que el buffer de auditoría lleno sea escrito al archivo de auditoría. El valor del SYNCPOINT podría ser muy pequeño, causando que el buffer de auditoría sea vaciado muy frecuentemente. El buffer de auditoría puede ser escrito al archivo de auditoría cuando la memoria total excede la memoria asignada. Si hay un alto número de OVERLAYS, incrementando el valor de la memoria asignada se podría reducir el número de OVERLAYS, y el promedio de tiempo de espera de I/O, sobre el archivo de auditoría (LOG).

e) ESTADISTICAS DE TRANSACCION

- CUENTA TOTAL DE TRANSACCIONES, es una variable que es incrementada en cada inicio de transacción. El valor refleja el número total de veces que los usuarios entraron al estado de transacción. Si la transacción es abortada y no se completa, la cuenta de transacciones no es decrementada.

- CUENTA TOTAL DE SYNCPOINT, es el número total de veces que el procesamiento de syncpoint fue realizado. No incluye syncpoints que causan procesamiento de controlpoints.

- CUENTA TOTAL DE CONTROLPOINT, es el número total de veces que el procesamiento de controlpoint fue realizado.

- CUENTA DE ESPERA PARA SYNCPOINT / CONTROLPOINT, es el número

total de veces que las tareas fueron " adormecidas " esperando que terminara el syncpoint o el controlpoint.

Cuando una tarea es "adormecida", esperando la terminación de un syncpoint o un controlpoint, su tiempo de espera es acumulado. este tiempo acumulado es dividido por la CUENTA DE ESPERA PARA SYNCPOINT / CONTROLPOINT, produciendo el PROMEDIO DE ESPERA PARA SYNCPOINT / CONTROLPOINT.

Los CONTROLPOINT típicamente toman más tiempo para completarse que los SYNCPOINT; por lo tanto, este tiempo de espera promedio puede ser severamente influenciado por el procesamiento de CONTROLPOINTS.

Cuando las rutinas de acceso inician el procesamiento del CONTROLPOINT, el tiempo gastado para completarlo es acumulado. Este tiempo es dividido entre la cuenta total de CONTROLPOINT, para producir el TIEMPO PROMEDIO POR CONTROLPOINT. Esto incluye el tiempo para escribir el buffer auditado presente, escribir todos los buffers modificados no vaciados en el CONTROLPOINT previo, y escribir la información sobre el control del almacenamiento.

El NUMERO PROMEDIO DE BUFFERS VACIADOS EN EL CONTROLPOINT indica el número promedio de los buffers modificados, escritos a disco en el CONTROLPOINT.

El PORCENTAJE DE BUFFERS MODIFICADOS, VACIADOS EN EL CONTROLPOINT, especifica el porcentaje de todos los buffers modificados que fueron escritos a disco, como resultado del procesamiento del CONTROLPOINT. El mejor valor para esta estadística es 0%, indicando que todos los buffers modificados fueron escritos a disco por el procedimiento de WRITEAHEAD, por el cierre de la Base de Datos, o por el procedimiento de OVERLAY. Controlando el tamaño del buffer de LOG, el número de buffers por estructura, la frecuencia del SYNCPOINT, y la frecuencia del CONTROLPOINT, este valor puede mantenerse bajo. Un valor del 50%

es el peor de los casos, porque indica que la mitad de los buffers modificados son escritos en CONTROLPOINT, y no están siendo escritos por el procedimiento de WRITEAHEAD.

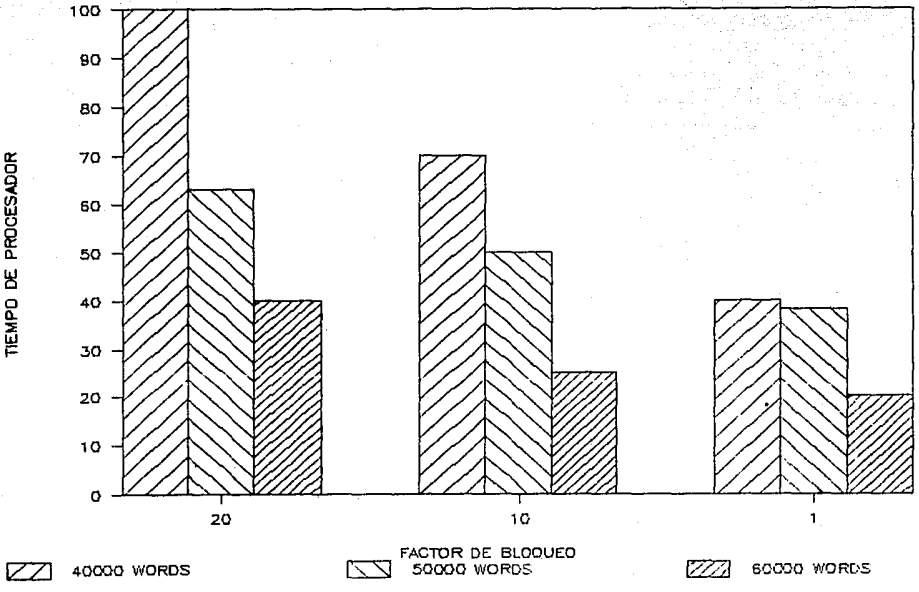
De todo lo expuesto a lo largo de este capítulo, resulta evidente la importancia del proceso de afinación de una base de datos en operación.

En un sistema de bases de datos afinado, se obtendrá el máximo rendimiento (performance) posible, de acuerdo a las características de hardware y software del sistema. Es decir, la afinación nos permite optimizar los recursos del sistema.

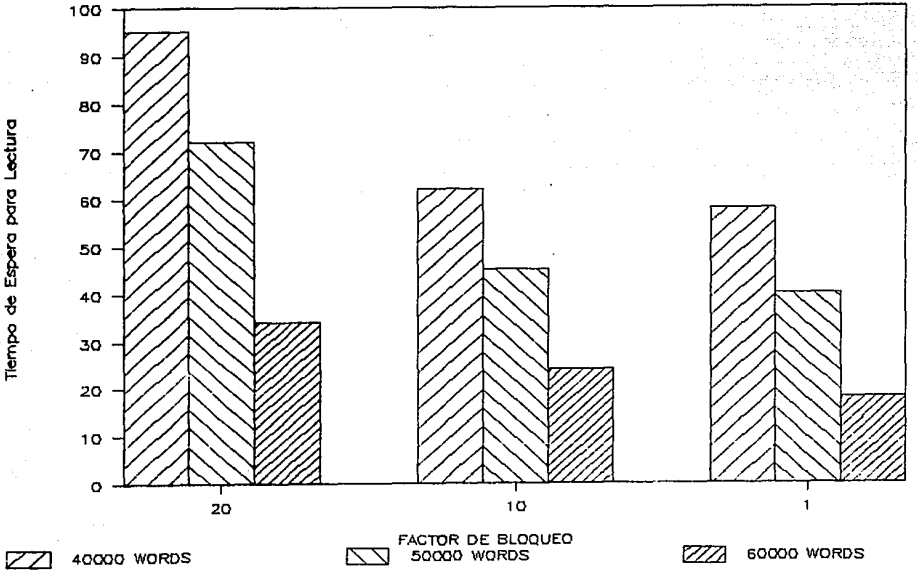
La labor del DBA puede resultar compleja, y es necesario que tenga un amplio conocimiento de la forma en que cada uno de los parámetros del sistema afecta el rendimiento del mismo, así como los efectos que puede tener en los parámetros del sistema, la modificación de un parámetro determinado.

El sistema de afinación de bases de datos que proponemos en el presente trabajo se basa, como ya dijimos anteriormente, en bases de datos imprecisas. Hasta el momento, tenemos los fundamentos teóricos de las Bases de Datos Imprecisas, y podemos pasar a diseñar una aplicación práctica de las mismas, que a la vez sirva como una herramienta de afinación para el DBA. En el capítulo siguiente procederemos a explicar el diseño del sistema.

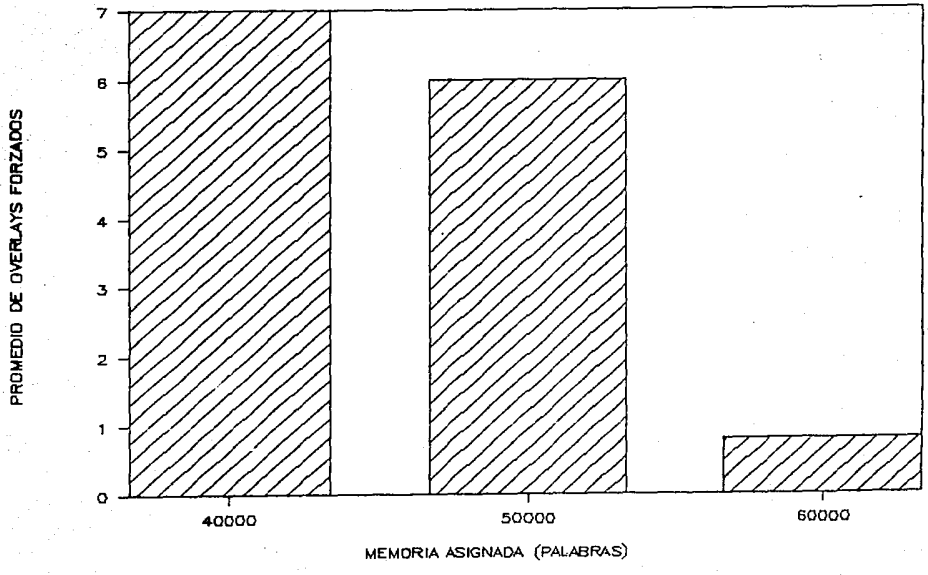
H I S T O G R A M A 1



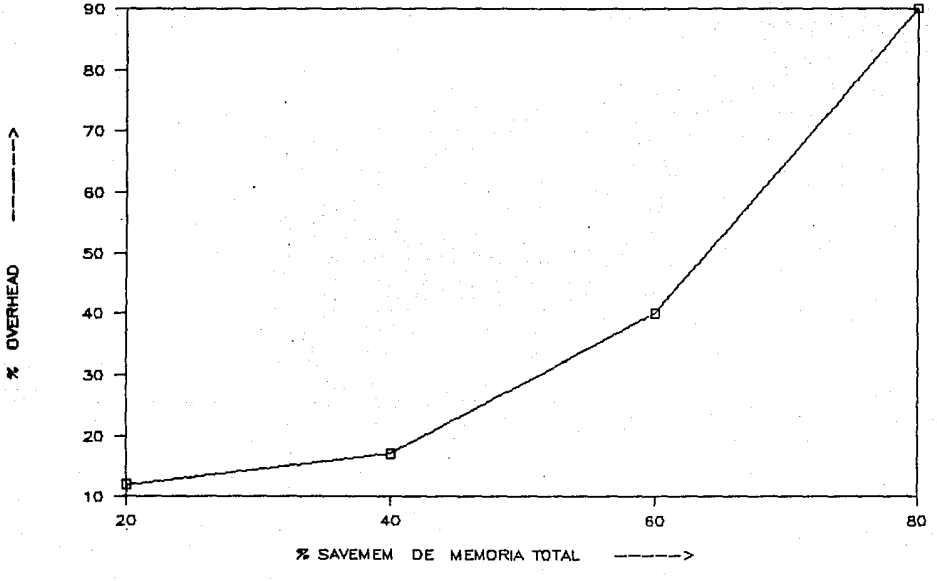
HISTOGRAMA 2



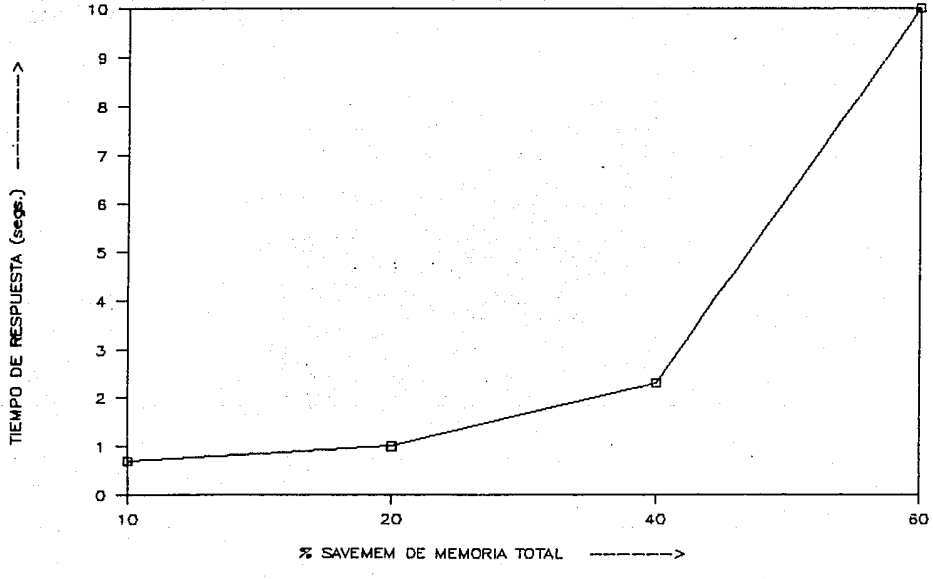
HISTOGRAMA 3

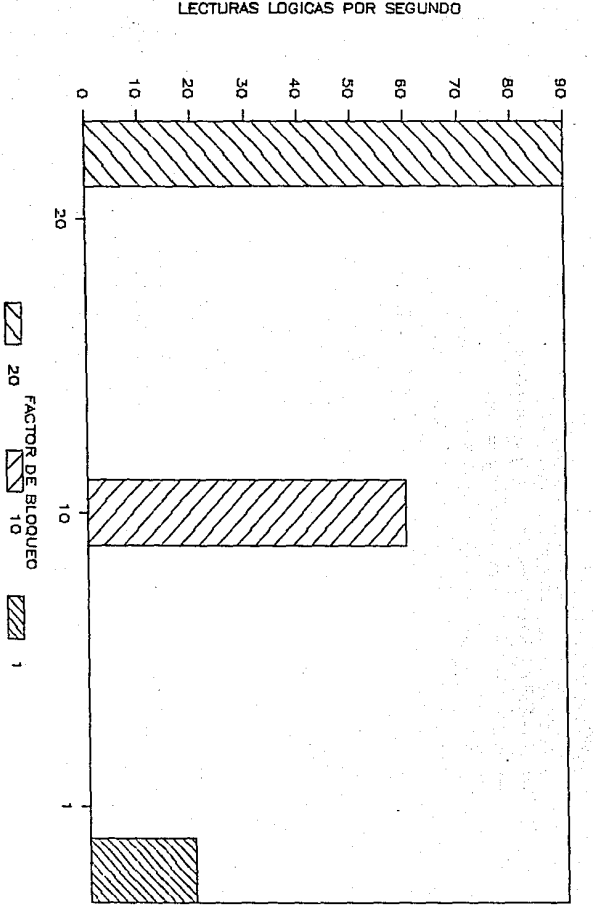


H I S T O G R A M A 4

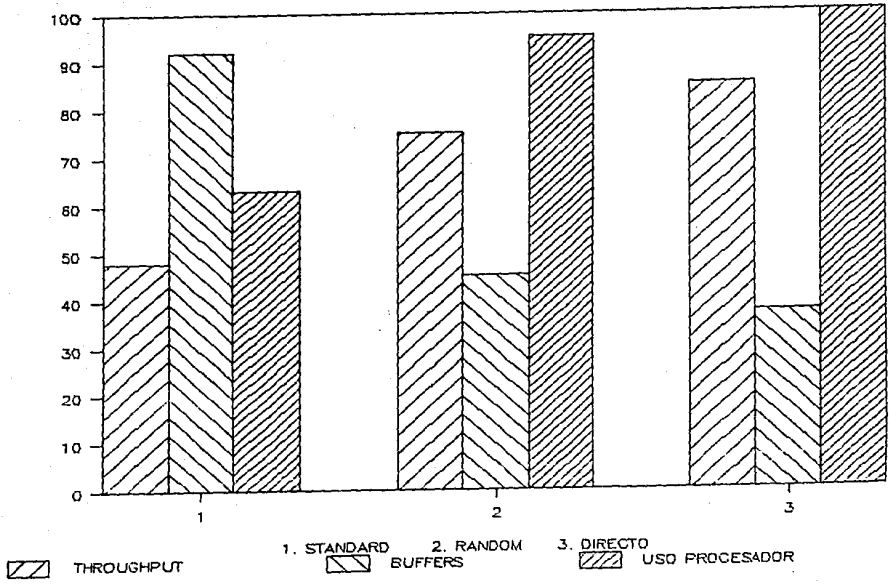


HISTOGRAMA 5




LECTURAS LOGICAS POR SEGUNDO
0
10
20
30
40
50
60
70
80
90
20
10
1
FACTOR DE BLOQUEO
20
10
1
HISTOGRAMA 6

HISTOGRAMA 7



VII. DISEÑO E IMPLEMENTACION DE UN SISTEMA DE AFINACION DE BASES DE DATOS

Hasta el momento hemos sentado las bases teóricas que nos permiten diseñar una novedosa aplicación de las Bases de Datos Imprecisas, consistente en un sistema que funja como una herramienta de afinación para el Administrador de la Base de Datos.

De las características observadas en diversos manejadores trabajando en equipos mini y mainframe, dedujimos en primer lugar la necesidad de agrupar a los manejadores en dos grandes clasificaciones: DBMS para minicomputadoras, y DBMS para equipos mainframe. Cabe hacer la aclaración que aunque existen diversos DBMS para microcomputadoras, nosotros no los consideraremos en virtud de que en estos sistemas no suele realizarse el proceso de afinación, como lo hemos estado manejando hasta el momento. En sí, el concepto de afinación involucra un ambiente multiusuario, y para un DBMS funcionando en microcomputadoras, la afinación se limitaría a que el usuario aprovechara de la mejor manera posible los recursos de la máquina. Es decir, el DBA resulta ser el mismo usuario.

En primer término, establecimos que no resultaba conveniente diseñar un mismo Sistema de Afinación para equipos mini y mainframe, en virtud de la gran diversidad de configuraciones que se pueden presentar, aún dentro de la misma clasificación, sin

olvidar también que cada DBMS presenta características muy peculiares que lo diferencian del resto de los manejadores.

Consideramos que una buena solución era avocarnos a analizar una clasificación, para poder extraer de la información obtenida algunos parámetros que resultaran comunes a varios manejadores. Esto de ninguna manera originará un sistema restringido a equipos pertenecientes a esta clasificación, ya que el sistema quedará estructurado de tal forma que resulte sencillo cambiar los valores de los parámetros de afinación considerados, e inclusive eliminar los que no se consideren representativos del sistema en cuestión, o añadir nuevos parámetros.

Con esto, podemos obtener un sistema de afinación que se adecúe a las características del sistema de Base de Datos que se desea afinar.

Elegimos los equipos mainframe para nuestro análisis preeliminar. Los manejadores que observamos fueron ADABAS, y DB2. El procedimiento a seguir fue realizar entrevistas con las personas encargadas de trabajar con la Base de Datos, para que nos informaran qué parámetros consideraban más significativos para reflejar el rendimiento (performance) del sistema. Tomamos como base los parámetros generales que fueron tratados en el capitulo anterior, para de éstos seleccionar los que, al menos para los dos DBMS considerados, resultaran ser más representativos.

También se investigaron los rangos que para cada uno de los parámetros se consideran aceptables o adecuados para el entorno de

la base de datos.

De la información obtenida logramos desprender los siguientes parámetros de afinación:

1. **Tiempo de respuesta.** Este parámetro es considerado como el más representativo del estado del sistema, y es así mismo el que más preocupa al DBA. Consideramos que un tiempo de respuesta de hasta 5 segundos es aceptable.

2. **Porcentaje de packs encolados.** El número de packs encolados en espera de ser accesados para operaciones de I/O afecta significativamente el performance de la Base de Datos, ya que al ir aumentando la cantidad de packs encolados, el tiempo de respuesta se ve severamente afectado. Un porcentaje del 60% de packs encolados, del total de los mismos, es considerado aceptable.

3. **Número de registros accesados.** En este parámetro resulta determinante el tamaño de los registros, así como el tamaño del bloque manejado. Un bloque es un conjunto de registros lógicos presente en un área de disco. Visto de otra forma, es el tamaño estándar en que se accesa la información del disco. Si dividimos el tamaño del bloque promedio aceptable entre el tamaño promedio de cada registro, obtenemos el número óptimo de registros accesados. Como el tamaño de los registros varía para cada aplicación, preferimos manejar esta característica como una variable. Más adelante explicaremos esto con más detalle.

4. **Porcentaje de buffers en memoria total.** El porcentaje de memoria destinada a buffers, del total de la memoria disponible

también afecta directamente el performance del sistema, puesto que determina el número de overlays que se producirán. Se considera aceptable dentro de un rango de hasta 40%.

5. **El intervalo de espera para que se lleven a cabo SYNCPOINT y CONTROLPOINT.** Este parámetro fue indicado como otro de los parámetros que deben ser tomados en cuenta al momento de la afinación. Se nos indicó que intervalos mayores a una hora resultan convenientes ya que no aumentan mucho el tiempo de respuesta.

6. **Número de conjuntos de extensión.** El número de conjuntos de extensión máximo que manejan la mayoría de los DBMS es de tres, aunque dependiendo de la configuración, algunos equipos soportan hasta cinco conjuntos de extensión, sin que se aprecie una degradación mayor del sistema.

7. **Tamaño del bloque promedio.** Para varios manejadores, el tamaño del bloque promedio es de 4 K. Este es un parámetro que suele ser también muy significativo ya que en ocasiones al variarlo,el rendimiento del sistema presenta diferencias sustanciales. Es por ello que algunos DBMS cuentan con opciones para que, a través de un comando determinado, pueda variarse el tamaño del bloque promedio, y se pueda determinar si el valor antes manejado estaba obstaculizando de algún modo el performance del sistema.

8. **Tamaño máximo del bloque.** Obtuvimos que oscila entre 28K y 32K. Por ejemplo, para DB2, el tamaño promedio que se maneja (por default) es de 4 K, pero puede variarse hasta un máximo de 32 K.

Es importante mencionar que en la mayoría de los parámetros de afinación los valores óptimos suelen encontrarse en la práctica, cuando el DBA se enfrenta a problemas de performance, y tiene que descubrir en donde está la falla, o el obstáculo que afecta al resto de los parámetros, impidiendo que converjan hacia el logro de un fin común: ofrecer al usuario un sistema que responda a las expectativas girando en torno a éste.

Con el sistema que diseñamos, más que tratar de fijar los valores óptimos de cada uno de los parámetros, pretendemos mejor explicar las ideas fundamentales del porqué manejamos Bases de Datos Imprecisas, y cómo lo hicimos. En cuanto a los valores optimos de los parámetros de afinación, como ya dijimos, el sistema será modular y estructurado para facilitar que en cualquier momento se cambie el valor límite considerado "aceptable", y se pueda evaluar, para los nuevos límites, el rendimiento que el sistema presenta.

Una vez que establecimos los parámetros a manejar, así como los valores que se consideran aceptables, podemos pasar al diseño de la Base de Datos Imprecisa, en la cual vamos a almacenar la información concerniente a los rangos de valores que, para cada uno de los parámetros de afinación, forman parte de un sistema que funciona con un alto rendimiento (performance).

Definimos primero una Base de Datos que contendrá tantos atributos como parámetros de afinación se manejen. En nuestro caso, como ya mencionamos, consideraremos ocho parámetros de afinación.

Así, nuestra Base de Datos estará formada por ocho atributos:

Parámetro 1, Parámetro 2,Parámetro 3,Parámetro 4,Parámetro 5, Parámetro 6, Parámetro 7,Parámetro 8,donde para cada atributo tendremos un dominio: dom (Parámetro 1), dom (Parámetro 2), dom (Parámetro 3),dom (Parámetro 4),dom (Parámetro 5),dom (Parámetro 6), dom (Parámetro 7),dom (Parámetro 8). Si usáramos una Base de Datos Relacional Clásica, el dominio de cada parámetro sería un conjunto clásico. Sin embargo, manejaremos los dominios como conjuntos imprecisos. Los conjuntos imprecisos serán:

dom (Parámetro 1) = "ALTO RENDIMIENTO DEL PARAMETRO 1"
dom (Parámetro 2) = "ALTO RENDIMIENTO DEL PARAMETRO 2"
dom (Parámetro 3) = "ALTO RENDIMIENTO DEL PARAMETRO 3"
dom (Parámetro 4) = "ALTO RENDIMIENTO DEL PARAMETRO 4"
dom (Parámetro 5) = "ALTO RENDIMIENTO DEL PARAMETRO 5"
dom (Parámetro 6) = "ALTO RENDIMIENTO DEL PARAMETRO 6"
dom (Parámetro 7) = "ALTO RENDIMIENTO DEL PARAMETRO 7"
dom (Parámetro 8) = "ALTO RENDIMIENTO DEL PARAMETRO 8"

Cabe señalar que manejaremos el modelo de datos relacional impreciso del tipo 1, donde el dominio de cada atributo puede ser un subconjunto relacional clásico o bien un subconjunto impreciso. En nuestra base de datos imprecisa, todos los dominios son subconjuntos imprecisos.

Así, la Base de Datos almacenará los valores de los parámetros de afinación obtenidos por estadísticas o por monitoreo:

	Par.1	Par.2	Par.3	Par.4	Par.5	Par.6	Par.7	Par.8	μ
Prueba 1									
Prueba 2									
.									
.									
.									
Prueba n									

Prueba 1, prueba 2, ... prueba n corresponden a cada vez que se analiza el performance del sistema. La idea es que cada vez que se modifica el parámetro o los parámetros que están afectando negativamente el rendimiento del sistema, se vuelva a calcular el valor de verdad de la proposición imprecisa: "El sistema de Base de Datos analizado presenta un alto rendimiento para todos los parámetros de afinación considerados".

Hasta aquí, parece una Base de Datos relacional clásica, a no ser porque se le da una interpretación probabilística a cada tupla, y se le asigna una columna μ, donde se anota, para cada estado de la base de datos, el grado de pertenencia de la tupla, al conjunto impreciso "ALTO RENDIMIENTO".

Como definimos un conjunto impreciso para cada parámetro, el siguiente paso es calcular el grado de pertenencia (μ) del valor de cada atributo, al conjunto impreciso definido como su dominio. Para calcular los grados de pertenencia, fue necesario definir

las funciones para los grados de pertenencia de cada atributo.

No hay una regla para obtener las funciones que definan los grados de pertenencia de un valor, hacia cierto conjunto impreciso determinado, aunque, como vimos en el capitulo V, existen ciertos modelos. No obstante, cada caso es diferente, y la función dependerá de la amplitud del rango que se vaya a manejar. Por lo general se hace en forma intuitiva, o experimentando con diferentes funciones hasta encontrar la que mejor se ajuste para definir el conjunto impreciso: ALTO RENDIMIENTO DEL PARAMETRO i.

La estructura general de las funciones para definir el grado de pertenencia que manejaremos es:

$$\mu = 1 \;/\; (1 + |\text{parametro}_i - \text{limite}|/\text{divisor})$$

El límite es el valor máximo que puede tener el atributo, para pertenecer al conjunto impreciso, y el divisor determinará los alcances o la flexibilidad del conjunto impreciso.

Por ejemplo, para el parámetro **tiempo de respuesta**, el dominio será el conjunto impreciso: 'ALTO RENDIMIENTO DEL TIEMPO DE RESPUESTA', y se define a través de la función:

$$\mu_{R1} = 1 \;/\; (1 + | \text{parametro } 1 - \text{limite} | \;/\; \text{divisor})$$

Se considera un tiempo de respuesta dentro del conjunto impreciso ALTO RENDIMIENTO, aquel que no exceda de 5 segundos, para un equipo mainframe. Por lo tanto, si el tiempo de respuesta es menor a 5, entonces $\mu_{R1} = 1$. Si el tiempo de respuesta es mayor o igual a 5, entonces μ_{R1} estará dado por la función que expresamos anteriormente, donde **limite** = 5 (el valor limite

aceptable), y solo nos restaria definir el valor del divisor. Experimentalmente, evaluamos la expresión para diferentes valores del divisor, y del parámetro 1, y obtuvimos los resultados siguientes:

	divisor = 3	divisor = 2	divisor = 1
Parametro 1 =7	0.6	0.5	0.33
=9	0.42	0.33	0.20
=11	0.33	0.25	0.14
=15	0.23	0.16	0.09
=20	0.16	0.12	
=25	0.13	0.09	

Los valores que más se ajustan a la realidad, son para divisor=3, porque admite una mayor tolerancia en lo que podríamos considerar un "TIEMPO DE RESPUESTA ACEPTABLE".

Procedemos en forma similar para obtener cada una de las funciones que definen a los conjuntos imprecisos que sirven como dominio para cada uno de los atributos de la base de datos.

Nuevamente recalcamos el hecho de que no necesariamente la función debería tener esa estructura. Dependiendo del problema que se trate, podemos manejar diferentes funciones que se ajusten mejor a la información que queremos representar a través de la Base de Datos Imprecisa.

Las funciones que definen el grado de pertenencia de cada

parámetro, se encontraron en forma similar, y son las que aparecen a continuación:

PORCENTAJE DE PACKS ENCOLADOS (P2).

$$\mu = 1 / (1 + |P2 - 0.6| / 0.04) \quad \text{para } P2 \geq 0.6$$
$$\mu = 1 \quad \text{para } P2 < 0.6$$

NUMERO DE REGISTROS ACCESADOS (P3).

```
TAMREG = Tamaño del registro
P3 = 4000 div TAMREG
LIMINF = P3 - 5;
LIMSUP = P3 + 5;
```

$$\mu = 1 \quad \text{para LIMINF} < P3 < \text{LIMSUP}$$
$$\mu = 1 / (1 + |P3 - \text{LIMINF}| / (\text{LIMINF} / 2)) \quad \text{para } P3 \leq \text{LIMINF}$$
$$\mu = 1 / (1 + |P3 - \text{LIMSUP}| / (\text{LIMSUP} / 2)) \quad \text{para } P3 \geq \text{LIMSUP}$$

PORCENTAJE DE BUFFERS EN MEMORIA TOTAL(P4)

$$\mu = 1 / (1 + | P4 - 0.4 | / 0.05 \quad \text{para } P4 \geq 0.4$$
$$\mu = 1 \quad \text{para } P4 < 0.4$$

INTERVALOS DE ESPERA PAPA SYNCPOINT / CONTROLPOINT (P5)

$$\mu = 1 / (1 + |P5 - 1| / 0.1) \quad \text{para } P5 \leq 1$$

$\mu = 1$ para P5 > 1

NUMERO DE CONJUNTOS DE EXTENSION (P6)

$\mu = 1 / (1 + | P6 - 3 | / 1)$ para P6 ≥ 3

$\mu = 1$ para P6 < 3

TAMAÑO DEL BLOQUE PROMEDIO (P7)

$\mu = 1 / (1 + |P7 - 4000| / 2000)$ para P7 ≤ 4000

$\mu = 1$ para 4000 < P7 < 6000

$\mu = 1 / (1 + |P7 - 6000| / 1500)$ para P7 ≥ 6000

TAMAÑO MAXIMO DEL BLOQUE (P8)

$\mu = 1 / (1 + | P8 - 28 | / 20)$ para P8 ≤ 28

$\mu = 1$ para 28 < P8 < 32

$\mu = 1 / (1 + | P8 - 32 | / 8)$ para P8 ≥ 32

En una tabla, que en el sistema definimos como DBINCOM, se almacenan los valores de μ_{Pi} para cada atributo, y a continuación se procede a calcular el valor de pertenencia μ de la tupla, al conjunto impreciso: **"EL SISTEMA TIENE ALTO RENDIMIENTO PARA EL PARAMETRO 1, ALTO RENDIMIENTO PARA EL PARAMETRO 2, ... ALTO RENDIMIENTO PARA EL PARAMETRO 8"**. es decir, **"EL SISTEMA TIENE ALTO RENDIMIENTO PARA TODOS SUS PARAMETROS"**.

El valor μ de la tupla estará dado por el valor de μ_{Pi} mínimo observado entre todos los valores de μ_{Pi} por atributo. Esto es:

$$\mu_{\text{TUPLA}}(PAR.1\ PAR.2\ \ldots\ PAR.6) = MIN\ \{\ \mu_{P1}(PAR.1),\ \mu_{P2}(PAR.2),\ \ldots\ ,\ \mu_{P6}(PAR.6)\ \}$$

Este valor μ mínimo se almacenará en la columna 9 de la tabla DBINCOM.

Si manejamos en nuestro sistema exclusivamente el concepto del **grado de pertenencia** al conjunto impreciso "ALTO RENDIMIENTO EN TODOS LOS PARAMETROS", observamos que no siempre resulta ser muy representativo del nivel de performance de nuestro Sistema de Base de Datos. Esto se debe a que no todos los parámetros tienen el mismo peso, o dicho de otro modo, no todos los parámetros tienen la misma influencia sobre el performance del sistema.

Tenemos por ejemplo parámetros como el tiempo de respuesta, que resulta ser representativo por excelencia del performance del sistema, mientras que otros parámetros, como por ejemplo el numero de conjuntos de extension , si afectan el performance, pero en un grado mucho menor.

De esta forma, cuando al calcular los grados de pertenencia de cada uno de los parámetros, observamos que un parámetro con poco peso presenta un grado de pertenencia μ muy pequeño, si éste valor es el mínimo de la tupla ,por consiguiente será el grado de pertenencia μ representativo de la misma. Para el DBA este grado de pertenencia puede no resultar del todo veraz, ya que puede ser que para otros parámetros de peso mayor, μ posea niveles mucho mas altos.

Es por esto que definimos la conveniencia de calcular el

promedio ponderado de los grados de pertenencia, para obtener un valor que sirva de guia al DBA, indicándole que aunque μ TUPLA $\neq$ 1.0, la tupla puede tener un grado de pertenencia ponderado que indique que el sistema presenta un performance aceptable. Más aún, si el parámetro con μ mínimo resulta ser "menos pesado" y además resultara complejo modificarlo, podríamos encontrar que el sistema presenta un **rendimiento aceptable** aunque la tupla no pertenezca al conjunto impreciso "ALTO RENDIMIENTO PARATODOS LOS PARAMETROS".

Para el ejemplo que venimos manejando, asignamos "pesos" a los parámetros, en función de su influencia en el performance del Sistema de Base de Datos. Cabe señalar que esto se basaría en la experiencia del encargado de administrar la Base de Datos (DBA).

Los "pesos" que asignamos fueron:

PARAMETRO	PESO
1.- Tiempo de respuesta	0.70
2.- Porcentaje de packs encolados	0.10
3.- Número de registros accesados	0.05
4.- Porcentaje de buffers en memoria total	0.10
5.- Intervalos de espera para SYNC/CONTROLPOINT	0.01
6.- Número de conjuntos de extensión	0.01
7.- Tamaño del bloque promedio	0.02
8. Tamaño máximo del bloque	0.01

Como podemos observar, la suma de los pesos es igual a la unidad. El promedio ponderado se obtiene con la siguiente expresión:

$$\text{Promedio ponderado} = \sum_{i=1}^{n} (\mu_i \ \text{PESO}_i)$$

donde n = número de atributos manejados; en este caso n = 8.

Con todas las ideas que expusimos anteriormente, solo nos restaría pasar a la implementación del sistema. Como ya dijimos, utilizaremos PASCAL como lenguaje de programación. A continuación incluimos el listado documentado del programa del sistema.

```
  TYPE D AFINA.PAS
PROGRAM AFINACION(INPUT,OUTPUT);
CONST
     NO_TUPLAS=15;
     PESO1=0.7;         { CONSTANTES QUE REPRESENTAN LOS "PESOS" DE LOS }
     PESO2=0.1;         { PARAMETROS DE AFINACION MANEJADOS }
     PESO3=0.05;
     PESO4=0.1;
     PESO5=0.01;
     PESO6=0.01;
     PESO7=0.02;
     PESO8=0.01;
TYPE
     RENGLONES=1..NO_TUPLAS;
     COLUMNAS=1..20;
     BASEDATOS=ARRAY [RENGLONES,COLUMNAS] OF REAL;
     VECTOR=ARRAY[1..20] OF REAL;

VAR
    PAR1,PAR2,PAR3,PAR4,PAR5,PAR6,PAR7,PAR8:REAL;
    MINIMO, PROMEDIO, ACEPTABLE:REAL;
    TAMREG,TUPLA:INTEGER;
    PRINCIPAL:BASEDATOS;
    CONTINUA:CHAR;
    PONDERADO:VECTOR;

PROCEDURE LECTURA;     (ESTE PROCEDIMIENTO SE ENCARGA DE RECOPILAR
                        LOS VALORES DE LOS PARAMETROS DE AFINACION
                        ENCONTRADOS EN EL SISTEMA DE BASE DE DATOS
                        EN OPERACION. )
BEGIN
      WRITELN('*****************************************************');
      WRITELN('*                                                   *');
      WRITELN('*   PROGRAMA   DE   AFINACION   DE   BASES   DE   DATOS *');
      WRITELN('*                                                   *');
      WRITELN('*          UTILIZANDO   BASES  DE  DATOS  IMPRECISAS  *');
      WRITELN('*                                                   *');
      WRITELN('*****************************************************');
      WRITELN;
      WRITELN;
      WRITELN('      ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES:');
      WRITELN;
      WRITELN;
      WRITE(' PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --->  ');
      READLN(PAR1);
      WRITELN;
      WRITE(' PARAMETRO 2 : PORCENTAJE DE PACKS ENCOLADOS      --->  ');
      READLN(PAR2);
      WRITELN;
      WRITE(' PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS      --->  ');
      READLN(PAR3);
      WRITELN;
      WRITE(' TAMANO DE LOS REGISTROS MANEJADOS  EN BYTES      --->  ');
      READLN(TAMREG);
      WRITELN;
      WRITE(' PARAMETRO 4 : PORCENTAJE DE BUFFERS EN MEMORIA TOTAL --->  ');
      READLN(PAR4);
      WRITELN;
      WRITE(' PARAMETRO 5 : INTERVALOS DE ESPERA SYNC/CONTROLPOINT ');
      WRITE(' -HORAS- --->  ');
      READLN(PAR5);
      WRITELN;
      WRITE(' PARAMETRO 6 : NUMERO DE CONJUNTOS DE EXTENSION  --->  ');
      READLN(PAR6);
      WRITELN;
```

```
      WRITE(' PARAMETRO 7 : TAMANO DEL BLOQUE PROMEDIO  EN KBYTES-  - > ');
      READLN(PAR7);
      WRITELN;
      WRITE(' PARAMETRO 8 : TAMANO MAXIMO DEL BLOQUE  EN KBYTES-    --> ');
      READLN(PAR8);
      WRITELN;
END;  {PROCEDURE LECTURA}


PROCEDURE PONDERA(MU:REAL; COL:INTEGER);  { EN ESTE PROCEDIMIENTO CALCULAMOS
                                            EL PROMEDIO PONDERADO DE LOS
                                            GRADOS DE PERTENENCIA DE CADA UNO
                                            DE LOS ATRIBUTOS DE LA TUPLA }
VAR
     X: REAL;
BEGIN
     CASE COL OF
      1:X:=MU*PESO1;
      2:X:=MU*PESO2;
      3:X:=MU*PESO3;
      4:X:=MU*PESO4;
      5:X:=MU*PESO5;
      6:X:=MU*PESO6;
      7:X:=MU*PESO7;
      8:X:=MU*PESO8;
     END; {DEL CASE}
      PROMEDIO:=PROMEDIO + X;
     IF COL=8 THEN
        PONDERADO[TUPLA]:=PROMEDIO;

END;  {DEL PROCEDURE PONDERA }


PROCEDURE LLENADB(PERTENENCIA:REAL;
                  ATRIBUTO:INTEGER);  { EN ESTE PROCEDIMIENTO SE VA LLENANDO
                                        LA BASE DE DATOS INCOMPLETA CON LOS
                                        GRADOS DE PERTENENCIA DEL CONJUNTO
                                        IMPRECISO "ALTO RENDIMIENTO", CORRES-
                                        PONDIENTES A CADA PARAMETRO DE AFINA-
                                        CION, TAMBIEN SE ENCUENTRA EL GRADO
                                        DE PERTENENCIA MINIMO POR TUPLA, PARA
                                        OBTENER ASI EL GRADO DE PERTENENCIA
                                        DE LA TUPLA, AL CONJUNTO "ALTO RENDI-
                                        MIENTO".  }

BEGIN
     DBINCOM[TUPLA,ATRIBUTO]:=PERTENENCIA;
     IF PERTENENCIA < MINIMO THEN
        MINIMO:=PERTENENCIA;
END;  {PROCEDURE LLENADB}


PROCEDURE CALCULA(P1,P2,P3,P4,P5,P6,P7,P8:REAL;
                  TR:INTEGER);     { ESTE PROCEDIMIENTO SE ENCARGA DE EVALUAR
                                     EL GRADO DE PERTENENCIA AL CONJUNTO IM-
                                     PRECISO "ALTO RENDIMIENTO", PARA CADA UNO
                                     DE LOS PARAMETROS DE AFINACION CONSIDERA-
                                     DOS.  }

VAR
    P,LIMINF,LIMSUP,COLUMNA:INTEGER;
    M:REAL;
BEGIN
     PROMEDIO:=0;
```

```
COLUMNA:=1;
IF P1 <= 5 THEN
   M:=1/(1+ABS(P1-5)/3)
ELSE
   M:=1;
PONDERA(M,COLUMNA);
LLENADB(M,COLUMNA);

COLUMNA:=2;
IF P2 <= 0.6 THEN
   M:=1/(1+ABS(P2-0.6)/0.04)
ELSE
   M:=1;
PONDERA(M,COLUMNA);
LLENADB(M,COLUMNA);

COLUMNA:=3;
P:=P99 DIV TAMRES;
LIMINF:=P*5;
LIMSUP:=P+5;
IF P3 <= LIMINF THEN
   M:=1/(1+ABS(P3-LIMINF)/(LIMINF/2))
ELSE
   IF P3 >= LIMSUP THEN
      M:=1/(1+ABS(P3-LIMSUP)/(LIMSUP/2))
   ELSE
      M:=1;
PONDERA(M,COLUMNA);
LLENADB(M,COLUMNA);

COLUMNA:=4;
IF P4 <= 0.4 THEN
   M:=1/(1+ABS(P4-0.4)/0.05)
ELSE
   M:=1;
PONDERA(M,COLUMNA);
LLENADB(M,COLUMNA);

COLUMNA:=5;
IF P5 <= 1 THEN
   M:=1/(1+ABS(P5-1)/0.1)
ELSE
   M:=1;
PONDERA(M,COLUMNA);
LLENADB(M,COLUMNA);

COLUMNA:=6;
IF P6 <= 3 THEN
   M:=1/(1+ABS(P6-3))
ELSE
   M:=1;
PONDERA(M,COLUMNA);
LLENADB(M,COLUMNA);

COLUMNA:=7;
IF P7 <= 4 THEN
   M:=1/(1+ABS(P7-4)/2)
ELSE
   IF P7 <= 6 THEN
      M:=1/(1+ABS(P7-6)/.5)
   ELSE
      M:=1;
PONDERA(M,COLUMNA);
LLENADB(M,COLUMNA);

COLUMNA:=8;
```

```
      IF P8 <= 20 THEN
          M:=1/(1+ABS(P8-20)/20)
      ELSE
          IF P8 >= 32 THEN
               M:=1/(1+ABS(P8-32)/8)
          ELSE
               M:=1;
      ENCUENTRAM(COLUMNA);
      LLENADR(M,COLUMNA);

END; {PROCEDURE CALCULA}

PROCEDURE MU_TUPLA;     { EN ESTE PROCEDIMIENTO SE ASIGNA  EL GRADO DE
                          PERTENENCIA DE LA TUPLA, A LA NOVENA COLUMNA
                          DE LA BASE DE DATOS INCOMPLETA DBINCOM. }
BEGIN
      DBINCOM[TUPLA,9]:=MINIMO;
END;  {PROCEDURE MU_TUPLA}


PROCEDURE TABLA;   { EN ESTE PROCEDIMIENTO DESPLEGAMOS EL GRADO DE
                     PERTENENCIA DE CADA UNO DE LOS PARAMETROS, PARA
                     LAS DIFERENTES EVALUACIONES QUE SE VAN HACIENDO
                     DEL MISMO }
VAR
      I,J:INTEGER;
BEGIN
      WRITELN;
      WRITE('              ');
      WRITELN('** GRADOS DE PERTENENCIA DE CADA UNO DE LOS PARAMETROS **');
      WRITE('              ');
      WRITELN('**   EN  LAS  DIFERENTES  EVALUACIONES  DEL  SISTEMA   **');
      WRITELN;
      WRITELN;
      WRITE('                PAR.1  PAR.2  PAR.3  PAR.4  PAR.5  PAR.6  PAR.7');
      WRITELN('  PAR.8    M ');
      WRITE('--------------------------------------------------------------');
      WRITELN('--------------');
      FOR I:=1 TO TUPLA DO
      BEGIN
      WRITE('EVALUACION  ',I);
      FOR J:=1 TO 9 DO
         WRITE(DBINCOM[I,J]:6:2,' ');
      WRITELN;
      WRITE('--------------------------------------------------------------');
      WRITELN('--------------');
      END;
      WRITELN;
      WRITELN;
      WRITELN;
      WRITELN('                     m         PROMEDIO PONDERADO ');
      WRITELN('----------------------------------------------');
      FOR I:=1 TO TUPLA DO
           BEGIN
           WRITE('EVALUACION  ', I);
           WRITE('   ',DBINCOM[I,9]:6:2);
           WRITELN('         ',PONDERADO[I]:6:2);
           WRITELN('-----------------------------------------------');
           END;
END;

PROCEDURE IMPRIME;      { PROCEDIMIENTO QUE IMPRIME LOS GRADOS DE PERTE-
                          NENCIA CALCULADOS POR PARAMETRO, ASI  COMO  EL
```

```
                        GRADO DE PERTENENCIA DE LA TUPLA, AL CONJUNTO
                        IMPRECISO "ALTO RENDIMIENTO" )
VAR
    I:INTEGER;
BEGIN
    WRITELN;
    WRITELN('   GRADOS   DE   PERTENENCIA ');
    WRITELN;
    WRITELN('*********************************');
    FOR I:=1 TO 8 DO
        WRITELN('      PARAMETRO ',I,' = ',DBINCOMITUPLA,I]:6:2,'      ');
    WRITELN('*********************************');
    WRITELN;
    WRITELN;
    WRITELN('EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO');
    WRITELN('" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL  A');
    WRITELN('                ',DBINCOMITUPLA,0]:6:2);
    WRITELN;
    WRITELN;

    WRITELN('*****************************************');
    WRITELN('  EL PROMEDIO PONDERADO DE LOS GRADOS DE ');
    WRITELN('  DE PERTENENCIA ES IGUAL A ',PONDERADO[TUPLA]:6:2);
    WRITELN('*****************************************');



    IF DBINCOMITUPLA,9]=1.0 THEN
                        ( CUANDO EL SISTEMA MUESTRA UN ALTO RENDIMIENTO )
       BEGIN
       WRITELN;
       WRITELN('*************************************************');
       WRITELN('******* EL SISTEMA PRESENTA UN ALTO RENDIMIENTO ********');
       WRITELN('******* EN TODOS LOS PARAMETROS.  NO HAY NECESIDAD ********');
       WRITELN('******* DE MODIFICARLOS.                           ********');
       WRITELN('*************************************************');
       WRITELN;
       TABLA;
       CONTINUA:='N';
       END
    ELSE
      BEGIN
        IF PONDERADO[TUPLA] > ACEPTABLE THEN
           BEGIN
           WRITELN;
           WRITELN('*************************************************');
           WRITELN('** EL PROMEDIO PONDERADO DE LOS GRADOS DE PERTENENCIA **');
           WRITELN('** SE CONSIDERA DENTRO  DE  UN  RANGO  ACEPTABLE  POR **');
           WRITELN('** LO TANTO, NO HAY NECESIDAD DE MODIFICAR PARAMETROS **');
           WRITELN('*************************************************');
           WRITELN;
           WRITELN;
           TABLA;
           CONTINUA:='N';
           END
        ELSE
        BEGIN
            WRITELN('QUIERES CONTINUAR? S/N');
            READLN(CONTINUA);
        END;
      END;
END; (PROCEDURE IMPRIME)

PROCEDURE LEE_CAMBIO;  ( EN ESTE PROCEDIMIENTO SE LEEN LOS NUEVOS VALORES
                         DE LOS PARAMETROS QUE SE ACONSEJO MODIFICAR. )
```

```
VAR
   I INTEGER;
BEGIN
     WRITELN;
     WRITELN;
     WRITELN('*****************************************************');
     WRITELN('*                                                   *');
     WRITELN('*     ADECTA LOS VALORES DE LOS PARAMETROS SIGUIENTES     *');
     WRITELN('*                                                   *');
     WRITELN('*****************************************************');
     WRITELN;
     WRITELN;
       FOR I:=1 TO 8 DO
       BEGIN
        IF DRINCOM[TUPLA,I] < 1 THEN
        BEGIN
          CASE I OF
            1:BEGIN
              WRITE('PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --> ');
              READLN(PAR1);
              WRITELN;
              END;
            2:BEGIN
              WRITE('PARAMETRO 2 : PORCENTAJE DE PAGES ENCOLADOS  --> ');
              READLN(PAR2);
              WRITELN;
              END;
            3:BEGIN
              WRITE('PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS  --> ');
              READLN(PAR3);
              WRITELN;
              WRITE('INDICA EL TAMANO DE LOS REGISTROS MANEJADOS');
              WRITE(' EN BYTES- --> ');
              READLN(TAMREG);
              WRITELN;
              END;
            4:BEGIN
              WRITE('PARAMETRO 4 : PORCENTAJE DE BUFFERS EN MEMORIA TOTAL --> ');
              READLN(PAR4);
              WRITELN;
              END;
            5:BEGIN
              WRITE('PARAMETRO 5 : INTERVALOS DE ESPERA SYNC/CONTROLPOINT');
              WRITE(' -HORAS- --> ');
              READLN(PAR5);
              WRITELN;
              END;
            6:BEGIN
              WRITE('PARAMETRO 6 : NUMERO DE CONJUNTOS DE EXTENSION --> ');
              READLN(PAR6);
              WRITELN;
              END;
            7:BEGIN
              WRITE('PARAMETRO 7 : TAMANO DEL BLOQUE PROMEDIO -EN KBYTES- --> ');
              READLN(PAR7);
              WRITELN;
              END;
            8:BEGIN
              WRITE('PARAMETRO 8 : TAMANO MAXIMO DEL BLOQUE -EN KBYTES  --> ');
              READLN(PAR8);
              WRITELN;
              END;
        END; (DEL CASE)
      END;
     END;
     TUPLA:=TUPLA+1;          { AVANZAMOS AL SIGUIENTE RENGLON DE LA BASE
```

```
                                        DE DATOS INCOMPLETA)
      MINIMO:=1;
      CALCULA(PAR1,PAR2,PAR3,PAR4,PAR5,PAR6,PAR7,PAR8,TAMREG);
END; (PROCEDURE LEE_CAMBIO)


PROCEDURE MODIFICA;
VAR
      I:INTEGER;
BEGIN
      IF DBINCOM[TUPLA,9] = 1 THEN
         BEGIN
              WRITELN('*************************************************');
              WRITELN('** LA BASE DE DATOS PRESENTA UN ALTO RENDIMIENTO **');
              WRITELN('** EN TODOS LOS PARAMETROS DE AFINACION. POR LO **');
              WRITELN('** TANTO NO ES NECESARIO MODIFICAR NINGUN PARAMETRO **');
              WRITELN('*************************************************');
              WRITELN;
              TABLA;
              CONTINUA:='N';
         END
      ELSE
       IF PONDERADO[TUPLA] < ACEPTABLE THEN
        BEGIN
        WRITELN;
        WRITELN('*************************************************');
        WRITELN('*                                               *');
        WRITELN('* RESULTA CONVENIENTE MODIFICAR LOS PARAMETROS SIGUIENTES: *');
        WRITELN('*                                               *');
        WRITELN('*************************************************');
        WRITELN;
        WRITELN;
        FOR I:=1 TO 8 DO
           BEGIN
             IF DBINCOM[TUPLA,I] <1 THEN
             CASE I OF
               1: WRITELN(' -   TIEMPO DE RESPUESTA  ');
               2: WRITELN(' -   PORCENTAJE DE PACKS ENCOLADOS  ');
               3: WRITELN(' -   NUMERO DE REGISTROS ACCESADOS  ');
               4: WRITELN(' -   PORCENTAJE DE BUFFERS DE MEMORIA TOTAL ');
               5: WRITELN(' -   INTERVALOS DE ESPERA PARA SYNC/CONTROLPOINT ');
               6: WRITELN(' -   NUMERO DE CONJUNTOS DE EXTENSION ');
               7: WRITELN(' -   TAMANO DEL BLOQUE PROMEDIO ');
               8: WRITELN(' -   TAMANO MAXIMO DEL BLOQUE ');
             END;
           END;
        LEE_CAMBIO;
       END;
END;  (PROCEDURE MODIFICA)


(********** P R O G R A M A    P R I N C I P A L **********)
BEGIN
      WRITELN;
      WRITELN('ANOTA EL PORCENTAJE DE RENDIMIENTO DEL SISTEMA');
      WRITELN('DE BASE DE DATOS QUE CONSIDERAS ACEPTABLE');
      WRITELN('- PARA EL PROMEDIO PONDERADO DE LOS GRADOS DE');
      WRITELN('PERTENENCIA -');
      READLN(ACEPTABLE);
      LECTURA;
      TUPLA:=1;
      MINIMO:=1;
      CALCULA(PAR1,PAR2,PAR3,PAR4,PAR5,PAR6,PAR7,PAR8,TAMREG);
      MU_TUPLA;
      IMPRIME;
```

```
    REPEAT
        IF CONTINUA='S' THEN
            BEGIN
               MODIFICA;
               MU_TUPLA;
               IMPRIME;
            END
         ELSE
            CONTINUA:='N';
     UNTIL CONTINUA='N';

END.
```

A>

A continuación mostramos algunas ejecuciones del Sistema de Afinación de Bases de Datos. Como podrá observarse, el DBA deberá introducir el porcentaje de performance considerado "aceptable" para el sistema que se está analizando. Este porcentaje se refiere al promedio ponderado de los grados de pertenencia de cada uno de los parámetros de afinación manejados.

Después se introducen los valores de los parámetros de afinación, obtenidos ya sea por monitoreo o a través de estadísticas arrojadas por el sistema. Con esta información, nuestro sistema calcula los grados de pertenencia de cada parámetro, y despliega el promedio ponderado de éstos. Si el promedio está dentro del rango considerado "ACEPTABLE", la ejecución del programa termina. Si no, aparecen los parámetros que se considera deben ser modificados para elevar el nivel de performance del sistema. El DBA introduce los nuevos valores de los parámetros, y el procedimiento se repite hasta que todos los parámetros de afinación presenten "ALTO RENDIMIENTO", o bien cuando el promedio ponderado se considera aceptable y se indica que el Sistema de Base de Datos está funcionando con un performance adecuado.

Por último, graficamos para cada ejecución, el promedio ponderado de los grados de pertenencia, y el grado de pertenencia de cada una de las evaluaciones del sistema.

ANOTA EL PORCENTAJE DE RENDIMIENTO DEL SISTEMA
DE BASE DE DATOS QUE CONSIDERAS ACEPTABLE
- PARA EL PROMEDIO PONDERADO DE LOS GRADOS DE
PERTENENCIA -
0.8

```
**********************************************************
*                                                        *
*   PROGRAMA   DE   AFINACION   DE   BASES   DE   DATOS  *
*                                                        *
*         UTILIZANDO   BASES  DE  DATOS  IMPRECISAS      *
*                                                        *
**********************************************************
```

ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES:

PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --> 6

PARAMETRO 2 : PORCENTAJE DE PACKS ENCOLADOS --> 0.5

PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS --> 50

TAMANO DE LOS REGISTROS MANEJADOS -EN BYTES- --> 300

PARAMETRO 4 : PORCENTAJE DE BUFFERS EN MEMORIA TOTAL --> 0.4

PARAMETRO 5 : INTERVALOS DE ESPERA SYNC/CONTROLPOINT -HORAS- --> 0.5

PARAMETRO 6 : NUMERO DE CONJUNTOS DE EXTENSION --> 3

PARAMETRO 7 : TAMANO DEL BLOQUE PROMEDIO -EN KBYTES- --> 6

PARAMETRO 8 : TAMANO MAXIMO DEL BLOQUE -EN KBYTES- --> 33

GRADOS DE PERTENENCIA

```
**************************
*  PARAMETRO 1 =  0.75   *
*  PARAMETRO 2 =  1.00   *
*  PARAMETRO 3 =  0.22   *
*  PARAMETRO 4 =  1.00   *
*  PARAMETRO 5 =  0.17   *
*  PARAMETRO 6 =  1.00   *
*  PARAMETRO 7 =  1.00   *
*  PARAMETRO 8 =  0.89   *
**************************
```

EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL A
0.17

****

```
  EL PROMEDIO PONDERADO DE LOS GRADOS DE
  DE PERTENENCIA ES IGUAL A    0.79
*****************************************
QUIERES CONTINUAR? S/N
S

*********************************************************
*                                                       *
* RESULTA CONVENIENTE MODIFICAR LOS PARAMETROS SIGUIENTES: *
*                                                       *
*********************************************************


  -    TIEMPO DE RESPUESTA
  -    NUMERO DE REGISTROS ACCESADOS
  -    INTERVALOS DE ESPERA PARA SYNC/CONTROLPOINT
       TAMANO MAXIMO DEL BLOQUE


*********************************************************
*                                                       *
*     ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES    *
*                                                       *
*********************************************************


PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --> 6

PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS  --> 30

INDICA EL TAMANO DE LOS REGISTROS MANEJADOS-EN BYTES- --> 300

PARAMETRO 5 : INTERVALOS DE ESPERA SYNC/CONTROLPOINT -HORAS- --> 0.75

PARAMETRO 8 : TAMANO MAXIMO DEL BLOQUE -EN KBYTES- --> 30

   GRADOS   DE   PERTENENCIA

*********************************
*     PARAMETRO 1 =   0.75      *
*     PARAMETRO 2 =   1.00      *
*     PARAMETRO 3 =   0.43      *
*     PARAMETRO 4 =   1.00      *
*     PARAMETRO 5 =   0.29      *
*     PARAMETRO 6 =   1.00      *
*     PARAMETRO 7 =   1.00      *
*     PARAMETRO 8 =   1.00      *
*********************************


EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL  A
                  0.29


*****************************************
  EL PROMEDIO PONDERADO DE LOS GRADOS DE
  DE PERTENENCIA ES IGUAL A    0.79
*****************************************
QUIERES CONTINUAR? S/N
S

*********************************************************
*                                                       *
* RESULTA CONVENIENTE MODIFICAR LOS PARAMETROS SIGUIENTES: *
*
```

****

- TIEMPO DE RESPUESTA
- NUMERO DE REGISTROS ACCESADOS
- INTERVALOS DE ESPERA PARA SYNC/CONTROLPOINT

****
* *
* ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES *
* *
****

PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --› 6

PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS --› 20

INDICA EL TAMANO DE LOS REGISTROS MANEJADOS-EN BYTES- --› 300

PARAMETRO 5 : INTERVALOS DE ESPERA SYNC/CONTROLPOINT -HORAS- --› 0.75

GRADOS DE PERTENENCIA

****
* PARAMETRO 1 = 0.75 *
* PARAMETRO 2 = 1.00 *
* PARAMETRO 3 = 0.82 *
* PARAMETRO 4 = 1.00 *
* PARAMETRO 5 = 0.29 *
* PARAMETRO 6 = 1.00 *
* PARAMETRO 7 = 1.00 *
* PARAMETRO 8 = 1.00 *
****

EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL A
0.29

****
EL PROMEDIO PONDERADO DE LOS GRADOS DE
DE PERTENENCIA ES IGUAL A 0.81
****

****
* EL PROMEDIO PONDERADO DE LOS GRADOS DE PERTENENCIA *
* SE CONSIDERA DENTRO DE UN RANGO ACEPTABLE. POR *
* LO TANTO, NO HAY NECESIDAD DE MODIFICAR PARAMETROS *
****

** GRADOS DE PERTENENCIA DE CADA UNO DE LOS PARAMETROS **
** EN LAS DIFERENTES EVALUACIONES DEL SISTEMA **

	PAR.1	PAR.2	PAR.3	PAR.4	PAR.5	PAR.6	PAR.7	PAR.8	M
EVALUACION 1	0.75	1.00	0.22	1.00	0.17	1.00	1.00	0.89	0.17
EVALUACION 2	0.75	1.00	0.43	1.00	0.29	1.00	1.00	1.00	0.29

EVALUACION	3	0.75	1.00	0.82	1.00	0.29	1.00	1.00	1.00	0.29

		m	PROMEDIO PONDERADO
EVALUACION	1	0.17	0.78
EVALUACION	2	0.29	0.79
EVALUACION	3	0.29	0.81

EJECUCION NO. 1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Porcentaje de rendimiento
1
2
3
Evaluaciones del sistema
m
promedio ponderado

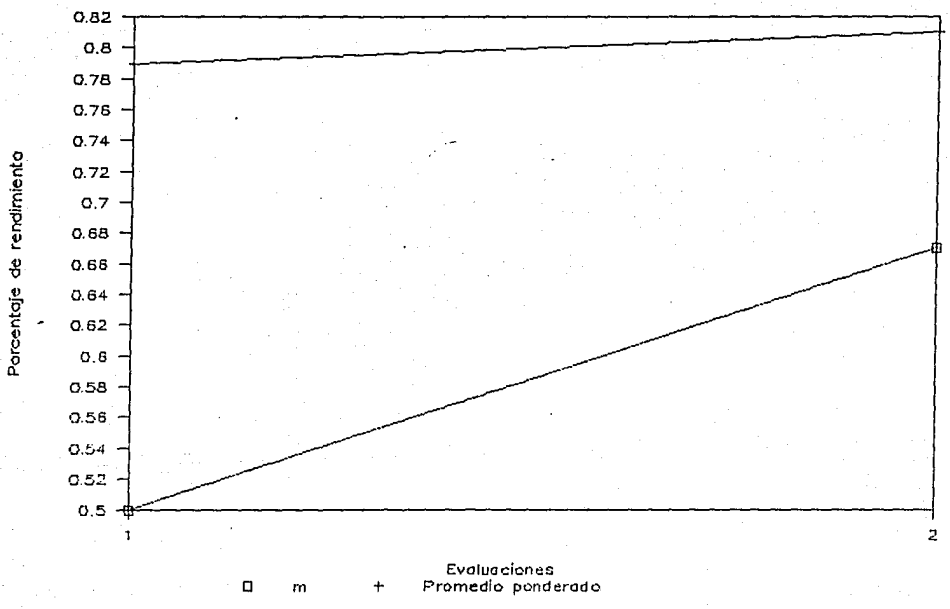
```
ANOTA EL PORCENTAJE DE RENDIMIENTO DEL SISTEMA
DE  BASE  DE  DATOS  QUE  CONSIDERAS ACEPTABLE
- PARA EL  PROMEDIO PONDERADO DE LOS GRADOS DE
PERTENENCIA -
0.8
**************************************************
*                                                *
*   PROGRAMA   DE   AFINACION   DE   BASES   DE   DATOS   *
*                                                *
*         UTILIZANDO   BASES  DE  DATOS  IMPRECISAS        *
*                                                *
**************************************************

     ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES:

 PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --> 6

 PARAMETRO 2 : PORCENTAJE DE PACKS ENCOLADOS     --> 0.55

 PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS     --> 30

 TAMANO DE LOS REGISTROS MANEJADOS -EN BYTES-    --> 256

 PARAMETRO 4 : PORCENTAJE DE BUFFERS EN MEMORIA TOTAL --> 0.4

 PARAMETRO 5 : INTERVALOS DE ESPERA SYNC/CONTROLPOINT  -HORAS- --> 1

 PARAMETRO 6 : NUMERO DE CONJUNTOS DE EXTENSION  --> 4

 PARAMETRO 7 : TAMANO DEL BLOQUE PROMEDIO -EN KBYTES-  --> 5

 PARAMETRO 8 : TAMANO MAXIMO DEL BLOQUE -EN KBYTES-    --> 32

   GRADOS   DE   PERTENENCIA

**********************************
*      PARAMETRO 1 =   0.75      *
*      PARAMETRO 2 =   1.00      *
*      PARAMETRO 3 =   0.50      *
*      PARAMETRO 4 =   1.00      *
*      PARAMETRO 5 =   1.00      *
*      PARAMETRO 6 =   0.50      *
*      PARAMETRO 7 =   1.00      *
*      PARAMETRO 8 =   1.00      *
**********************************


EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL  A
                  0.50

****************************************
  EL PROMEDIO PONDERADO DE LOS GRADOS DE
  DE PERTENENCIA ES IGUAL A    0.79
****************************************
```

```
QUIERES CONTINUAR? S/N
S

**********************************************************
*                                                        *
* RESULTA CONVENIENTE MODIFICAR LOS PARAMETROS SIGUIENTES: *
*                                                        *
**********************************************************


 -   TIEMPO DE RESPUESTA
 -   NUMERO DE REGISTROS ACCESADOS
 -   NUMERO DE CONJUNTOS DE EXTENSION


**********************************************************
*                                                        *
*    ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES      *
*                                                        *
**********************************************************


PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --> 6

PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS  --> 25

INDICA EL TAMANO DE LOS REGISTROS MANEJADOS-EN BYTES- --> 256

PARAMETRO 6 : NUMERO DE CONJUNTOS DE EXTENSION --> 3


   GRADOS   DE   PERTENENCIA

**********************************
*      PARAMETRO 1 =   0.75      *
*      PARAMETRO 2 =   1.00      *
*      PARAMETRO 3 =   0.67      *
*      PARAMETRO 4 =   1.00      *
*      PARAMETRO 5 =   1.00      *
*      PARAMETRO 6 =   1.00      *
*      PARAMETRO 7 =   1.00      *
*      PARAMETRO 8 =   1.00      *
**********************************


EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL  A
                  0.67


*****************************************
  EL PROMEDIO PONDERADO DE LOS GRADOS DE
  DE PERTENENCIA ES IGUAL A   0.81
*****************************************

*****************************************************
* EL PROMEDIO PONDERADO DE LOS GRADOS DE PERTENENCIA *
* SE CONSIDERA DENTRO  DE  UN  RANGO  ACEPTABLE. POR *
* LO TANTO, NO HAY NECESIDAD DE MODIFICAR PARAMETROS *
*****************************************************


        ** GRADOS DE PERTENENCIA DE CADA UNO DE LOS PARAMETROS **
        **   EN  LAS  DIFERENTES  EVALUACIONES  DEL  SISTEMA   **
```

		PAR.1	PAR.2	PAR.3	PAR.4	PAR.5	PAR.6	PAR.7	PAR.8	M
EVALUACION	1	0.75	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50
EVALUACION	2	0.75	1.00	0.67	1.00	1.00	1.00	1.00	1.00	0.67

		m	PROMEDIO PONDERADO
EVALUACION	1	0.50	0.79
EVALUACION	2	0.67	0.81

EJECUCION NO. 2



ANOTA EL PORCENTAJE DE RENDIMIENTO DEL SISTEMA
DE BASE DE DATOS QUE CONSIDERAS ACEPTABLE
- PARA EL PROMEDIO PONDERADO DE LOS GRADOS DE
PERTENENCIA -
0.. .85

```
**********************************************************
*                                                        *
*   PROGRAMA   DE   AFINACION   DE   BASES   DE   DATOS  *
*                                                        *
*         UTILIZANDO   BASES  DE  DATOS  IMPRECISAS      *
*                                                        *
**********************************************************
```

ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES:

PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --› 7

PARAMETRO 2 : PORCENTAJE DE PACKS ENCOLADOS --› 0.4

PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS --› 40

TAMANO DE LOS REGISTROS MANEJADOS -EN BYTES- --› 300

PARAMETRO 4 : PORCENTAJE DE BUFFERS EN MEMORIA TOTAL --› 0.5

PARAMETRO 5 : INTERVALOS DE ESPERA SYNC/CONTROLPOINT -HORAS- --› 4

PARAMETRO 6 : NUMERO DE CONJUNTOS DE EXTENSION --› 3

PARAMETRO 7 : TAMANO DEL BLOQUE PROMEDIO -EN KBYTES- --› 3

PARAMETRO 8 : TAMANO MAXIMO DEL BLOQUE -EN KBYTES- --› 30

GRADOS DE PERTENENCIA

```
**********************************
*     PARAMETRO 1 =   0.60       *
*     PARAMETRO 2 =   1.00       *
*     PARAMETRO 3 =   0.29       *
*     PARAMETRO 4 =   0.33       *
*     PARAMETRO 5 =   1.00       *
*     PARAMETRO 6 =   1.00       *
*     PARAMETRO 7 =   0.67       *
*     PARAMETRO 8 =   1.00       *
**********************************
```

EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL A
0.29

```
*****************************************
  EL PROMEDIO PONDERADO DE LOS GRADOS DE
  DE PERTENENCIA ES IGUAL A   0.61
*****************************************
```

```
QUIERES CONTINUAR? S/N
S

*************************************************************
*                                                           *
* RESULTA CONVENIENTE MODIFICAR LOS PARAMETROS SIGUIENTES:  *
*                                                           *
*************************************************************

  -  TIEMPO DE RESPUESTA
  -  NUMERO DE REGISTROS ACCESADOS
  -  PORCENTAJE DE BUFFERS DE MEMORIA TOTAL
  -  TAMANO DEL BLOQUE PROMEDIO

*************************************************
*                                               *
*    ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES   *
*                                               *
*************************************************

PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS-    6

PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS --> 10

INDICA EL TAMANO DE LOS REGISTROS MANEJADOS-EN BYTES- --> 300

PARAMETRO 4 : PORCENTAJE DE BUFFERS EN MEMORIA TOTAL --> 0.6

PARAMETRO 7 : TAMANO DEL BLOQUE PROMEDIO -EN KBYTES- --> 5

   GRADOS   DE   PERTENENCIA

*********************************
*     PARAMETRO 1 =   0.75      *
*     PARAMETRO 2 =   1.00      *
*     PARAMETRO 3 =   0.43      *
*     PARAMETRO 4 =   1.00      *
*     PARAMETRO 5 =   1.00      *
*     PARAMETRO 6 =   1.00      *
*     PARAMETRO 7 =   1.00      *
*     PARAMETRO 8 =   1.00      *
*********************************

EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL  A
                0.43

*****************************************
  EL PROMEDIO PONDERADO DE LOS GRADOS DE
  DE PERTENENCIA ES IGUAL A   0.80
*****************************************
QUIERES CONTINUAR? S/N
S

*************************************************************
*                                                           *
* RESULTA CONVENIENTE MODIFICAR LOS PARAMETROS SIGUIENTES:  *
*                                                           *
*************************************************************
```

```
-   TIEMPO DE RESPUESTA
    NUMERO DE REGISTROS ACCESADOS

*****************************************************
*                                                   *
*    ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES *
*                                                   *
*****************************************************

PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --> 6

PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS  --> 20

INDICA EL TAMANO DE LOS REGISTROS MANEJADOS EN BYTES-  --> 200

   GRADOS   DE   PERTENENCIA

*********************************
*     PARAMETRO 1 =   0.75      *
*     PARAMETRO 2 =   1.00      *
*     PARAMETRO 3 =   0.82      *
*     PARAMETRO 4 =   1.00      *
*     PARAMETRO 5 =   1.00      *
*     PARAMETRO 6 =   1.00      *
*     PARAMETRO 7 =   1.00      *
*     PARAMETRO 8 =   1.00      *
*********************************

EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL  A
                  0.75

*******************************************
  EL PROMEDIO PONDERADO DE LOS GRADOS DE
  DE PERTENENCIA ES IGUAL A    0.103
*******************************************
QUIERES CONTINUAR? S/N
S

*************************************************************
*                                                           *
* RESULTA CONVENIENTE MODIFICAR LOS PARAMETROS SIGUIENTES:  *
*                                                           *
*************************************************************


  -   TIEMPO DE RESPUESTA
      NUMERO DE REGISTROS ACCESADOS

*****************************************************
*                                                   *
*    ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES *
*                                                   *
*****************************************************

PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --> 6

PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS  --> 10
```

INDICA EL TAMAÑO DE LOS REGISTROS MANEJADOS EN BYTES: ---> 200

GRADOS DE PERTENENCIA

PARAMETRO 1 =	0.75
PARAMETRO 2 =	1.00
PARAMETRO 3 =	1.00
PARAMETRO 4 =	1.00
PARAMETRO 5 =	1.00
PARAMETRO 6 =	1.00
PARAMETRO 7 =	1.00
PARAMETRO 8 =	1.00

EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL A
0.75

EL PROMEDIO PONDERADO DE LOS GRADOS DE
DE PERTENENCIA ES IGUAL A 0.80

QUIERES CONTINUAR? S/N
S

RESULTA CONVENIENTE MODIFICAR LOS PARAMETROS SIGUIENTES:

- TIEMPO DE RESPUESTA

ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES

PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- ---> 5

GRADOS DE PERTENENCIA

PARAMETRO 1 =	1.00
PARAMETRO 2 =	1.00
PARAMETRO 3 =	1.00
PARAMETRO 4 =	1.00
PARAMETRO 5 =	1.00
PARAMETRO 6 =	1.00
PARAMETRO 7 =	1.00
PARAMETRO 8 =	1.00

EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL A
1.00

1.00

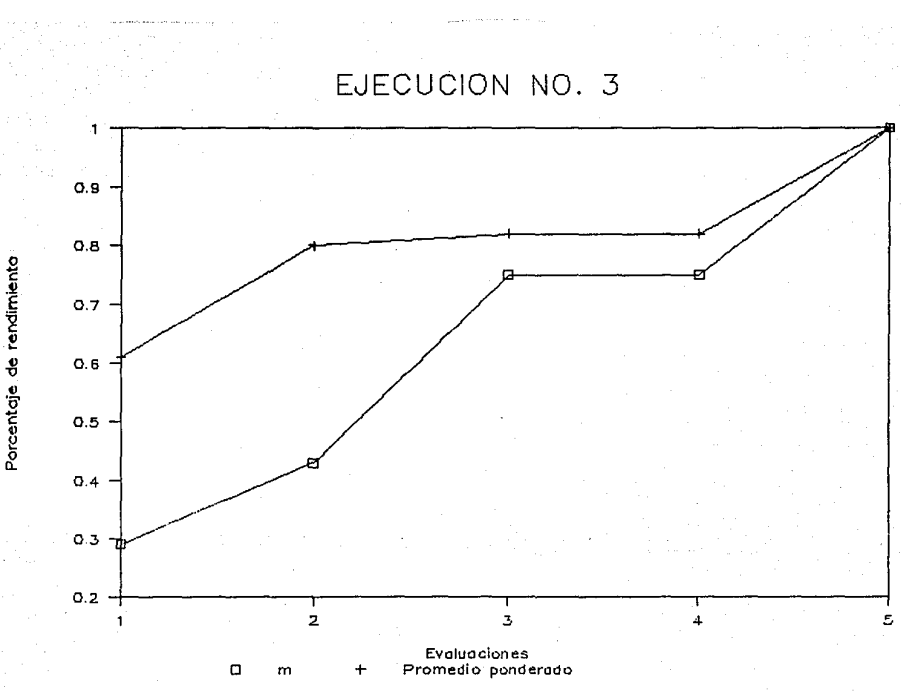
****
EL PROMEDIO PONDERADO DE LOS GRADOS DE
DE PERTENENCIA ES IGUAL A 1.00
****

****
****** EL SISTEMA PRESENTA UN ALTO RENDIMIENTO ********
****** EN TODOS LOS PARAMETROS. NO HAY NECESIDAD ********
****** DE MODIFICARLOS. ********
****

** GRADOS DE PERTENENCIA DE CADA UNO DE LOS PARAMETROS **
** EN LAS DIFERENTES EVALUACIONES DEL SISTEMA **

	PAR.1	PAR.2	PAR.3	PAR.4	PAR.5	PAR.6	PAR.7	PAR.8	M
EVALUACION 1	0.60	1.00	0.29	0.33	1.00	1.00	0.67	1.00	0.29
EVALUACION 2	0.75	1.00	0.43	1.00	1.00	1.00	1.00	1.00	0.43
EVALUACION 3	0.75	1.00	0.82	1.00	1.00	1.00	1.00	1.00	0.75
EVALUACION 4	0.75	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.75
EVALUACION 5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

	m	PROMEDIO PONDERADO
EVALUACION 1	0.29	0.61
EVALUACION 2	0.43	0.80
EVALUACION 3	0.75	0.82
EVALUACION 4	0.75	0.82
EVALUACION 5	1.00	1.00


EJECUCION NO. 3
Porcentaje de rendimiento
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
1
2
3
4
5
Evaluaciones
m
Promedio ponderado

```
ANOTA EL PORCENTAJE DE RENDIMIENTO DEL SISTEMA
DE BASE DE DATOS QUE CONSIDERAS ACEPTABLE
- PARA EL PROMEDIO PONDERADO DE LOS GRADOS DE
PERTENENCIA -
0.75
**********************************************************
*                                                        *
*   PROGRAMA DE AFINACION DE BASES DE DATOS              *
*                                                        *
*        UTILIZANDO BASES DE DATOS IMPRECISAS            *
*                                                        *
**********************************************************

     ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES:

 PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --> 7

 PARAMETRO 2 : PORCENTAJE DE PACKS ENCOLADOS     --> 0.5

 PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS     --> 30 5

 TAMANO DE LOS REGISTROS MANEJADOS -EN BYTES-    --> 256

 PARAMETRO 4 : PORCENTAJE DE BUFFERS EN MEMORIA TOTAL --> 0.4

 PARAMETRO 5 : INTERVALOS DE ESPERA SYNC/CONTROLPOINT -HORAS- --> 3

 PARAMETRO 6 : NUMERO DE CONJUNTOS DE EXTENSION  --> 4

 PARAMETRO 7 : TAMANO DEL BLOQUE PROMEDIO -EN KBYTES- --> 5

 PARAMETRO 8 : TAMANO MAXIMO DEL BLOQUE -EN KBYTES-  --> 28

   GRADOS DE PERTENENCIA

*********************************
*     PARAMETRO 1 =   0.60      *
*     PARAMETRO 2 =   1.00      *
*     PARAMETRO 3 =   0.40      *
*     PARAMETRO 4 =   1.00      *
*     PARAMETRO 5 =   1.00      *
*     PARAMETRO 6 =   0.50      *
*     PARAMETRO 7 =   1.00      *
*     PARAMETRO 8 =   1.00      *
*********************************


EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL A
                  0.40

****************************************
```

```
  EL PROMEDIO PONDERADO DE LOS GRADOS DE
  DE PERTENENCIA ES IGUAL A   0.68
*********************************************
QUIERES CONTINUAR? S/N
S

*************************************************************
*                                                           *
* RESULTA CONVENIENTE MODIFICAR LOS PARAMETROS SIGUIENTES:  *
*                                                           *
*************************************************************


 -    TIEMPO DE RESPUESTA
 -    NUMERO DE REGISTROS ACCESADOS
 -    NUMERO DE CONJUNTOS DE EXTENSION


*************************************************************
*                                                           *
*    ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES         *
*                                                           *
*************************************************************


PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS   --> 7

PARAMETRO 3 : NUMERO DE REGISTROS ACCESADOS  --> 20

INDICA EL TAMANO DE LOS REGISTROS MANEJADOS-EN BYTES- --> 256

PARAMETRO 6 : NUMERO DE CONJUNTOS DE EXTENSION --> 2


   GRADOS   DE   PERTENENCIA

*********************************
*      PARAMETRO 1 =   0.60     *
*      PARAMETRO 2 =   1.00     *
*      PARAMETRO 3 =   1.00     *
*      PARAMETRO 4 =   1.00     *
*      PARAMETRO 5 =   1.00     *
*      PARAMETRO 6 =   1.00     *
*      PARAMETRO 7 =   1.00     *
*      PARAMETRO 8 =   1.00     *
*********************************


EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO IMPRECISO
" ALTO RENDIMIENTO EN TODOS LOS PARAMETROS ", ES IGUAL  A
                  0.60


*********************************************
  EL PROMEDIO PONDERADO DE LOS GRADOS DE
  DE PERTENENCIA ES IGUAL A   0.72
*********************************************
QUIERES CONTINUAR? S/N
S

*************************************************************
*                                                           *
* RESULTA CONVENIENTE MODIFICAR LOS PARAMETROS SIGUIENTES:  *
*                                                           *
*************************************************************
```

- TIEMPO DE RESPUESTA

```
*****************************************************
*                                                   *
*     ANOTA LOS VALORES DE LOS PARAMETROS SIGUIENTES*
*                                                   *
*****************************************************
```

PARAMETRO 1 : TIEMPO DE RESPUESTA -EN SEGUNDOS- --> 6

GRADOS DE PERTENENCIA

```
*********************************
*     PARAMETRO 1 =   0.75      *
*     PARAMETRO 2 =   1.00      *
*     PARAMETRO 3 =   1.00      *
*     PARAMETRO 4 =   1.00      *
*     PARAMETRO 5 =   1.00      *
*     PARAMETRO 6 =   1.00      *
*     PARAMETRO 7 =   1.00      *
*     PARAMETRO 8 =   1.00      *
*********************************
```

EL GRADO DE PERTENENCIA DE LA TUPLA AL CONJUNTO DIFUSO "ALTO RENDIMIENTO EN TODOS LOS PARAMETROS", ES IGUAL A 0.75

```
*******************************************
  EL PROMEDIO PONDERADO DE LOS GRADOS DE
  DE PERTENENCIA ES IGUAL A   0.82
*******************************************
```

```
*****************************************************
* EL PROMEDIO PONDERADO DE LOS GRADOS DE PERTENENCIA *
* SE CONSIDERA DENTRO DE UN RANGO ACEPTABLE, POR    *
* LO TANTO, NO HAY NECESIDAD DE MODIFICAR PARAMETROS *
*****************************************************
```

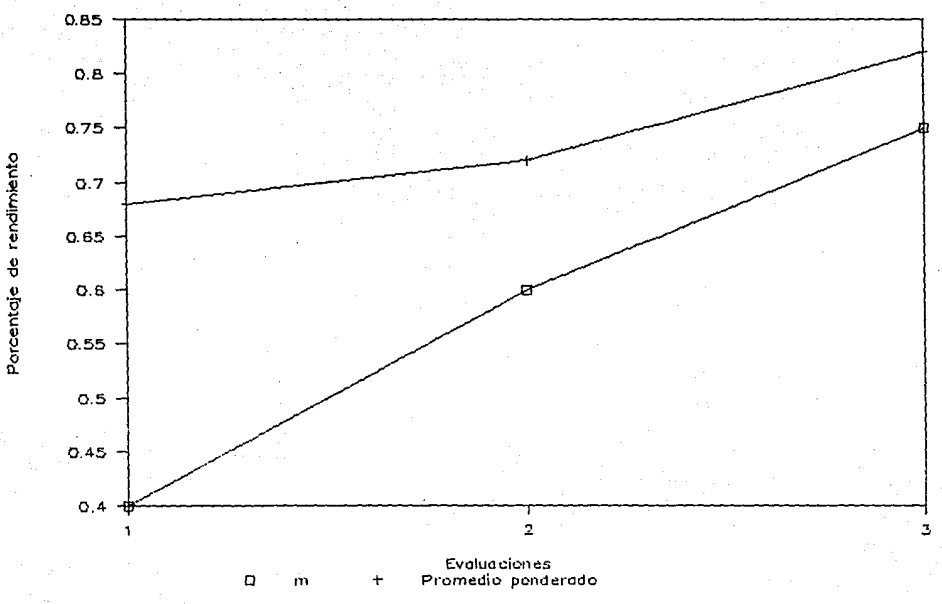
** GRADOS DE PERTENENCIA DE CADA UNO DE LOS PARAMETROS **
** EN LAS DIFERENTES EVALUACIONES DEL SISTEMA **

	PAR.1	PAR.2	PAR.3	PAR.4	PAR.5	PAR.6	PAR.7	PAR.8	m
EVALUACION 1	0.60	1.00	0.40	1.00	1.00	0.50	1.00	1.00	0.40
EVALUACION 2	0.60	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.60
EVALUACION 3	0.75	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.75

	m	PROMEDIO PONDERADO
EVALUACION 1	0.40	0.68
EVALUACION 2	0.60	0.72

VALUACION	3	0.75	0.82

EJECUCION NO. 4



C O N C L U S I O N E S.

Consideramos que se alcanzó el objetivo primordial planteado al inicio de nuestro trabajo, y que era desarrollar un sistema que sirviera de apoyo al Administrador de la Base de Datos en la difícil tarea de afinar un Sistema de Base de Datos en operación. Como pudo observarse, la tarea de afinación resulta sumamente compleja, y un sistema como el que proponemos facilitaría bastante la labor del DBA.

Un aspecto muy interesante fue el incluir conceptos novedosos dentro del área de Bases de Datos para desarrollar nuestro sistema. De esta forma se explotó un concepto que resulta potencialmente muy poderoso, no solo para el aspecto de afinación de Bases de Datos, sino dentro de cualquier otra área que requiera de la enorme flexibilidad y facilidades para representar información perteneciente al mundo real, que ofrecen las Bases de Datos Incompletas y las Bases de Datos Imprecisas.

Por ser un concepto novedoso, aún falta mucho por explorar,pero de todo lo que hemos visto a lo largo de nuestro trabajo, podemos

vislumbrar que su futuro es altamente prometedor.

En lo que se refiere al aspecto de afinación de Bases de Datos, algunos manejadores actuales ya cuentan con sistemas de afinación que se encargan de arrojar estadísticas con la información más relevante que servirá de apoyo al Administrador de la Base de Datos para realizar la tarea de afinación.

Por ejemplo, en el mercado existe un producto llamado DB2PM (IBM DATABASE 2 PERFORMANCE MONITOR), consistente en una herramienta de análisis y monitoreo del rendimiento (performance) del Sistema de Base de Datos, que ayuda al Administrador de la Base de Datos en el proceso de afinación de la misma.

Con DB2PM se pueden obtener reportes que proveen información del sistema y de sus aplicaciones. Esta información muestra, en diferentes niveles, las características de performance durante un intervalo determinado. Este tipo de sistema de monitoreo se usa en un ambiente MVS, para el DBMS conocido como DB2.

Sin embargo, este producto no ayuda al DBA a analizar los datos: solo los despliega. El análisis debe ser hecho fuera de DB2, y es ahí donde DB2PM puede ser útil.

Sin embargo, ninguna de las herramientas de afinación con que se cuenta actualmente en el mercado, manejan el concepto de imprecisión de Bases de Datos.

B I B L I O G R A F I A

1.- "Fuzzy Functional Dependencies and Lossless Join Decomposition of Fussy Relational Database Systems".
Aron k. Majumdar(University of Guelph); V.N. Raju (Andhra University).
ACM TRANSACTIONS ON DATABASE SYSTEMS.
Vol. 13, No. 2, June 1988, Pags. 129 - 166.

2.- "Database Relaxation: An approach to query processing in incomplete dabatases".
Sheldon Sheen, Ohio State University.
INFORMATION PROCESSING & MANAGEMENT.
Vol. 24, No. 2, 1988, pags. 151 - 159.
Printed in Great Britain.

3.- "Performance Tools help DBA's task".
Carmen Cifelli.
CANADIAN DATA SYSTEMS.
Vol 19, No. 3, 1987. Pag. 65

8.- "ORGANIZACION DE LAS BASES DE DATOS"
James Martin.
Editorial Mc Graw Hill. 1985.

9.- "INTRODUCCION A LOS SISTEMAS DE BASES DE DATOS".
C. J. Date.
Sistemas Tecnicos de Edicion. S.A. de C.V.
México, 1986.

10.- IBM SYSTEMS JOURNAL.
Volumen 23. Numero 2. 1984.
Pags. 189 - 211

4.- "File Structures for Database Management Systems".
Walters, Richard F.
M.D. COMPUTING.
Vol. 4, No. 5, Pags. 30 - 40, 1987.

5.- MANUAL "PERFORMANCE AND TUNING".
Editado por:
Software AG of North America, Inc. and
Software AG, Darmstadt, W. Germany.
April 1983.

6.- "GESTION DE BASES DE DATOS".
James Hannan.
Guías prácticas Chip - Auerbach.
Ediciones Arcadia, S.A.
España, 1984.

7.- "FUNDAMENTOS DE BASES DE DATOS".
Henry F. Korth, Abraham Silberschatz.
Edit. Mc Graw Hill.
Primera edición, 1987.