UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE CONTADURIA Y ADMINISTRACION

ANALISIS Y DISEÑO DE LA CONSTRUCCION DE UNA BASE DE DATOS DISTRIBUIDA

SEMINARIO DE INVESTIGACION INFORMATICA

QUE PARA OBTENER EL TITULO DE:

LICENCIADO EN INFORMATICA

P R E S E N T A :
CLAUDIA RUIZ MONTER

ASESOR DEL SEMINARIO:
ING. MIGUEL SANTIAGO SUAREZ CASTAÑON

MEXICO, D.F. **1997**

AGRADECIMIENTOS

A MI UNIVERSIDAD

Por brindarme todo lo necesario
para mi formación profesional

A MI ASESOR

Por darme su apoyo y ayuda
incondicional en todo momento.

Al L.I. J. Raymundo Iglesias L.
por dedicarme su tiempo y
paciencia.

A mis padres

Con todo mi amor , les doy gracias a papá y mamá por haberme dado la vida porque gracias a eso tuve la oportunidad de enriquecerme como persona y superarme profesionalmente.

A mis abuelas:

A mamá Mila y a tía Susy porque me enseñaron a conocer la bondad y la justicia lo cual fue importante para abrirme las puertas del éxito y sobre todo por su gran cariño incomparable.

A mis hermanos

Laly, te dedico esta tésis como agradecimiento de tu dulzura y cariño que siempre me has dado, por consertirme en todo momento y porque has sido como una segunda mamá para mí . Te quiero mucho.
Merry, en realidad tu has sido mi gran amiga y compañera de todo lo bueno y lo malo, deseo que la culminación de mi carrera te motive a terminar la tuya y que nada te impida a seguir adelante como hasta ahora lo has hecho.
Benja, amigo y hermano, porque siempre me cuidaste cuando estuviste a mi lado y deseo que esto te aliente a seguir siempre adelante y nunca dejarte vencer por nada ni nadie.

Rober, gracias a ti estudié esta carrera, pues con tu ejemplo y dedicación que siempre mostraste en la tuya como Ing. en Computación me motivaste para que yo siguiera tu ejemplo. Espero también yo lo sea para tí.
Luz, Olivia, Emiliano y Elvira les agradezco su apoyo y ayuda porque eso me sirvió de mucho para que yo siguiera adelante.

Les doy gracias a todos y cada uno de mis hermanos por quererme y confiar siempre en mí, los quiero con todo mi corazón y mi alma.

A mis amigos

A tí Samuel, te doy las gracias porque me enseñaste a querer la carrera tanto como tú la quieres.

A Tanns, por ser la amiga que me ha dado apoyo y cariño cuando más lo he necesitado y a Mac y a July porque me han brindado su amistad incondicional.

Y por último le doy gracias a ese alguien que sobre todas las cosas lo quiero y lo amo, el que siempre ha estado a mi lado guiándome y queriéndome, a mi Dios, gracias por todo lo que me ha dado.

INDICE

INTRODUCCION

El propósito de este trabajo de investigación es de tener un punto de partida para llevar a cabo el desarrollo e implementación de un manejador de base de datos distribuido, que aunque es primitivo cumple con los requisitos básicos que requiere un DDBMS. Para esto se presenta el análisis y diseño de la construcción del DDBMS y conceptos necesarios para su realización.

Día a día crece en gran medida la información de las organizaciones y las bases de datos se han convertido en una parte importante de éstas. Las grandes organizaciones se han distribuido de manera notable en diferentes regiones geográficas y se han visto en la necesidad de comunicarse una con otra para accesar a la información.

Por ese motivo se propone en esta tesis el análisis y diseño de la construcción de una base de datos distribuida, la cual proporciona ventajas en el manejo y acceso de la información pues con ella se obtiene compartición de la información de una manera rápida y confiable y además ofrece agilizar el procesamiento de las consultas entre diferentes lugares en donde se encuentra la información almacenada.

Esta tesis consta de 6 capítulos en los que se describe el prototipo del manejador de base de datos distribuidas, al que se ha denominado DISBAGE.

El capítulo 1 contiene los conceptos principales de teoría de conjuntos, álgebra relacional y cálculo relacional que son temas importantes para comprender las estructura de datos relacionales que son utilizadas en los capítulos posteriores

Los conceptos básicos de bases de datos que a lo largo de esta tesis se mencionan se encuentran en el capítulo 2. Además se establecerán las ventajas y desventajas de las bases de datos jerárquicas, de red, relacional y distribuidas.

Dentro de capítulo 3 explica lo que es una base de datos distribuidas y sus principales características así como también su arquitectura.

El análisis y diseño de la construcción del DDBMS Disbage, se establecerán en el capítulo 4. Así mismo se hablará de los requerimientos del manejador y el diseño en base al cual se llevará el desarrollo.

En el capítulo 5 se ofrecerá una discusión acerca de las características de manejadores como Oracle, Informix y Sybase en cuanto a lo que ofrecen referente a las bases de datos distribuidas haciendo así una comparación contra el trabajo de investigación que se presenta en esta tesis.

Por último, el capítulo 6 dará las conclusiones que se obtuvieron al término de la investigación.

1. TEORIA DE CONJUNTOS

Antes de analizar el modelo relacional estudiaremos la teoría de conjuntos revisando los conceptos básicos para así comprender con mayor facilidad la estructura de datos relacionales.

1.1 Conjuntos

Un conjunto es una lista o colección de elementos bien definidos. Los conjuntos se representarán con letras mayúsculas por ejemplo, A, B, X, Y y a los elementos de los conjuntos se denotarán por letras minúsculas a, b, x, y. El contenido de un conjunto se puede representar de diferentes maneras:

- Enumerando los elementos que representan. Un ejemplo sería: $A = \{ 2, 4, 6, 8 \}$
- Enunciando propiedades o reglas las cuales deciden si un objeto particular es o no elemento del conjunto por ejemplo, $B = \{ x \mid x \text{ es impar} \}$

Los conjuntos pueden ser finitos o infinitos. Un conjunto es finito si esta constituido por un número definido de elementos, si el conjunto no tiene un número definido de elementos se dice que el conjunto es infinito.

El conjunto A es igual al conjunto B si ambos tienen los mismos elementos, es decir, si cada elemento que pertenece a A pertenece también a B y si cada elemento que pertenece a B también pertenece a A. La igualdad de los conjuntos A y B se escribe $A = B$

Un conjunto A no es igual al conjunto B si ambos no contienen los mismos elementos. La desigualdad de conjuntos se representa $A \neq B$

Un conjunto vacío o conjunto nulo es aquél que carece de elementos y un ejemplo puede ser el siguiente: $B = \{x \mid x^2 = 4, x \text{ es impar}\}$, B es entonces un conjunto vacío.

$A \subset B$ denotará que el conjunto A está contenido en el conjunto B, lo que significa que todos los elementos que están en A también se encuentran en B. La negación de la definición de lo anterior se representa $A \not\subset B$.

Todo conjunto A es un subconjunto de sí mismo, B es un subconjunto propio de A si B es un subconjunto de A y, si B no es igual a A, es decir, B es un subconjunto propio de A si: $B \subset A$ y $B \neq A$.

Dos conjuntos son comparables si $A \subset B$ o $B \neq A$ esto es, si uno de los conjuntos es subconjunto del otro. En cambio, dos conjuntos A y B se dicen no comparables si $A \not\subset B$ y $B \not\subset A$. Si A no es comparable con B, entonces hay en A un elemento que no está en B y hay también en B un elemento que no está en A.

Se presentan casos en que un conjunto tiene por elementos conjuntos. Para evitar decir "conjuntos de conjuntos", se puede decir familia de conjuntos o clase de conjuntos. Es posible que un conjunto tenga entre sus elementos tanto conjuntos como elementos que no lo sean, pero es muy rara la vez que se presentan.

En toda aplicación de la teoría de conjuntos todos los conjuntos que se consideran son subconjuntos de un mismo conjunto dado. Este conjunto se llamará conjunto universal o universo del discurso y se denotará por U.

La familia de todos los subconjuntos de un conjunto S se llama conjunto potencia de S y se le denomina por 2^S

Si dos conjuntos A y B no tienen elementos comunes, es decir, sin ningún elemento de A está en B y si ningún elemento de B está en A, se dice que A y B son conjuntos disjuntos.

1.2 Conjuntos productos

Un par ordenado es un conjunto compuesto de dos elementos a y b entre los que existe un orden, de manera que a es el primer elemento y b el segundo. Un par ordenado se representa por (a, b).

Dos pares ordenados (a,b) y (c,d) son iguales si, y solamente si, $a=c$ y $b=d$. Dados dos conjuntos A y B, se llama conjunto producto A y B el conjunto de todos los pares ordenados (a, b) con $a \in A$ y $b \in B$. El conjunto producto de A y B se representa así $A \times B$, por ejemplo sean $A = \{1, 2, 3\}$ y $B = \{a, b\}$. El producto conjunto es entonces

$$A \times B = \{ (1, a), (1,b), (2,a), (2,b), (3,a), (3,b) \}$$

El conjunto producto $A \times B$ se llama también producto cartesiano de A y B, este nombre se le atribuye porque el primero en investigar el conjunto producto fue el matemático Descartes en el siglo diecisiete.

Si el conjunto A tiene n elementos y el conjunto B tiene m elementos, entonces el conjunto producto $A \times B$ tiene n veces m elementos, esto es, nm elementos. Si uno de los conjuntos A o B es vacío, entonces $A \times B$ es vacío. Si uno de los conjuntos es infinito y el otro no es vacío, entonces $A \times B$ es infinito. El producto cartesiano de dos conjuntos no es conmutativo, es decir, que $A \times B \neq B \times A$ a menos que $A=B$ o que uno de los dos conjuntos sea vacío.

1.3 Relaciones

Se llama predicado definido sobre el producto cartesiano $A \times B$ de dos conjuntos A y B, a una expresión denotada por $P(x,y)$ que se caracteriza porque cuando en $P(x,y)$ se sustituyen las variables x e y, respectivamente, por a y b, se convierte en un enunciado ya sea verdadero o falso, para todo par ordenado $(a, b) \in A \times B$. Por ejemplo, si A es el conjunto de profesores y B el de estudiantes, entonces $P(x,y) =$ << x enseña a y >>

Una relación R es un subconjunto AxB, donde se cumple que existe p tal que, para todo $(x,y) \in AxB$ y $P(x,y)=V$ entonces $(x,y) \in R$. Una relación n-aria sobre A es un subconjunto de A^n.

Para una relación R redefinimos las siguientes propiedades:

R es reflexiva en A si y solo si $(x,x) \in R$ para cualquier $x \in A$.

R es simétrica si y solo si $(x,y) \in R$ para cualquier $(y,x) \in R$.

R es transitiva si y solo si $(x,y) \in R \wedge (y,z) \in R$ entonces $(x,z) \in R$.

R satisface tricotomía en A si y solo si para cada x,y en A solo se cumple una de las siguientes posibilidades :

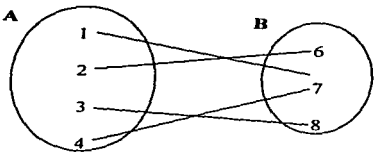
$$(x,y) \in R,\ x = y,\ \text{ó}\ (y,x) \in R$$

R es una relación de orden en A si y solo si R es transitiva y satisface tricotomía en A.

1.4 Funciones

Una función f de A en B es un subconjunto de AxB en el cual cada $a \in A$ aparece como primer elemento en un par ordenado de f y solo en uno. Como todo subconjunto de AxB es una relación, una función es un tipo especial de relación. En otras palabras, una función es cuando a cada elemento de un conjunto A le corresponde un elemento único de un conjunto B. Una función se representa $f: A \rightarrow B$, esto se lee, f es una función de A en B. El conjunto A se llama dominio de la función f, y B se llama codominio de f. Si $a \in A$, el elemento de B que le corresponde a a se le llama imagen de a y se representa por $f(a)$ que se lee f de a.

Por ejemplo sea $A = \{ 1, 2, 3, 4 \}$, $B = \{6, 7, 8\}$, y $f:A \to B$ la definida por el diagrama



Se dice que f y g son funciones iguales, denotándose como $f=g$, si están definidas en el mismo dominio D y si $f(a)=g(a)$ para todo $a \in D$. Una función es inyectiva si elementos distintos de B corresponden a elementos distintos de A. Por ejemplo: La función f que asigna a cada país del mundo su ciudad capital es una función inyectiva, ya que países distintos tienen capitales diferentes, es decir, ninguna ciudad es la capital de dos países diferentes.

Dada $f:A \to B$ el conjunto de imágenes f es subconjunto de B, esto es, $f(A) \subset B$. Si $f(A)=B$, es decir, si todo elemento de B es imagen de al menos un elemento A, se dice entonces que f es una función sobreyectiva de A en B o que f es una función de A sobre B, o bien que f aplica A sobre B. Por ejemplo sea, $f:A \to B$ la función del ejemplo del diagrama, vea que $f(A)=(6, 7, 8)=B$, esto es, que la imagen de f es igual al codominio B. Así, pues, f aplica sobre B y se puede decir entonces que es una función sobreyectiva.

La función $f:A \to A$, definida por $f(x)=x$, o sea la función f que hace corresponder a cada elemento de A el mismo elemento, se llama función idéntica o transformación idéntica sobre A.

Una función f de A en B se llama función constante si a cada elemento de A se le asigna el mismo elemento $b \in B$, es decir, si $f:A \to B$ es una función constante si el dominio de la imagen de f consta de un elemento solamente.

1.5 Algebra relacional

El álgebra como cualquier otro sistema matemático deductivo, puede definirse con un conjunto de elementos, un conjunto de operadores y un número de axiomas no probados o postulados. Un operador binario definido en un conjunto de S de elementos es una regla que se asigna a cada par de elementos de S un elementos único de S. Como ejemplo, considérese la relación $a*b=c$. Se dice que * es un operador binario y especifica una regla para encontrar c mediante el par (a,b) y también si $a,b,c \in S$. Si embargo, * no es operador binario si $a, b \in S$, si la regla encuentra que $c \notin S$.

Los postulados de un sistema matemático forman los supuestos básicos mediante los cuales es posible deducir las reglas, teoremas y propiedades del sistema. Los postulados más comunes que se utilizan para formular diversas estructuras algebraicas son:

1. Cierre. Un conjunto S está cerrado con respecto a un operador binario si, para cada par de elementos de S, el operador binario especifica una regla para obtener un elemento único de S. Por ejemplo, el conjunto de los números naturales $N=\{1,2,3,4,...\}$ está cerrado con respecto al operador binario más (+) por las reglas de la adición aritmética, ya que para cualquier $a,b \in N$ se obtiene una única $c \in N$ por la operación $a+b=c$. El conjunto de los números naturales no está cerrado con respecto al operador binario menos (-) por las reglas de la resta artimética debido a que $2-3=-1$ y $2,3 \in N$, ya que $(-1) \notin N$.

2. Ley asociativa. Un operador binario * en un conjunto S se dice que es asociativo siempre que $(x*y)*z = x*(y*z)$ para todo $x, y, z \in S$

3. Ley conmutativa. Un operador binario * en un conjunto S se dice que es conmutativo siempre que $x*y = y*x$ para todo $x,y \in S$.

4. Elemento Identidad. Un conjunto S se dice que tiene un elemento identidad respecto a una operación binaria * en S si existe un elemento $e \in S$ con la propiedad: $e*x = x*e = x$ para cada $x \in S$.

5. Inversa. Un conjunto S que tiene el elemento identidad e con respecto a un operador binario * se dice que tiene una inversa siempre que, para cada $x \in S$, existe un elemento $y \in S$ tal que $x*y=e$.

6. Ley distributiva. Si * y . son dos operadores binarios en un conjunto S, * se dice que es distributivo sobre . siempre que: x*(y.z) = (x*y).(x*z).

Los operadores y los postulados tienen los siguientes significados:

- **El operador binario + define la adición.**
- **La identidad aditiva es 0.**
- **La inversa aditiva define la sustracción.**
- **El operador binario . define la multiplicación.**
- **La identidad multiplicativa es 1.**
- **La inversa multiplicativa de $a = 1/a$ define la división, esto es, $a \ 1/a = 1$.**
- **La única ley distributiva aplicable es la de . sobre +: $a . (b+c)=(a.b)+(a.c)$**

El <u>álgebra relacional</u> es una serie de operaciones sobre relaciones donde cada operación usa una o más relaciones como sus operandos y produce otra relación como resultado. Dado que el resultado de una operación del álgebra relacional es una relación, dicha relación se puede utilizar para otras operaciones algebraicas.

Cuando dos entidades están definidas sobre los mismos dominios, se pueden considerar como conjuntos del mismo universo: el conjunto de todos las posibles *n*adas en la relación resultante del producto cartesiano de esos dominios.

En el álgebra relacional que aquí se presenta se consideran dos tipos de operadores:

1. Los operadores tradicionales sobre conjuntos:

   Unión, intersección, diferencia, producto cartesiano.

2. Los operadores especiales:

   Proyección, Selección, Join, División.

Los operadores especiales son:

Proyección.

Sea $R(A_1,A_2,...,A_n)$ una relación de orden n con atributos $A_1,A_2,...,A_n$, $X \subseteq \{A_1,A_2...,A_n\}$ y $Y = \{A_1,A_2...,A_n\}/X$ permutando los atributos de R podemos representar a $R\{A_1,A_2,...,A_n\}$ como $R(X,Y)$. El resultado de la operación de proyección de la relación R sobre los atributos X, representando como $\pi_X(R)$, se define como:

$$\pi_X(R) = \{ x | \exists y \text{ tal que } (x, y) \in R(X, Y) \}$$

Selección

Sea $R(A_1, A_2, ..., A_n)$ una relación de orden n con atributos $A_1, A_2, ..., A_n$ y P una condición lógica definida sobre el conjunto $D_1 \times D_2 \times ... \times D_n$ donde D_i es el dominio del atributo A_i (i= 1...n), el resultado de la selección en la relación R con respecto a la condición P, representado como $\sigma_p(R)$, se define como:

$$\sigma_p(R) = \{x | x \in R \wedge P(x) \text{ es verdadero } \}$$

Join

Sean $R(A_1,A_2, ...,A_n)$ y $S(B_1,B_2, ...,B_m)$ dos relaciones y, $X \subseteq \{A_1,A_2, ...,A_n\}$ y $Y \subseteq \{B_1, B_2, ..., B_m\}$ dos conjuntos de atributos, asumiendo que X y Y tienen el mismo número de atributos y que los atributos correspondientes están definidos sobre el mismo dominio. Si escribimos $Z = \{A_1, A_2, ..., A_n\} / X$ y $W = \subseteq \{B_1, B_2, ..., B_m\} / Y$ entonces, permutando el orden de los atributos, las relaciones R y S pueden representarse como $R(Z,X)$ y $S(Y,W)$. El "Join" natural[2] de las relaciones R y S sobre los atributos X y Y, representado como:

$R \underset{x=y}{\lhd\rhd} S$ se define como $R \underset{x=y}{\lhd\rhd} S = \{(z,x,w) | (z,x) \in R \wedge (y,w) \in S \wedge x = y\}$

[2] Se define el equijoin, ya que para realizar el join natural es necesario que los atributos tengan el mismo nombre en las relaciones R y S, sin embargo, cuando los nombres de los atributos que participan en la condición de join son diferentes, el equijoin es equivalente al join natural.

Los operadores tradicionales sobre conjuntos, con excepción del producto cartesiano,. tienen que ser compatibles a la unión, es decir, los operandos tienen que ser del mismo grado y los atributos correspondientes deben estar definidos sobre el mismo dominio.

Unión

La unión de dos relaciones compatibles R y S representada como $R \cup S$ es el conjunto de todas las *n*adas que están en R, en S o en ambas, es decir, $R \cup S = \{t/t \in R \vee t \in S\}$

Intersección

La intersección de dos relaciones compatibles R y S ($R \cap S$) es el conjunto de las *n*adas t que pertenecen a ambos R y S, es decir, $R \cap S = \{t/t \in R \wedge t \in S\}$

Diferencia

La diferencia entre dos relaciones compatibles R y S en ese orden, $R - S$, es el conjunto de *n*adas que pertenecen a R pero no a S, es decir: $R\text{-}S = \{t/t \in R \wedge t \notin S\}$

Por ejemplo sean dos relaciones, R y S definidas sobre los dominios $A\ B\ C$:

$$R=\{(a1,b1,c1), (a1,b2,c1), (a2,b1,c2)\}$$

$$S=\{(a1,b1,c1), (a2,b2,c1), (a2,b2,c2)\}$$

Los resultados de aplicar las operaciones $R \cap S$, $R \cup S$ y $R\text{-}S$ se muestra a continuación.

$R \cap S = \{(a1,b1,c1)\}$

$R \cup S = \{(a1,b1,c1), (a1,b2,c1), (a2,b1,c2), (a2,b2,c1), (a2,b2,c2)\}$

$R\text{-}S = \{(a1,b2,c1), (a2,b1,c2)\}$

Producto Cartesiano

El producto cartesiano de dos relaciones ***A*** y ***B*** es el conjunto de todas los *nadas* ***t*** tales que ***t*** es una concatenación de una *nada* $a \in A$ y una *nada* $b \in B$. La concatenación de una *nada* $a=(a_1, a_2, \ldots a_m)$ y una *nada* $b=(b_1,b_2,\ldots,b_n)$, en ese orden, es la *nada* $t=(t_1,t_2,\ldots,t_m,t_{m+1},\ldots,t_{m+n})$, donde

$$t_i \begin{cases} a_i \text{ para } i=1,\ldots,m \\ b_j \text{ para } i=m+1,\ldots,m+n(j=i-m) \end{cases}$$

Por ejemplo sea ***R*** y ***S***:

R= {(a1,b1,c1), (a1,b2,c1), (a2,b1,c2)}

S={(d1,e1,f1), (d2,e2,f1), (d2,e2,f2)}

El producto cartesiano aplicado a estas relaciones daría como resultado:

$R \times S$={(a1,b1,c1,e1,f1), (a1,b1,c1,d2,e2,f1), (a1,b1,c1,d2,e2,f2), (a1,b2,c1,d1,e1,f1), (a1,b2,c1,d2,e2,f1), (a1,b2,c1,d2,e2,f2), (a2,b1,c2,d1,e1,f1), (a2,b1,c2,d2,e2,f1), (a2,b1,c2,d2,e2,f2)}

División

Sean $R(x,y)$ y $S(z)$ dos relaciones, donde x,y,z son conjuntos de atributos. Asumiendo que y y z contienen el mismo número de atributos y que los dominios de los atributos correspondientes son iguales. El resultado de la división de la relación $R(x,y)$ con la relación $S(z)$, representado como $R \div S$, se define como el subconjunto máximo de la proyección $\pi_X(R)$ tal que su producto cartesiano con $S(z)$ está contenido en $R(x,y)$. Es decir, tenemos: $R(x,y) = ((R \div S) \times S(z)) \cup Q(x,y)$ donde $R \div S$ es el cociente de la división y $Q(x,y)$ es el residuo de la división. El resultado de la división es el conjunto de las *nadas* ***t*** tales que para todos las *nadas* $s \in S$ existe una *nada* $r \in R(x,y)$ que satisface las siguientes condiciones: $\pi_Y(r) = s$ y $\pi_X(r) = t$. A partir de las dos condiciones anteriores se puede observar que la división se puede representar en función de las otras operaciones como sigue: $R \div S = (\pi_X R) - (\pi_X((\pi_X R \times S(z)) - R(x,y))$

1.6 Cálculo Relacional

Cálculo relacional no implica ninguna conección con la rama de las matemáticas llamada "Cálculo Diferencial e Integral". Cálculo Relacional viene a ser el cálculo de primer orden o cálculo de predicados, es decir, del campo de la lógica.

Las expresiones de cálculo relacional de tuplos son de la forma $\{t/\psi\ (t)\}$, donde t es un tuplo variable, es decir, una variable denotando un tuplo de alguna longitud compuesta y ψ es una fórmula construida de átomos y una colección de operadores que serán definidos brevemente.

Los átomos de fórmulas ψ son de 3 tipos:

1. $R(s)$, donde R es un nombre de relación y s es una variable de tuplo. Este átomo afirma que s es un tuplo de la relación R.

2. $s[i]\ \theta\ u[j]$, donde s y ψ son tuplos variables y θ es un operador de comparación aritmética (<, =, etc). Este átomo demuestra que el componente i^{th} de s esta en relación θ al componente j^{th} de u. Por ejemplo, $s[1] < u[2]$ significa que el primer componente de s es menor que el segundo componente de u.

3. $s[i]\ \theta\ a$ y $a\ \theta\ s[i]$, donde θ and $s[i]$ son como en el punto anterior, y a es una constante. El primero de estos átomos afirman que el componente i^{th} de s permanece en la relación θ a la constante a, y el segundo tiene un significado análogo. Por ejemplo $s[1] = 3$ significa que el valor de el primer componente de s es 3.

Para explicar los operadores de cálculo relacional, es necesario definir las nociones de variables de tupla libre y limitado.

Estas nociones son iguales a las empleadas en el cálculo de predicados. Informalmente, la ocurrencia de una variable en una fórmula es limitada si esta variable ha sido introducida por un cuantificador existencial o un cuantificador universal, y se dice que la variable es libre si esto no pasa.

La noción de una variable libre es análoga a aquella de una variable global en un lenguaje de programación, es decir, una variable definida fuera de el procedimiento actual. Una variable limitada es como una variable local, como aquella que esta definida en el procedimiento en uso, se la puede utilizar solo ahí y no puede ser referenciada fuera de el. En efecto, los cuantificadores de cálculo relacional juegan un papel de declaraciones en un lenguaje de programación.

Las fórmulas, así como las ocurrencias libres y limitadas de variables de tuplos en estas fórmulas se definen recursivamente como sigue:

1. Cada átomo es una fórmula. Todas las ocurrencias de variables de tuplo mencionadas en el átomo son libres en esta fórmula.

2. If ψ_1 y ψ_2 son fórmulas entonces $\psi_1 \wedge \psi_2$, $\psi_1 \wedge \psi_2$, y $\neg\psi_1$ son fórmulas que afirman que ψ_1 y ψ_2, son ambas verdaderas, ψ_1 o ψ_2, o ambas, son verdaderas, y ψ_1 es falsa, respectivamente. Ocurrencias de variables de tuplos son libres o limitadas dentro $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, y $\neg\psi_1$ así como son libres o limitadas en ψ_1 o ψ_2, dependiendo del lugar donde ocurren. Debe notarse que una ocurrencia de una variable s pudo ser limitada en ψ_1, mientras que otra ocurrencia de s es libre en ψ_2, o a la inversa.

3. Si ψ es una fórmula, entonces $(\exists s)(\psi)$ es una fórmula. El símbolo $\exists$ es un cuantificador existencial. El otro cuantificador que se usa $\forall$ el cual se lee como universal, se describe en el siguiente punto. Ocurrencias de s que son libres dentro de u son limitadas para $(\exists s)(\psi)$ dentro $(\exists s)(\psi)$. Otras ocurrencias de variables de tuplo en ψ, que incluyen ocurrencias posibles de s que fueron limitadas en ψ, son libres o limitadas en $(\exists s)(\psi)$ como lo fueron en ψ. La fórmula $(\exists s)(\psi)$ afirma que existe un valor de s que cuando se sustituye este valor para todas las ocurrencias de s en ψ, la fórmula ψ se convierte en verdadera. Por ejemplo, $(\exists s)(\boldsymbol{R}(s))$ dice que la relación $\boldsymbol{R}$ no esta vacía, es decir, que existe un tuplo de s en $\boldsymbol{R}$.

4. Si u es una fórmula, entonces $(\forall s)(\psi)$ es una fórmula. Ocurrencias libres de s en u son limitadas para $(\forall s)$ en $(\forall s)(\psi)$, y otras ocurrencias de variable en u son tratadas como en el punto anterior. La fórmula $(\forall s)(\psi)$ confirma que cualquier valor de la aridad apropiada que sustituyamos por ocurrencias libres de s en ψ, la fórmula ψ llega a ser verdadera.

5. Los parantésis pueden colocarse alrededor de la fórmula como sea necesario. Asumimos que el orden de precedencia es: el operador más alto de comparación aritmética, que son los cuantificadores $\exists$ y $\forall$, y después $\neg$, $\wedge$, y por último $\vee$. Este es el orden a seguir.

6. Ninguna otra cosa es una fórmula.
   Una expresión de cálculo relacional de tuplos es un expresión de la forma $\{t \mid \psi(t)\}$ donde t es la única variable de tuplo libre en ψ.

Es decir "el conjunto de tuplos de t tal que t esta en R o t esta en S". Nótese que la unión solo tienen sentido si R y S tienen la misma aridad, y similarmente, la fórmula $R(t) \vee S(t)$ solamente tiene sentido si R y S tiene la misma aridad, ya que la variable de tuplo t esta asumiendo que tiene algún tamaño determinado.
La diferencia $R - S$ se expresa por $\{t \mid R(t) \wedge \neg S(t)\}$. Si R y S son relaciones de aridad r y s, respectivamente, entonces $R \times S$ puede expresada en cálculo por:

$$\{t^{(r+s)} \mid (\exists u^{(r)})\,(\exists v^{(s)})(R^{(u)} \wedge S^{(v)} \wedge t[1] = u[1] \wedge t[r] = u[r] \wedge t[r+1] = v[1] \wedge \ldots \wedge t[r+s] = v[s])\}$$

Recordando que $t(i)$ indica que t tiene aridad i. Es decir, $R \times S$ es el conjunto de tuplos t (lo que entendemos que es el tamaño $r+s$) tal que existe u y v, con u en R , v en S; los primeros componentes r de t forman u, y los siguientes componentes s de t forman v.

La proyección $\pi_{i_1,i_2,\ldots,i_k}(R)$ es expresada por

$$\{t^{(k)} \mid (\exists u)(R(u) \wedge t[1] = u[i_1] \wedge \ldots \wedge t[k] = u[i_K])\}$$

La selección $\sigma_F(R)$ es expresado por $\{t \mid R(t) \wedge F'\}$, donde F' es la fórmula F con cada operando i, denotando el componente i^{th}, reemplazado por $t[i]$. Como un último ejemplo, si R es una relación de aridad dos, entonces $\{t^{(u)} \mid (\exists u)(R(t) \wedge R[u] \wedge (t[1] \neq u[1] \vee t[2] \neq u[2]))\}$ es una expresión de cálculo que denota R si R tiene dos o más elementos y denota la relación vacía si R esta vacía o tiene un solo elemento.

Restricciones de Cálculo Relacional para proporcionar solo relaciones finitas

El Cálculo Relacional de tuplos como lo hemos definido nos permite definir algunas relaciones infinitas tales como $\{t \mid \neg R(t)\}$, el cual denota todos los tuplos posibles que no están en R , pero son del tamaño que asociamos con t. Ese tamaño tiene que ser la aridad de R para que la expresión tenga sentido. Como no se pudo imprimir todos los tuplos posibles (¿sobre que dominio?), debemos suprimir aquellas expresiones insignificantes, lo que usualmente se hace es restringir la consideración a ciertas expresiones $\{t \mid \psi(t),\}$ las cuales llamamos *seguras*.

Para definir seguridad, antes que nada definiremos DOM(ψ) que es el conjunto de símbolos que pueden aparecer o no explícitamente en la expresiones ψ o son componentes de algunos tuplos en alguna relación R mencionada en ψ. Esta opción de DOM(ψ) no es necesariamente el conjunto más pequeño de símbolos pudimos usar, pero puede ser suficiente. Informalmente, una expresión ψ es *segura* si cada

componente de cualquier t que satisface ψ debe ser un elemento de DOM(ψ). Una definición más formal es la siguiente:

Nótese que DOM(ψ) no esta determinada al ver ψ pero es una función de las relaciones actuales para ser sustituidas para las variables de relación en ψ. Sin embargo, como todas las relaciones son asumidas finitas. DOM(ψ) por ejemplo siempre es finita, por ejemplo si $\psi(t)$ es $t[1] = a \vee \boldsymbol{R}(t)$, donde $\boldsymbol{R}$ es una relación binaria, entonces DOM(ψ) es la relación unaria dada por la fórmula de álgebra relacional $\{a\} \cup \pi_1(\boldsymbol{R}) \cup \pi_2(\boldsymbol{R})$.

Decimos que una expresión de cálculo de tuplos $\{t|\psi(t)\}$ es segura si:

1. Siempre que t satisface u, cada componente de t es un elemento de DOM(ψ).
2. Para cada subfórmula de ψ de la forma $(\exists u)(w(u))$, si w se satisface por u para cualquier otros valores de las otras variables libres en w, entonces cada componente de u es un elemento de DOM(w).
3. Para cada subfórmula de u de la forma $(\forall u)(w(u))$, si cualquier componente de u no esta en DOM(w), entonces u satisface w para todos los valores de otras variables libres en w.

El propósito de los puntos (2) y (3) es asegurar que podemos determinar la verdad de una fórmula cuantificada $(\exists u)(w(u))$ o $(\forall u)(w(u))$ al considerar solamente aquellos símbolos compuestos de u en DOM(w). Por ejemplo, cualquier fórmula $(\exists u)(\boldsymbol{R}(u) \wedge ...)$ satisface (2) y cualquier fórmula $(\forall u)(\neg \boldsymbol{R}(u) \vee ...)$ satisface (3). Nótese que en la definición de seguridad, no asumimos que cualquier variable libre de w, además u, necesariamente tiene valores en DOM(w). Las Reglas (2) y (3) deben permanecer independientemente de el valor de aquellas variables.

Mientras la reglas (3) puede aparecer intituivamente, debemos observar que la fórmula $(\forall u)(w(u))$ es lógicamente equivalente a $\neg(\exists u)(\neg w(u))$. La última fórmula es insegura si y solo si hay un u_0 para la cual $\neg w(u_0)$ es verdadera, y u_0 no esta en el dominio de la fórmula $\neg w$. Como los dominios de w y $\neg w$ son los mismos, la regla (3) dice que la fórmula $(\forall u)(w(u))$ es exactamente segura cuando la fórmula $\neg(\exists u)(\neg w(u))$ es segura.

Cálculo Relacional de Dominio

El cálculo relacional de dominio esta construido de los mismos operadores como el cálculo relacional de tuplos. Las diferencias son:

1. No hay variables de tuplos en el cálculo de dominio, pero hay variables de dominio para representar componentes de tuplos en su lugar.

2. Un átomo puede estar en cualquiera de estas formas:

i) $R(x_1,x_2,...x_k)$, donde R es una relación $n$aria y cada x_i es un constante o variable de dominio.

ii) $x\ \theta\ y$, donde x y y son constantes de variables de dominio, y θ es un operador relacional aritmético.

$R(x_1,x_2...x_k)$ confirma que los valores de estos x_i que son variables, deben ser escogidos de una manera en que $x_1,x_2,....x_k$ sea un tuplo en R. El significado de un átomo $x\ \theta\ y$ es que x y y deben tener valores que hagan $x\ \theta\ y$ sean verdadero.

3. Fórmulas en el cálculo relacional de dominio usan la conectores $\wedge$, $\vee$, y $\neg$, como en el cálculo de tuplos. Nosotros también utilizamos $(\exists x)$ y $(\forall x)$ para formar expresiones de cálculo de dominio, pero x es una variable de dominio en lugar de una variable de tuplo.

Las nociones de variables de dominio libres y limitadas y la intención de una variable limitada son definidas exactamente en el cálculo de dominio como en el cálculo de tuplos, y por lo tanto no repetiremos estas definiciones aquí. Una expresión de cálculo de dominio esta de la forma $\{x_1,x_2,...x_k \mid \psi(x_1,x_2,...x_k)\}$, donde ψ es una fórmula cuyas variables de dominio libre son de variables distintas $x_1, x_2,...x_k$.

En analogía con el cálculo de tuplos, definimos una expresión de cálculo de dominio $\{x_1,x_2,...,x_k \mid \psi(x_1,x_2,...x_k)\}$ para que esta sea segura si:

1. $\psi(x_1,x_2,...x_k)$ es verdadera implica x_i este en DOM(ψ),
2. Si $(\exists u)(w(u))$ es una subfórmula de ψ, entonces $w(u)$ es verdadera para cualquier valor de las variables libres de w (además de u) implica que u este en DOM(w) y
3. Si $(\forall u)(w(u))$ es una subfórmula de ψ, entonces $w(u)$ es falsa para cualquier valor de las variables libres de w(además de u) implica que u este en DOM(w)

Reduciendo Cálculo de Tuplos a Cálculo de Dominio

La construcción de una expresión de cálculo de dominio equivalente a una expresión de cálculo de tuplos dada $\{t \mid \psi(t)\}$ es directa. Si t tiene aridad n, introduce n nuevas variables de dominio $t_1,t_2,...t_k$ y reemplaza la expresión por $\{t_1,t_2...t_k \mid \psi(t_1,t_2,...,t_k)\}$ donde ψ es ψ con cualquier átomo $R(t)$ reemplazado por $R(t_1,t_2,...,t_k)$, y cada ocurrencia libre de $t[i]$ reemplazado por t_i. Nótese que puede haber ocurrencias limitadas de t en ψ, si hubiera un cuantificador $(\exists t)$ o $(\forall t)$; los usos de esta t se refieren a una variable de tuplo *diferente* y no son reemplazados.

Además, para cada cuantificador $(\exists u)$ o $(\forall u)$, si u tiene aridad m, introduce m nuevas variables de dominio $u_1, u_2,...,u_m$, y, dentro de el propósito de esta cuantificación de u, reemplaza $u[i]$ por u_i, y $R(u)$ por $R(u_1,u_2...u_m)$. Reemplaza $(\exists u)$ por $(\exists u_1)...(\exists u_m)$ y reemplazar $(\forall u)$ por $(\forall u_1)...(\forall u_m)$. El resultado es una expresión en el cálculo de dominio que es equivalente a la original expresión del cálculo de tuplos.

Debe estar claro que los valores que se asumen por t_i son exactamente aquellos que pueden ser asumidos por $t[i]$ en la expresión original. De esta manera, si $\{t/\psi(t)\}$ es segura entonces el resultado de la expresión de cálculo de dominio también será segura. Por lo tanto establecemos el siguiente teorema sin más prueba:

Teorema. para cada expresión segura de cálculo relacional de tuplo hay una expresión segura equivalente en el cálculo relacional de dominio.

2. CONCEPTOS BASICOS DE BASES DE DATOS

2.1 Sistema de base de datos

La tecnología de la base de datos es una de las áreas en las ciencias de la computación que ha tenido más avance en los últimos años, pues millones de organizaciones dependen de sistemas de bases de datos para tener un buen control de su información.

Un sistema de base de datos es un conjunto de datos almacenados en una computadora los cuales están interrelacionados. Los datos son compartidos por diferentes usuarios y programas de aplicación, pero existe un mecanismo para la inserción, modificación, borrado o consulta de estos.

Los usuarios finales y los programadores de aplicación no necesitan saber como se encuentran las estructuras de almacenamiento de un sistema de base de datos para poder usarlo. En la Figura 2.1. se muestra el esquema de un sistema de base de datos.

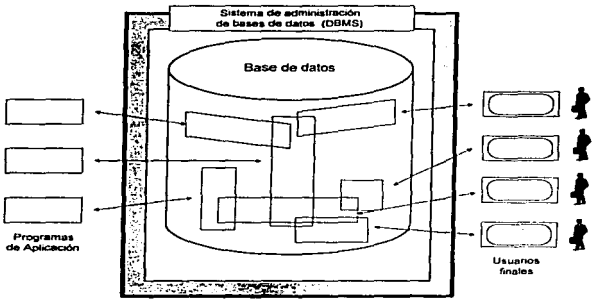


Fig. 2.1 Representación simplificada de un sistema de base de datos

Un sistema de base de datos se compone por datos, hardware, software y usuarios. Los datos son los valores registrados físicamente en la base de datos, éstos se almacenan en el sistema y se dividen en una o más bases de datos. Una base de datos es tanto integrada como compartida. Es *Integrada* debido a que la unificación de archivos de datos independientes elimina cualquier redundancia que se pueda presentar entre ellos mismos. Es *Compartida* pues la base de datos es compartida entre varios usuarios y éstos pueden tener acceso a una misma parte de ella y utilizarla con fines totalmente diferentes.

El hardware se compone de los volúmenes de almacenamiento secundario, donde reside la base de datos, junto con dispositivos asociados como las unidades de control, los canales , etc.

El software es el que permite el uso y/o modificación de los datos de una base de datos y se le conoce como el Sistema Manejador de Bases de Datos (DBMS: "DataBase Management System"). La función que tiene el DBMS es que el usuario no tenga que intervenir a nivel hardware y ayuda a que éste pueda realizar las operaciones de inserción, borrado, consulta o modificación de la base de datos.

Las principales funciones de un DBMS son abstracción, seguridad, integridad, reducción de redundancia, compartición de datos, protección contra fallas y recuperación de la información. Abstracción se refiere a que el usuario pueda tratar con los datos en términos abstractos en lugar de tratar con ellos en la forma que los almacena la computadora.

No todos los usuarios pueden tener acceso a la base de datos. El usuario debe de tener permiso para poder accesar a ella y la persona que controla estos permisos es el Administrador de la Base de Datos (DBA) [1] cuya responsabilidad es controlar las datos de operación. Este mecanismo da una seguridad a la base de datos contra usuarios no autorizados. La información que se obtenga de la base de datos debe ser veraz y por lo tanto confiable. Para que la información sea consistente se requiere que exista integridad.

Un DBMS ayuda a evitar que se repitan datos almacenados en varias partes de la base de datos. Al reducirse la redundancia también se evita que la base se encuentre en estado de inconsistencia lo cual provoca que proporcione información incorrecta.
No solo se comparten la base de datos entre distintos usuarios sino también permite que programas de aplicación operen con los mismos datos almacenados, es decir, se pueden compartir datos entre aplicaciones.

Otra función que tiene un DBMS es proteger a la base de datos contra fallas ya sea de hardware o de software y además permite la recuperación de la información.

2.2 Independencia de los datos

La independencia de los datos es la "inmunidad de las aplicaciones a los cambios de la estructura de almacenamiento y de la estrategia de acceso" según [C.J. Date, 1983], es decir, la manera como los datos se organizan en almacenamiento secundario y la manera como se accesan es independiente de los requerimientos de la aplicación.

La independencia de los datos implica la separación de las propiedades lógicas y físicas. La independencia lógica de los datos se refiere al tratamiento de los

[1] En un ambiente de bases de datos accesan muchos usuarios, los cuales tienen distintos requerimientos. Para la coordinación y dirección del diseño, implantación y mantenimiento de una base de datos se crea la función de un Administrador de la base de datos (más adelante se dará más detalle de la función del DBA).

requerimientos individuales y de conjunto separadamente, es decir, tener la capacidad de modificar el esquema conceptual sin obligar a que se vuelvan a escribir los programas de aplicaciones. Las modificaciones en el nivel conceptual son necesarias siempre que se altera la estructura lógica de la base de datos y la independencia física de los datos se refiere al tratamiento de los requerimientos funcionales y de ejecución separadamente, es decir, la capacidad de modificar el esquema físico sin obligar a que se vuelvan a escribir los programas de aplicaciones. En algunas ocasiones son necesarias modificaciones en el nivel físico para mejorar el rendimiento.

La independencia lógica de los datos es más fácil de lograr que la independencia física, ya que los programas de aplicaciones dependen de la estructura lógica de los datos a los que tienen acceso.

Un esquema de base de datos se especifica por medio de una serie de definiciones que se expresan en lenguaje especial llamado lenguaje de definición de datos (DDL, lenguaje de definición de datos) El resultado de la compilación de las proposiciones en DDL es un conjunto de relaciones que se almacenan en un archivo especial llamado diccionario de datos.

Un diccionario de datos es un archivo que contiene metadatos, es decir, "datos acerca de los datos". Este archivo se revisa antes de utilizar los datos reales en el sistema de base de datos.

La estructura de almacenamiento y los métodos de acceso utilizados por sistema de base de datos se especifican por medio de un conjunto de definiciones de un tipo especial de DDL llamado lenguaje de almacenamiento y definición los datos. El resultado de la compilación de estas definiciones es una serie de instrucciones que especifican los detalles de implantación de los esquemas de base de datos que no ven los usuarios.

Un lenguaje de manejo de datos (DML, data manipulation language) permite a los usuarios manejar o tener acceso a los datos que estén organizados por medio del modelo apropiado. Existen básicamente dos tipos de DML, de procedimientos y sin procedimientos.

- **De procedimientos,** necesitan que el usuario especifique cuáles datos quieren y cómo deben obtenerse.
- **Sin procedimientos,** requieren que el usuario especifique cuáles datos quiere sin especificar cómo obtenerlos.

2.3 Estructura de almacenamiento

Un archivo almacenado puede realizarse físicamente de diversas maneras. Una de ellas puede ser que el archivo este contenido por completo en un volumen de almacenamiento (paquetes de discos) o distribuirse en diferentes volúmenes de varios tipos de almacenamiento. También puede tener un ordenamiento físico en secuencia según los valores de algún campo almacenado. Puede ser o no accesible por medio de direccionamiento por dispersión. Los registros almacenados pueden agruparse en bloques. Todo lo anterior se puede hacer siempre y cuando no se afecten a las aplicaciones (excepto en el desempeño).

Tomando en cuenta la consideraciones antes mencionadas se entiende que la base de datos puede crecer sin afectar las aplicaciones existentes y por lo tanto permite que haya una independencia de datos.

2.4 Arquitectura de un sistema de bases de datos

No necesariamente todos los sistemas de bases de datos presentan una misma arquitectura, pero si sirve como punto de partida para el análisis de muchos sistemas. Sin embargo, el establecer una arquitectura sirve como marco de referencia para describir conceptos generales sobre bases de datos y para explicar la estructura de los sistemas individuales.

La arquitectura de divide en tres grandes niveles (ver Figura 2.2).

- Nivel externo (esquema interno)
- Nivel conceptual (esquema conceptual)
- Nivel interno (esquema interno)

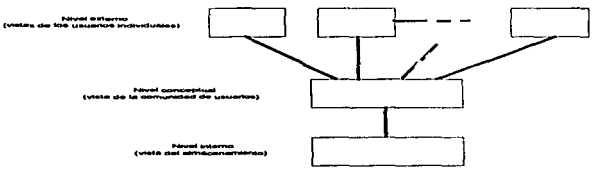


Fig. 2.2 Los tres niveles de arquitectura

El nivel interno es el que está más relacionado con el almacenamiento físico, es decir, con la forma en que los datos se almacenan. El nivel externo está estrechamente relacionado con el usuario, es decir, con la forma en que los datos son vistos por cada usuario.

Y por último el nivel conceptual es un nivel de mediación entre los otros dos. Define la vista global de los datos, es decir, si el nivel externo se relaciona con las vistas de los usuarios individuales, el nivel conceptual puede considerarse como el que define una vista de la comunidad de usuarios. En otras palabras, habrá muchas vistas externas, cada una compuesta por una representación más o menos abstracta de alguna parte de la base de datos, y habrá una sola vista conceptual, compuesta por una representación también abstracta de la base de datos en su totalidad. Asimismo, habrá una sola vista interna, que representa la base de datos total tal como está almacenado

Se ha dicho que un usuario individual normalmente sólo se interesará en alguna parte de la base de datos; además la vista que tiene el usuario en cuanto a los datos es abstracta cuando ésta es comparada con la manera que se encuentran físicamente almacenados los datos. La vista de un usuario individual en términos a ANSI/SPARC (American National Standars Institute/SPARC) se llama vista externa. Una vista externa se refiere al contenido de la base de datos tal como lo ve un usuario específico, en general, una vista externa se compone de múltiples ocurrencias de múltiples tipo de registros externos. Un registro externo no es necesariamente un registro almacenado.

Cada vista externa se define por medio de un esquema externo, el cual se compone esencialmente de las definiciones de cada uno de los diversos tipos de registros externos de esa vista externa.

La vista conceptual es una representación del contenido total de información de la base de datos, en general se pretende que la vista de los datos sea tal como lo es en realidad y no como los usuarios están obligados a verlos. La vista conceptual se compone de múltiples ocurrencias de múltiples tipos de registros conceptuales. A los registros conceptuales también se les conoce como entidades o como asociaciones.

La vista conceptual se define por medio del esquema conceptual, el cual incluye definiciones de cada uno de los distintos tipos de registros conceptuales. Por lo tanto una vista conceptual es una vista del contenido total de la base de datos, y el esquema conceptual es un definición de esa vista.

El tercer y último nivel de la arquitectura aquí descrita es el interno. La vista interna es la representación del nivel más bajo de la base de datos en su totalidad; se compone de múltiples ocurrencias de múltiples tipo de registros internos; sin embargo la vista interna se mantiene a un paso del nivel físico, ya que no afecta a registros físicos o bloques ni a ninguna restricción específica de dispositivos tales como capacidades de cilindros o pistas. La vista interna se describe por medio del esquema interno, el cual no solo define los diversos tipos de registros almacenados, sino también dice cuáles índices existen, de qué forma se representan los campos almacenados, en qué secuencia física se hallan los registros almacenados. Si vemos a la Figura 2.3 se observa que faltan tres cosas por analizar: el sistema de administración de bases de datos, el administrador de bases de datos y la interfaz con el usuario.

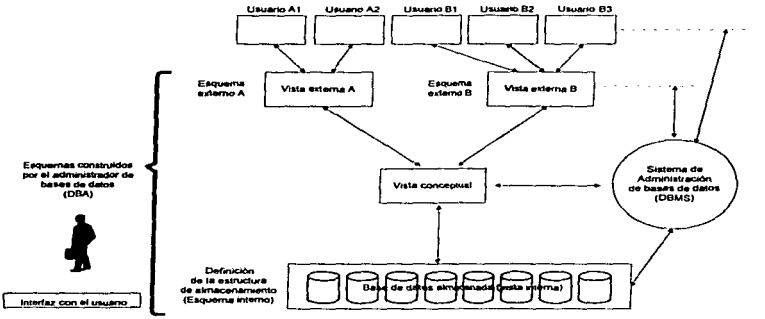


Fig. 2.3 Arquitectura del sistema de bases de datos

Un DBMS es el software que controla todos los accesos a la base de datos. Un DBA es el encargado del control general del sistema de bases de datos, se señalan algunas

responsabilidades del DBA a continuación:

1. Controla la estructura de la información de la base de datos.
2. Vincularse con los usuarios.
3. Definir los controles de autorización y los procedimientos de validación.
4. Definir una estrategia de respaldo y recuperación.
5. Controlar el desempeño y responder a los cambios de requerimientos.

Uno de los recursos más importantes del DBA es el diccionario de datos el cual es una herramienta que proporciona información uniforme y centralizada acerca de todos los recursos de datos. Un diccionario de datos es un almacenamiento centralizado de información acerca de las entidades, las interrelaciones entre las entidades, sus orígenes, usos y formatos de representación. Un diccionario de datos debe ayudar a un usuario de la base de datos en:

- La comunicación entre los usuarios.
- El control de los datos.
- La reducción de la redundancia y la inconsistencia.
- El diseño y ampliación de la base de datos.

El DBA con la ayuda del diccionario de datos tendrá facilidad para decir qué programas tienen probabilidad de ser afectados por algún cambio propuesto al sistema.

La interfaz con el usuario se puede definir como un límite del sistema por debajo del cual todo es transparente para el usuario y ésta se encuentra en el nivel externo.

2.5 Base de datos jerárquica

Una base de datos jerárquica utiliza estructuras de árboles donde cada nodo del árbol representa una entidad. El lugar en el que se encuentra una entidad nos puede decir que tipo de enlace tiene con otras entidades en el mismo árbol.

Cada nodo tiene elementos, los cuales representan un tipo de registro o un tipo de entidad. A continuación se explican la terminología típica que se utiliza en una estructura jerárquica.

- Raíz es un registro que no tiene padres.
- Hoja se les llama a todos aquellos que no tiene un subordinado, es decir, hijos.
- Padre e hijo para que exista un nodo hijo debe de tener un nodo padre conectado a él. Por tratarse de un árbol no puede haber más de un hijo con más de un padre.

Una base de datos jerárquica es identificada por las siguientes características:

1. Las entidades en una base de datos jerárquica se encuentran organizadas en estructuras de árbol.
2. Los distintos tipos de registros en un archivo jerárquico se encuentran enlazados por medio de relaciones de uno-a-muchos, pero las relaciones de muchos-a-muchos no se puede enlazar de una forma directa.
3. Cada nodo consta de uno o más datos.
4. Las ocurrencias de los padres pueden tener distinto número de ocurrencias de hijos.
5. Un registro hijo no puede existir si no existe un registro padre.
6. Si un registro padre se elimina también de tienen que eliminar los registros hijos.

Por ejemplo en la figura 2.4 representa una estructura jerárquica, HOSPITAL es el segmento padre de Laboratorio y Consultorio. El miembro Sala es el segmento padre de DOCTOR Y PACIENTE. Como se puede notar un segmento padre puede tener varios segmentos hijos. El segmento HOSPITAL así como los demás segmentos de la figura 2.4 representa entidades las cuales tienen su atributos.

Las desventajas principales de un sistema de base de datos jerárquica son:

1. No hay simplicidad en su representación en la estructura lógica de datos.
2. El manejo de las relaciones de muchos a muchos es muy compleja y se realiza de una forma indirecta.
3. Cada nodo no puede tener más de un padre.
4. La independencia de datos en mínima.
5. Puede haber inconsistencia de los datos ya que el contenido de un registro específico puede repetirse en varios lugares.
6. Por el punto anterior se presenta un desperdicio de espacio.

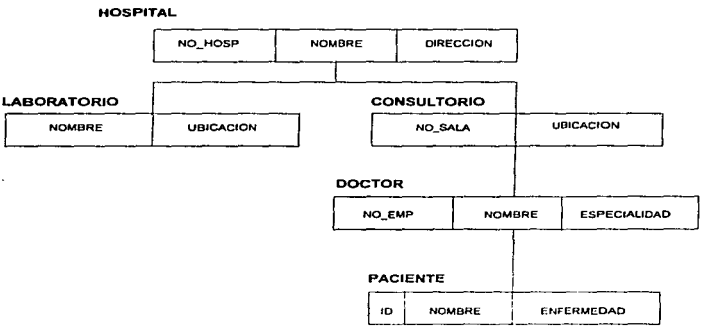


Fig. 2.4 Ejemplo de un esquema de una estructura jerárquica

2.6 Base de datos de red

Las principales características que identifican una base de datos de red.

1. La estructura de red permite que un segmento hijo pueda tener más de un segmento padre.
2. Puede representar con mayor facilidad una conexión entre dos entidades de muchos-a-muchos teniendo diversas alternativas para hacerlo.
3. En una estructura de red se presenta la situación de que dos entidades se conectan de muchos-a-muchos y esto provoca que no haya datos propios de conexión que no pertenece exclusivamente a una de las entidades. Para este tipo de situaciones la estructura de red utiliza un tipo de conexión entre las dos entidades.

Para representar el esquema de una estructura de red, se dibuja un grafo[1] hacia los nodos (entidades), las aristas son ligas o conectores y representan una conexión entre dos entidades. A las ligas se le asigna una etiqueta la cual nos dice que tipo de conexión existe mientras que la ligadura nos informa sobre la naturaleza de la conexión. Al final de la flecha se coloca el nodo padre y en la punta de la flecha el nodo hijo.

Un registro padre se le llama propietario y al registro hijo se denomina miembro. Un registro de tipo miembro puede pertenecer al mismo tiempo a dos tipos de conjuntos diferentes, a esto se le llama parentesco múltiple. Un tipo de registro puede ser miembro en un tipo de conjunto y al mismo tiempo puede ser propietario en otro, por lo que pueden existir niveles múltiples de jerarquía.

En la figura 2.5 muestra un esquema de una estructura de red. y en la figura 2.6 representa relaciones de muchos-a-muchos en una estructura de red.

Las desventajas que presenta un sistema de base de datos de red es que:

- No hay simplicidad en su representación de estructura lógica de datos.
- No existe una buena comunicación entre en DBA y los usuarios finales por el tipo de estructura.
- Una liga es una asociación entre dos registros exclusivamente. Así pues se puede considerar como una forma restringida (binaria) de relación en el sentido del modelo Entidad-Relación.

[1] Un grafo G consta de dos conjuntos V y A. V es un conjunto finito de vértices no vacíos. A es un conjunto de pares de vértices denominados aristas. No habrá más de una arista uniendo un par de vértices.

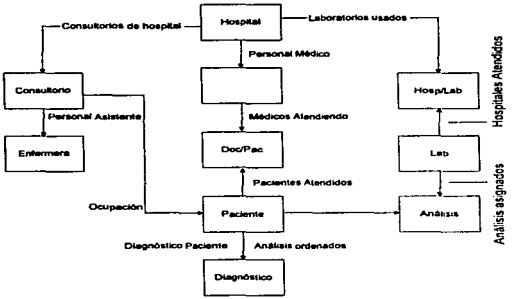


Fig. 2.5 Esquema de una estrutura de red

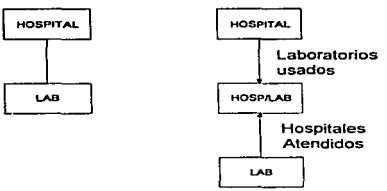


Figura. 3.6 Manejo de relaciones muchoas-a-muchos en una estructura de red

2.7 Base de datos relacional

Una base de datos relacional esta formada por una estructuras lógica que representa relaciones. Todas las entidades en una base de datos relacional están almacenadas de manera separada y no tienen ninguna jerarquía fija como en el caso de la estructura jerárquica o la estructura de red.

En el esquema relacional existe una independencia de datos ya que en vez de utilizar señaladores para conectar registro relacionados con otros archivos como sucede en la base de datos jerárquica o en la base de datos de red se usan campos de conexión, es decir, en vez de utilizar señaladores para tener relaciones uno-a-muchos en un registro padre en un archivo y sus registros hijos en otro, el sistema relacional maneja una llave o clave del registro en el campo de conexión correspondiente a todos sus registros con los que está ligado.

El álgebra relacional se convierte en un lenguaje de consulta en donde el usuario le ordena al sistema que realice una serie de operaciones con la base de datos para obtener el resultado deseado. Para que el usuario solicite información de la base de datos utiliza el álgebra relacional, la cual ya se revisó en el capítulo 1. Sin embargo, por su relevancia en lo que resta del trabajo aquí se presentan algunos ejemplos pero se mencionarán las ventajas y desventajas que tiene una base de datos.

Las ventajas de un sistema de base de datos relacionales son:

1. Simplicidad en su representación en la estructura lógica de datos.
2. Flexibilidad para establecer relaciones de datos por medio de campos de conexión.
3. Mayor entendimiento entre el DBA los usuarios finales.
4. Hay una mayor independencia de datos.
5. Se puede accesar directamente y a cada una de las relaciones sin necesidad de abrir las demás.
6. La recuperación de registros puede ser de un conjunto (y no de un sólo registro) por medio de una sola proposición DML.

La principal desventaja que presenta un sistema de base de datos relacionales es que el software que controla las operaciones de acceso (DBMS) no es sólo difícil de desarrollar para que cumpla las metas deseadas, sino que también es costoso y el ejecutarlo toma algo de tiempo.

Permutación

En las tablas de abajo se ve un ejemplo de permutación. La primera es la relación original y la segunda es el resultado de aplicar la permutación π 1,3,2,4,5 (EMPLEADO).

Entidad EMPLEADO

RUMT120565	T. Ruiz M	Cirujano	20,000	2
LOPM121268	M. López	Residente	20,000	5
AEGS031270	S. Acevedo	Cirujano	7,500	7
BELG061169	G. Beltrán	Instrumentista	6,700	1
GOHJ050870	J. Gómez	Residente	10,000	5

Permutación de la entidad EMPLEADO

RUMT120565	Cirujano	T. Ruiz M	20,000	2
LOPM121268	Residente	M. López	20,000	5
AEGS031270	Cirujano	S. Acevedo	7,500	7
BELG061169	Instrumentista	G. Beltrán	6,700	1
GOHJ050870	Residente	J. Gómez	10,000	5

Proyección

Sea la relación EMPLEADO $\subset$ D_{RFC} X D_{NOM_EMP} X $D_{PROFESION}$ X D_{SUELDO} X D_{DEPTO} y utilizando los datos de la entidad EMPLEADO.

Entonces la $\pi_{1,4,5}$ (EMPLEADO) será la relación S = {(RUMT120565, 20,000,2), (LOPM121268, 20,000, 5), (AEGS031270, 7,500, 7), (BELG061169, 6,700,1), (GOHJ050870, 10,000,5)

La tabla siguiente muestra el resultado al aplicar $\pi_{1,4,5}$ (EMPLEADO)

RUMT120565	20,000	2
LOPM121268	20,000	5
AEGS031270	7,500	7
BELG061169	6,700	1
GOHJ050870	10,000	5

Restricción

Utilicemos la entidad EMPLEADO y una entidad denominada PROF-SDO que contiene información de profesión y sueldo:

EMPLEADOS $\subset D_{RFC} \times D_{NOM\_EMP} \times D_{PROFESION} \times D_{SUELDO} \times D_{DEPTO}$, utilizando los datos de la entidad EMPLEADO.

PROF-SDO $\subset D_{PROFESION} \times D_{SUELDO}$

PROF-SDO = {(Residente, 20,000), (Cirujano, 7,500,), (Residente, 10,000)}

Sea L=3, 4 y M=1, 2. La restricción EMPLEADOL | MPROF-SDO = {(LOPM121268, M. López, Residente, 20,000, 5), (AEGS031270 , S. Acevedo, Cirujano, 7,500, 7), (GOHJ050870, J. Gómez, Residente, 10,000, 5)}

Podemos ver que se cumple: $\pi_{3,4}(EMPLEADO_{3,4} \mid PROF\text{-}SDO_{1,2}) = \pi_{1,2}(PROF\text{-}SDO)$, para las tuplas que cumplen con que $\pi_3(EMPLEADO) = \pi_1(PROF\text{-}SDO)$ y $\pi_4(EMPLEADO) = \pi_2(PROF\text{-}DO)$.

En la siguiente tabla se muestra el resultado de $EMPLEADO_{3,4} \mid PROF\text{-}SDO_{1,2}$

EMPLEADO.

LOPM121268	M. López	Residente	20,000	5
AEGS031270	S. Acevedo	Cirujano	7,500	7
GOHJ050870	J. Gómez	Residente	10,000	5

Join

Esta es una operación binaria que combina dos relaciones usando sus dominios comunes. Informalmente, el join entre dos relacionales puede existir sobre cualesquiera de los atributos de la entidad que tenga las mismas características. De hecho, sólo debería existir sobre esos conjuntos.

Sea la entidad EMPLEADO mostrada anteriormente y entidad DEPARTAMENTO que se encuentra en la siguiente tabla:

ENTIDAD DEPARTAMENTO

1	Medicina General	Edif. 3
2	Cardiología	Edif. 2
5	Neurología	Edif. 2
7	Administración	Edif. 1

Obtener una lista de los empleados con el departamento en que trabajan. La relación resultante llamada EMP_NOM_DEPTO representa el resultado de la operación join que fue realizada por los atributos comunes de las dos relaciones, que como se observa fue por medio del atributo DEPTO.

El resultado EMPLEADO⋈ DEPARTAMENTO se muestra en la tabla siguiente:

EMP_NOM_DEPTO

RUMT120565	T. Ruiz M	Cirujano	20,000	2	Cardiología	Edif. 2
LOPM121268	M. López	Residente	20,000	5	Neurología	Edif. 2
AEGS031270	S. Acevedo	Cirujano	7,500	7	Administración	Edif. 1
BELG061169	G. Beltrán	Instrumentista	6,700	1	Medicina General	Edif. 3
GOHJ050870	J. Gómez	Residente	10,000	5	Neurología	Edif. 2.

Composición

Es una operación sobre dos relaciones, se aplica a la relación resultante de un join y su efecto es eliminar el atributo usado para realizar la operación join. En el caso anterior sería el atributo DEPTO.

Sean R y S dos relaciones de grados n, m respectivamente, T es una composición de R con S si: $\exists\ u = R * S$ tal que $T = \pi_L (u)$. L es una lista de índices que incluye a todos los dominios de u exceptuando el dominio de DEPTO.

Utilizando el ejemplo del operador JOIN se tiene que la relación : EMP_NOM_DEPTO = EMPLEADO*DEPARTAMENTO; entonces T=1,2,3,4,6,7 (EMP_NOMBRE_DEPTO). El resultado de la operación de muestra en la tabla que se encuentra abajo.

RUMT120565	T. Ruiz M	Cirujano	20,000	Cardiología	Edif. 2
LOPM121268	M. López	Residente	20,000	Neurología	Edif. 2
AEGS031270	S. Acevedo	Cirujano	7,500	Administración	Edif. 1
BELG061169	G. Beltrán	Instrumentista	6,700	Medicina General	Edif. 3
GOIU050870	J. Gómez	Residente	10,000	Neurología	Edif. 2.

2.8 Base de datos distribuidas

En un sistema de base de datos distribuido, los datos se almacenan en distintas computadoras. Las computadoras de un sistema distribuido se comunican entre sí a través de diversos medios de comunicación.

Un sistema distribuido de base de datos consiste en un conjunto de localidades[2], cada una de las cuales puede participar en la ejecución de transacciones que accesen datos de una o más localidades. A diferencia del distribuido, en un sistema centralizado los datos se encuentran en una sola localidad.

En un sistema distribuido de base de datos cada localidad posee un sistema de base de datos. Cada localidad puede hacer dos tipos de transacciones:

Es transacción local aquellas que sólo accesan información que reside en esa localidad y es transacción global aquellas que accesan información de varias localidades. En este tipo de transacción requiere de medios de comunicación entre las localidades.

Las localidades del sistema pueden conectarse físicamente de diferentes maneras. Las distintas configuraciones se representan por medio de gráficas y cada nodo corresponde una localidad. Una arista del nodo A al nodo B corresponde a una conexión directa entre las dos localidades. En la Figura 2.7 se muestra las configuraciones más comunes que se manejan. Las diferencias principales entre estas configuraciones son:

- **Costo de instalación:** costo de conectar físicamente las localidades del sistema.
- **Costo de comunicaciones:** costo en tiempo y dinero para enviar un mensaje de la localidad A a la B.
- **Confiabilidad:** la frecuencia con que falla una línea de comunicación o una localidad.
- **Disponibilidad:** La posibilidad de accesar la información a pesar de fallas en algunas localidades o líneas de comunicación.

Existen varias razones para construir sistemas distribuidos de base de datos, algunas de esas razones son: compartir la información, mejorar la confiabilidad y la disponibilidad, y agilizar el procesamiento de las consultas.

---

[2] Los procesadores de un sistema distribuido incluyen computadoras pequeñas, estaciones de trabajo, y sistema de cómputo grandes de aplicación general, estos procesadores reciben diferentes nombres: localidades, nodos, computadoras, etc., dependiendo del contexto en el que se mencionen. Aquí se utilizará normalmente el término localidad, para hacer hincapié en la distribución física de estos sistemas.

Las principales ventajas que presenta un sistema distribuido son

1. Como varias localidades están conectadas entre sí, un usuario puede accesar a los datos que tienen las distintas localidades.

2. Cada localidad puede controlar hasta cierto punto los datos almacenados localmente.

3. Si por algún motivo se presenta una falla en cualquiera de las localidades, es posible que las demás sigan funcionando sin ningún problema.

4. En el momento que se detecta que una localidad falló entonces el sistema no utilizará esa localidad hasta que haya sido reparada.

5. A pesar de que la recuperación de fallas sea más compleja en los sistemas distribuidos que en los centralizados, la capacidad con que cuenta el sistema para seguir funcionando es mucho más disponible.

6. Si una consulta consta de datos de varias localidades, es posible dividir la consulta en varias subconsultas que se ejecuten en paralelo en distintas localidades. Este cálculo en paralelo permite procesar con rapidez la consulta del usuario.

La principal desventaja que hay en los sistemas distribuidos de base de datos es el aumento en la complejidad que se refleja en:

1. **Costo de desarrollo de Software.**
2. **Mayor posibilidad de errores:** como las localidades del sistema distribuido operan en paralelo, es más difícil garantizar que los algoritmos sean correctos. El arte de construir algoritmos para sistemas distribuidos sigue siendo un campo de investigación activo e importante.

3. **Mayor tiempo extra de procesamiento:** el intercambio de mensajes y los cálculos adicionales que se requieren para coordinar las localidades son una forma de tiempo extra que no existe en los sistemas centralizados.

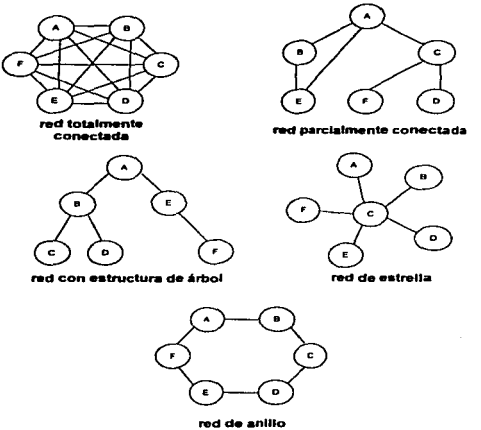


Fig. 2.7 Topología de las redes

3. BASES DE DATOS DISTRIBUIDAS

Un manejador de bases de datos distribuidas es el software que permite administrar el sistema DDBS y permite que sea transparente para los usuarios. Este capítulo se dedicará a los factores de diseño que se aplican a las bases de datos distribuidas.

3.1 Diseño de bases de datos distribuidas

Para almacenar una relación ***R*** en una base de datos distruibuida se toman en cuenta los siguientes factores:

- **Replicación:** el sistema guarda varias copias iguales de la relación y cada copia se almacena en una localidad diferente, y el resultado es una repetición de la información.

- **Fragmentación:** la relación ***R*** se divide en varios fragmentos y cada fragmento se almacena en una localidad diferente.

- **Replicación y fragmentación:** es la combinación de los dos conceptos anteriores. La relación se divide en varios fragmentos, el sistema realiza copias iguales de cada uno de los fragmentos y los almacena en localidades diferentes.

3.2 Replicación de los datos

Si la relación ***R*** está repetida, la copia se almacena en una o más localidades. Hay ocasiones que se presenta la repetición total, es decir, se almacena la copia de la relación en todas las localidades del sistema.

La replicación tiene ventajas y desventajas:

- **Disponibilidad:** si falla una de las localidades que tiene la relación ***R***, no importa, porque se puede disponer de otra localidad que también contenga ***R***, de esta manera el sistema puede seguir operando y procesando consultas que tengan que ver con ***R***.

- **Mayor paralelismo:** si la mayoría de las veces que se accesa a la relación ***R*** son solo lecturas, varias localidades podrán procesar consultas que involucren a ***R*** en paralelo. Mientra más copias de ***R*** existan mayor será la probabilidad de que los datos se encuentren en las localidades en donde se realice la transacción y esto permitirá que se reduzca al mínimo el movimiento de información entre localidades.

- **Mayor tiempo extra para actualizaciones:** el sistema debe asegurarse de que las copias de la relación ***R*** sean consistentes ya que si no hay consistencia puede provocar errores. Cada que ***R*** es actualizada o modificada, es necesario actualizar todas las copias de ***R*** que se encuentren en diferentes localidades y esto requiere de un tiempo extra.

3.3 Transparencias en un DBMS distribuido

La transparencia en una DBMS distribuido se refiere a la separación de el nivel más alto de la semántica de un sistema del nivel más bajo de la implementación. En otras palabras un sistema transparente oculta los detalles de implementación a los usuarios. La ventaja de un DBMS completamente transparente es el alto nivel de soporte que éste proporciona al desarrollo de aplicaciones complejas.

Transparencia de Red

En un sistema de base de datos centralizado el único recurso disponible que necesita ser protegido del usuario es el dato. En un ambiente de administración de bases de datos distribuidas, el usuario debe ser protegido de los detalles operacionales de la red y si es posible ocultar la existencia y no debe darse cuenta si las aplicaciones de la base de datos corre bajo un sistema de bases centralizada o un sistema de base datos distribuida. Este tipo de transparencia se conoce como transparencia de red o transparencia de distribución.

La transparencia de red se puede considerar ya sea por los servicios proporcionados o los datos proporcionados. Para esto es necesario tener medios uniformes por los cuales los servicios son accesados.

Existen sistemas operativos de red que no cumplen con la transparencia de red, un ejemplo sería el comando de UNIX para realizar la copia de un archivo ya sea en la misma máquina o a través de dos máquinas conectadas. Los comandos para cualquiera de los casos es diferente y esto hace que no se tenga una transparencia de red pues para copiar un archivo a otra máquina es necesario especificar el nombre de la máquina de origen y el de la máquina fuente. Es decir, se ocupan dos comandos diferentes para realizar una misma acción.

El ejemplo anterior demuestra que se necesitan otros tipos de transparencia en una base de datos distribuida, la transparencia de locación y la transparencia de nombramiento. Transparencia de locación se refiere al hecho de que los comandos usados es independiente de ambos, de la locación de los datos y del sistema en el cual una operación se lleva a cabo y Transparencia de nombramiento se refiere a que debe de existir un nombre único para cada objeto en la base de datos.

Transparencia de replicación

La replicación puede ayudar al rendimiento general del sistema porque algunos requerimientos conflictivos del usuario pueden ser resueltos fácilmente. Por ejemplo, los datos que son accesados comunmente por un usuario pueden ser situados en la máquina local de dicho usuario, así como también en la máquina de otro usuario que tenga los mismos requerimientos de acceso.

La decisión de que si hay replicación o no, y de cuántas copias se deben de tener para cada objeto depende en gran parte en las aplicaciones que se usen; pues deben hacerse

solo las copias necesarias y no más, ya que la replicación causa serios problemas de actualización.

El punto relativo a la transparencia que debemos definir es si los usuarios deberían estar concientes de las existencias de copias o si el sistema debería manipular la administración de copias y el usuario debiera actuar como si hubiera una sola copia de los datos. Desde el punto de vista de un usuario, la respuesta es obvia. Es preferible no estar involucrado con el control de copias ni tener que tratar con el hecho de que una cierta acción puede o debe ser tomada sobre múltiples copias. Desde el punto de vista de un sistema, la respuesta no es tan simple. Cuando delegamos a un usuario la responsabilidad de especificar que una acción necesita ser ejecutada sobre múltiples copias, la administración de las transacciones se hace más simple para las DBMS's distribuidas.

Cuando se solicita un dato, no es necesario especificar la copia. El sistema utliza una entidad para determinar cuáles son todas las copias de ese dato. Además, el sistema debe ser responsable de determinar qué copia debe de accesarse para su lectura y actualizar todas las copias cuando se solicite su modificación.

Transparencia de Fragmentación

La transparencia de fragmentación se refiere a la división de cada relación de la base de datos en pequeños fragmentos y cada fragmento de la base de datos es tratado como un objeto separado. Esto es para lograr un mejor desempeño, disponibilidad e integridad. Además, la fragmentación puede reducir los aspectos negativos de la replicación. Cada replica no es la relación completa sino solo un subconjunto de ella; por ello se requiere menos espacio y un número menor de datos necesitan ser manejados.

Cuando los objetos de la base de datos son fragmentados, se tiene que hablar sobre los problemas de manejo de consultas de usuario ya que antes de ser fragmentados eran especificados en relaciones enteras y ahora se presentarán en subrelaciones. Esto requiere una conversión de una "consulta global" a varias "consultas fragmentadas".

De la misma manera que la repetición, los usuarios no deben darse cuenta de cómo está fragmentado una relación. Por lo tanto un sistema de bases de datos distribuidas debe permitir que las consultas se hagan en términos de los elementos de información sin fragmentar.

Transparencia y autonomía

Es fundamental que el sistema evite al máximo que el usuario vea cómo está almacenada una relación. Un sistema puede ocultar los detalles de la distribución de la información en la red y a esto se le llama *transparencia de la red.*

La transparencia de las redes está estrechamente relacionada con la autonomía local. Esto se dice porque la transparencia de una red es el grado en el que los usuarios del sistema ignoran los detalles del diseño del sistema distribuido y la autonomía local es el grado en el cual el administrador de una localidad es independiente del sistema distribuido.

La transparencia y la autonomía se consideran según los siguientes puntos de vista:

- Nombre de los datos.
- Repetición de los datos.
- Fragmentación de los datos.
- Localización de fragmentos y copias.

Asignación de nombres y autonomía local

Cualquier elemento de información debe tener un nombre único. La bases de datos distribuidas deben asegurarse de que todas las localidades utilicen distintos nombres para dos datos diferentes.

Puede usarse un asignador de nombres para evitar que se duplique un mismo nombre, sin embargo esto trae varias desventajas que se darán a continuación:

- Es muy posible que el asignador se sature y por lo tanto se convierta en un cuello de botella.

- Si el asignador falla es muy posible que todas las localidades del sistema distribuido no puedan seguir funcionando.

- Se presenta una reducción de autonomía local, ya que la asignación de los nombres es controlada de una manera centralizada.

Para evitar que se reduzca la autonomía local puede establecerse que cada que se genere un nombre en una localidad se asigne un prefijo a cada nombre generado y esto

asegura que nunca se generará un nombre igual puesto que cada localidad tiene su propio prefijo y además esto evita que se requiera un control central.

La desventaja de lo antes dicho es que no hay transparencia de la red pues son asignados identificadores de localidad a los nombres.

Es de importancia que el sistema identifique qué copias son copias de la misma relación y cuáles fragmentos son fragmentos de la misma relación y en ocasiones se opta por agregrar "f_1", "f_2", ..., "f_n" a los fragmentos de un elemento de información y "r_1", "r_2", ..., "r_n" a las copias.

Transparencia de localización

Como el sistema es transparente en cuanto a repetición y fragmentación, la planeación de la bases de datos distribuidas permanece en gran parte oculta al usuario. Sin embargo el componente de los nombres que identifica a cada localidad revela que el sistema es distribuido.

La transparencia de localización se realiza creando un conjunto de seudónimos o alias para cada usuario, de tal manera que no será necesario que el usuario conozca la localización física de un dato y además el administrador podrá cambiar un dato de una localidad a otra sin afectar al usuario.

¿Quién debe proporcionar transparencia ?

Para proporcionar un eficiente y fácil acceso a usuarios novatos a los servicios de el DBMS, uno debe tener una completa transparencia y éste es un compromiso inevitable que proporciona una comodidad de uso pero también un alto costo.

Es posible identificar tres tipos diferentes de capas que les pueden ser proporcionados a los servicios de transparencia:

1. Podemos dejar la responsabilidad de proporcionar acceso transparente de recursos de datos a la capa de acceso. Las características de transparencia pueden ser construidas en un lenguaje de usuario, el cual traduce los servicios requeridos en las operaciones requeridas. En otras palabras, el compilador o el interpretador toma la tarea y el servicio no transparente y es dado al implementador de el compilador o de el interpretador.

2. La segunda capa a la cual puede ser proporcionada transparencia es a nivel de sistema operativo. Los sistemas operativos "State-of-the-art" proporcionan algún nivel de transparencia para usuarios de sistema. Proporcionar acceso transparente a recursos a nivel sistema operativo puede ser extendido al ambiente distribuido donde la administración de los recursos de la red se toman sobre el sistema operativo distribuido. Este es un buen nivel en el que se proporciona transparencia de red, claro si éste puede ser llevado a cabo. Pero desafortunadamente no todos los sistemas operativos distribuidos disponibles en el mercado pueden proporcionar un nivel razonable de transparencia en la admnistración de red.

3. La tercera capa que puede proporcionar transparencia es en el DBMS. En las máquina de bases de datos, por ejemplo el DBMS generalmente no espera ningún servicio transparente por parte del sistema operativo. Un ambiente más común es el desarrollo de un DBMS en una computadora de propósito general que corre algunos sistemas operativos. En este tipo de ambiente, la transparencia y soporte para funciones de bases de datos proporcionadas a los diseñadores de DBMSs es mínima y está limitada a operaciones muy fundamentales para llevar a cabo ciertas tareas. Es la responsabilidad de un DBMS hacer todas las traducciones necesarias del sistema operativo a la interface de usuario de más alto nivel. Este modo de operación es el método más común de hoy en día. Hay, sin embargo, varios problemas asociados con dejar la tareas de proporcionar completa transparencia al DBMS.

La transparencia de la red puede ser manejada fácilmente por el sistema operativo distribuido para proporcionar transparencias de replicación y fragmentación. El DBMS debe ser responsable para dar un alto nivel de independencia de datos junto con las transparencias de fragmentación y replicación.

Una jerarquía de estas transparencias es mostrada en la Figura 3.1. No es fácil delinear claramente los niveles de transparencia pero esta figura cumple un propósito importante instruccional aunque no sea totalmente correcta. Para completar la figura se ha agregado una capa de transparencia de lenguaje "transparencia de lenguaje". Con esta capa los usuarios tienen acceso de alto nivel a los datos.

Fig. 3.1 Capas de transparencia

3.4 Estandarización del DBMS

El modelo de referencia de un DDBMS puede ser pensado como un modelo de arquitectura idealizado del sistema. Esto es definido como una marco conceptual cuyo propósito es dividir trabajo de estandarización en piezas manejables, y mostrar a nivel general como estas piezas están relacionadas unas con otras.

Un modelo de referencia puede ser descrito en tres diferentes puntos:

1. **Basado en componentes:** los componentes del sistema son definidos junto con la interrelación entre componentes. De este modo un DBMS consiste en un número de componentes que proporcionan alguna funcionalidad. Su ordenamiento e interacción bien definida proporciona una total funcionalidad al sistema.

2. **Basado en funciones:** Las diferentes clases de usuarios son identificadas y las funciones que se llevan a cabo para cada clase son definidas. Las especificaciones de esta categoría especifican una estructura jerárquica para las clases de usuarios. Esto resulta de la arquitectura de un sistema jerárquico con interfaces bien definidas entre las funcionalidades de diferentes capas.

3. **Basado en datos:** Los diferentes tipos de datos son identificados, y un marco estructurado es especificado y este define las unidades funcionales que comprenden datos, de acuerdo a sus vistas diferentes. Porque los datos es el elemento central que un DBMS maneja.

Esto tres puntos nunca deben perderse de vista, deben tomarse siempre encuenta y los tres deben tener interacción conjuntamente[1].

[1] Como indica un reporte de la "Database Architecture Framework Task Group of ANSI [DAFTG, 1986]," todos los tres puntos necesitan ser usados juntos para definir un modelo de arquitectura.

3.5 Modelo de arquitectura para DBMSs distribuido

Existen diferentes maneras en las que se puede conectar múltiples bases de datos para que sean compartidas por múltiples DBMS's. En la Figura 3.2 muestra la clasificación de los sistemas en base a tres características ortogonales entre sí. Las tres características son: la autonomía de los sistemas locales, la distribución de los datos y heterogeneidad.

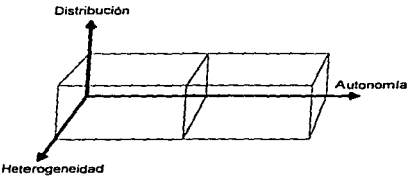


Fig. 3.2 Alternativas de implantación de DBMS

La **autonomía de los sistemas locales** se refiere a la distribución de control, no de los datos, indicando el grado en el que una base de datos individual puede operar independientemente. Autonomía es una función de un número de factores tales como:

- Si los sistemas componentes intercambiaran información.
- Si pueden ejecutar independientemente transacciones y
- Si uno esta permitido a modificarlos.

2. **Autonomía de comunicación:** Cada DBMSs individual es libre de decidir que tipo de información puede proporcionar al otro DBMSs o el software que controla su ejecución global.

Se consideran tres tipos de sistemas:

- **Integración fuerte:** donde una sola imagen de entrada de la base de datos es disponible para cualquier usuario que quiera compartir información la cuál debe residir en muchas bases de datos. Para el usuario los datos se encuentran lógicamente almacenados en una base de datos centralizada.

- **Sistema semiautónomo:** consiste en que un DBMS puede operar de manera independiente pero también puede decidir si puede participar en una federación para permitir compartición de datos locales. Cada DBMSs determina que parte de su base de datos pueden ser accesadas por otros usuarios que pertenezca a otro DBMSs. Estos no son completamente autónomos porque necesitan ser configurados para que permitan intercambio de información con los demás.

- **Aislamiento total,** son los DBMSs individuales que permanecen solos, es decir, no conocen la existencia de otro DBMS ni mucho menos el como comunicarse.

La **distribución de los datos** se puede decir que se dividen en dos casos, los que están físicamente distribuidos en múltiples sitios que se pueden comunicar con cualquier otro a través de un medio de comunicación y los que se encuentran en un solo sitio.

La **heterogeneidad** puede ocurrir en diferentes formas en un sistema distribuido, algunas de las más importantes son: el modelo de datos, el lenguaje de consulta y los protocolos para el manejo de transacciones. Sin embargo podemos en general de hablar de dos tipos de sistemas:

- Sistemas homogeneos
- Sistemas heterogeneos

3.6 Arquitectura de un DBMS distribuido

Empezaremos la descripción de la arquitectura de un DBMS distribuido por la vista organizacional de los datos. Primero haremos notar que la organización física de los datos de cada máquina probablemente sean diferentes. Esto significa que se necesita hacer una definición de esquema interno individual de cada sitio, los cuales llamaremos, *esquema local interno (LIS)*. La vista de la empresa de los datos es descrita por el *esquema conceptual* global *(GCS)*, la cual se dice global porque describe la estructura lógica de los datos en todos los sitios.

Vimos en temas anteriores que los datos en bases de datos distribuidas presenta fenómenos como fragmentación y replicación. Por lo que la organización lógica de datos de cada sitio necesita ser descrita.

Por todo lo anterior es necesario crear una tercera capa en la arquitectura , *el esquema conceptual local (LCS)*. En el modelo de arquitectura entonces escogeremos el esquema conceptual global que es la unión del esquema conceptual local. Finalmente, aplicaciones de usuario y acceso de usuario a la base de datos es soportada por el *esquema externo* (Ess).

El modelo de arquitectura definido en la Figura 3.3 proporciona los niveles de transparencia discutidos anteriormente. La independencia de datos es soportada porque el modelo es una ampliación de ANSI/SPARC, la cual proporciona naturalmente independencia.

Las transparencias de replicación y locación son soportadas por la definición del esquema conceptual global y local y mapeo entre ellos. La transparencia de red es por lo tanto, soportada por la definición del esquema conceptual global. Los datos de consulta de usuario no consideran su locación o que componente local del sistema de bases de datos distribuidas van a servirlos. Como mencionamos antes, los DBMS distribuidos traducen consultas globales dentro de un grupo de consultas locales las cuales son ejecutadas por componentes en diferentes sitios que se comunican uno con otro.

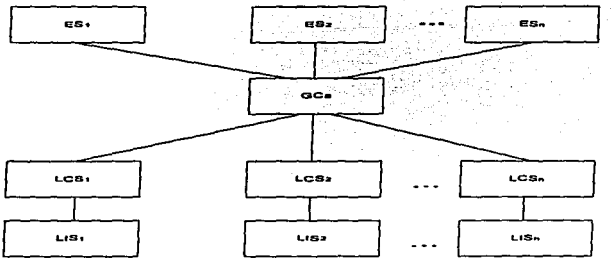


Fig. 3.3 Arquitectura de Referencia de Base de Datos Distribuida

En términos de la descripción funcional detallada de nuestro modelo, el modelo ANSI/SPARC es ampliado para agregar un directorio/directorio global (GD/D) que permite los mapeos globales requeridos. Los mapeos globales son aún llevados a cabo por un directorio/directorio local (LD/D).

De este modo los componentes de administración de base de datos son integrados por medio de funciones de DBMS globales. (Ver Figura 3.4). Como podemos observar en la Figura 3.4, los esquemas conceptuales locales son mapeos del esquema global dentro de cada sitio. Además, tales sistemas son diseñados típicamente en un modelo arriba-abajo, y por lo tanto, todas las definiciones de vista externa son hechas globalmente. También se describe, en la Figura 3.4 un administrador de base de datos local de cada sitio. La existencia de cada papel debe ser discutida. Sin embargo, se recuerda que una de las motivaciones primarias de procesamiento distribuido es el deseo de tener un control local sobre la administración de datos.

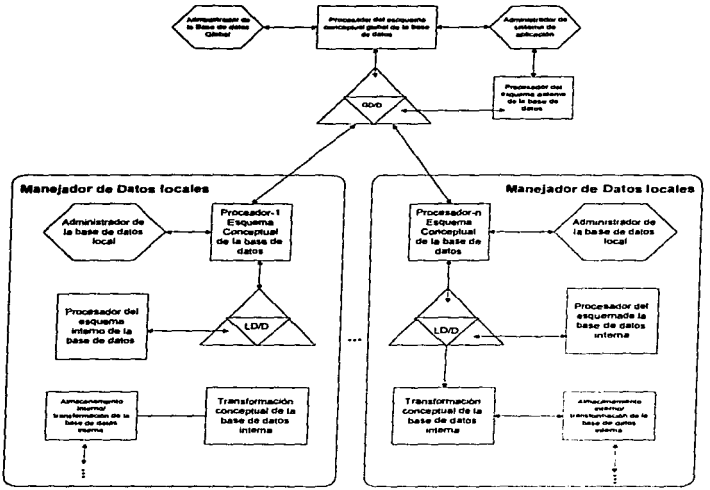


Fig. 3.4 Esquema Funcional de un DBMS Distribuido

Los componentes detallados de un DBMS distribuido son mostrados en la Figura 3.5 Uno de los componentes controla la interacción con usuarios, y otro habla de su almacenamiento. El primer componente, lo llamamos ***procesador de usuario***, y consiste en cuatro elementos:

1. **El control de interface del usuario** es responsable de interpretar comandos del usuario como ellos vayan llegando, y de formatear los datos de resultado como se vayan mandando al usuario.

2. **El control de datos semánticos** usa las normas de integridad y autorización que son definidas como parte del esquema global para confirmar si la consulta del usuario puede ser procesada.

3. **El Optimizador de consulta global y fragmentador** determina una estrategia de ejecución para minimizar una función de costo, traduciendo consultas globales en consultas locales. El optimizador global es responsable, entre otras cosas, para generar la mejor estrategia de ejecución de operación join distribuida.

4. **El Monitor de ejecución distribuida** coordina la ejecución distribuida de los requerimientos del usuario. El monitor de ejecución es llamado también "Manejador de transacciones distribuidas". Las consultas ejecutadas de manera distribuida y los monitores de ejecución se encuentran en varios sitios y esto usualmente hacen comunicación con algún otro.

El segundo componente de bases de datos distribuidas es el ***procesador de datos*** y consiste en tres elementos:

1. **El optimizador de consulta local** actúa como *el "selector de caminos de acceso"*. Es responsable para elegir el mejor a camino de acceso para entrar a cualquier objeto de datos. El término *"selector de caminos de acceso"* se refiere a las estructuras de datos y a los algoritmos que son usados para accesar los datos. Un típico *selector de caminos de acceso*, por ejemplo, es un índice en uno o más atributos de una relación.

2. **El manejador de recuperación local** es el responsable de asegurarse que la base de datos permanezca consistente cuando ocurren fallas en el sistema.

3. **El procesador de soporte de tiempo-ejecución** accesa físicamente a la base de datos conforme a los comandos físicos en el esquema generado por la consulta del optimizador. El procesador de soporte de tiempo-ejecución es la interface para el sistema operativo y contiene el controlador del buffer de la base de datos, el cual es responsable de mantener los buffers de memoria principal y controlar los accesos a los datos.

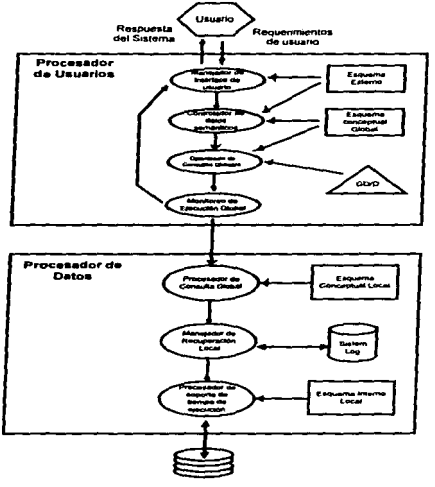


Fig. 3.5 Componentes de un DBMS distribuido

3.7 Diccionario de Datos

La discusión del diccionario global es concerniente solo si uno habla sobre un DBMS distribuido o un multi-DBMS que usa un esquema conceptual global. De lo contrario no hay un concepto de lo que es un diccionario global. Como se indicó antes, el diccionario es en sí una base de datos que contiene meta-datos sobre los datos acumulados actuales en la base de datos.

Un diccionario puede ser global a la entrada de base de datos o local a cada sitio. En otras palabras, puede haber un único diccionario que contenga información sobre todos los datos de la base de datos, o un conjunto de diccionarios, cada uno conteniendo la información almacenada en cada sitio. En este último caso, se construye una jerarquía de diccionarios para facilitar la búsqueda, o se debe implementar una estrategia de búsqueda distribuida que abarca una considerable comunicación entre los sitios que contiene los diccionarios.

El segundo punto tiene que ver con *asignación*. El diccionario debe mantenerse centralmente en un sitio, o en una estructura distribuida para distribuirlo sobre un número de sitios.

Guardando el diccionario en un sitio se incrementará la carga de ese sitio, y por lo tanto causará un embotellamiento además de incrementar mensajes de tráfico alrededor del sitio. Distribuirlo sobre un número de sitios, por otro lado, incrementa la complejidad por el control de diccionarios.

El último punto es la replicación. Puede haber una sola copia del diccionario o múltiples copias que proporcionan más integridad y un menor retraso al accesar el diccionario. Por otro lado, actualizar el diccionario es más difícil, porque para que el diccionario sea consistente deben actualizarse todas las copias existentes. Por lo tanto, la selección debe de depender del ambiente en el cual opera el sistema y debe de tener equilibrio con los factores de requerimientos de tiempo-respuesta, el tamaño del diccionario y la capacidad de la máquina en cada lugar.

3.8 Diseño de bases de datos distribuidas

El diseño de un sistema distribuido implica hacer decisiones de la colocación de datos y programas en las localidades que conforman el sistema. En el caso de un DBMSs distribuido, la distribución de aplicaciones implica dos cosas, la distribución del software DBMS distribuido y la distribución de los programas de aplicación que corren en él. El primero no es un problema significante, porque nosotros tomamos que una copia del software DBMS distribuido existe en cada sitio donde los datos están almacenados.

Se ha sugerido que la organización de sistemas distribuidos puede ser investigada a lo largo de tres dimensiones ortogonales [Levin, Morgan, 1975]:

1. Nivel de compartición
2. Comportamientos de modelos de acceso
3. Nivel de conocimiento en comportamiento de modelo de acceso.

La Figura 3.6. describe las alternativas a lo largo de estas dimensiones.

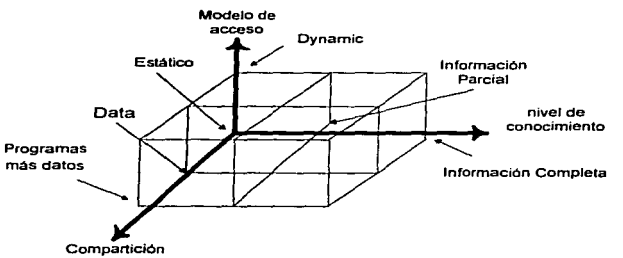


Fig. 3.6 Estructura de Distribución

En términos del nivel de compartición, existen tres posibilidades. Primero *no hay compartición* y cada aplicación y sus datos se ejecutan en un sitio, de manera que no hay comunicación con ningún otro programa o no hay acceso a archivos de datos en otro sitio. En consecuencia, los requerimientos del usuario son manejados en el sitio donde fueron originados y los archivos necesarios son movidos alrededor de la red. Por el otro lado está, el comportamiento *data-plus-program*, donde datos y programas pueden ser compartidos, significando que un programa que se encuentra en un sitio puede requerir un servicio de otro programa que se encuentre en un segundo sitio, el cuál a su vez puede requerir de otro programa que se encuentre en un tercer sitio.

Leving y Morgan remarcan una distinción entre la compartición de datos y la compartición de data-plus-program para ilustrar las diferencias entre sistemas computarizados distribuidos heterogéneos y homogéneos. Ellos indican correctamente, que un ambiente heterogéneo es usualmente muy difícil, si no imposible, ejecutar un programa dado sobre un hardware diferente bajo un sistema operativo diferente.

A lo largo de la dimensión de comportamiento de modelos de acceso, es posible identificar dos alternativas. Los modelos de acceso de requerimientos del usuario pueden ser *estáticos* (aquéllos que no cambian con el tiempo), o *dinámicos*. Es obviamente bastante más fácil planear y administrar los ambientes estáticos que los dinámicos como es el caso de los sistemas distribuidos dinámicos.

Desafortunadamente es difícil encontrar muchas aplicaciones distribuidas en la vida real que pudieran clasificarse como estáticas. La pregunta importante no es entonces si un sistema es dinámico o estático, sino qué tan dinámico es. Es a lo largo de esta dimensión que la relación entre el diseño de la base de datos distribuida y el procesamiento de consultas es establecido.

• La tercera dimensión de clasificación es el nivel de conocimiento acerca del comportamiento del modelo de acceso. Una posibilidad, por supuesto es que los diseñadores no tengan nada de información acerca de como los usuarios accesarán las bases de datos. En esta situación diseñar un DBMS distribuido que pueda resolver efectivamente el problema en muy difícil. Existen dos alternativas más prácticas: que los diseñadores tengan información completa, donde los modelos de acceso puedan ser predecidos razonablemente y no se desvíen significativamente de esas predicciones y que los diseñadores tengan información parcial donde existirán desviaciones de las predicciones.

3.9 Diseño de distribución

En cuanto a la fragmentación, el tema importante es la unidad apropiada de distribución. Una relación no es la unidad adecuada pues las vistas de aplicación son usualmente subconjuntos de relaciones. Por lo tanto, la localidad de los accesos de las aplicaciones es definida no sobre las relaciones enteras sino sobre sus subconjuntos. De esta manera es natural considerar los subconjuntos de las relaciones como unidades de distribución. Además, si la aplicación tiene vistas definidas sobre una relación obtenida en diferentes sitios, se pueden seguir dos alternativas con la relación entera siendo la unidad de distribución:

- Que la relación es no duplicada y es almacenada solamente en un sitio.
- Que la relación es replicada en todo o algunos de los sitios donde la replicación reside.

La primera resulta de un alto volumen innecesario de accesos de datos remotos. Y la segunda, por otro lado, tiene innecesaria duplicación, las cuales causan problemas al ejecutar actualizaciones y esto no es adecuado si el espacio es limitado.

En la descomposición de una relación en fragmentos, cada fragmento es tratado como una unidad, y permite que las transacciones se ejecuten concurrentemente. La fragmentación de las relaciones típicamente resulta en la ejecución paralela de una sola consulta la cual es dividida en un conjunto de subconsultas que operan en fragmentos.

Por otro lado también se tiene que discutir sobre las desventajas de fragmentación. El principal problema está relacionado al control de datos semánticos, específicamente al chequeo de integridad. Como resultado de la fragmentación, los atributos participantes en un dependencia pueden ser descompuestos en diferentes fragmentos los cuales deberían estar situados en diferentes lugares. En este caso, el simple trabajo de checar las dependencias daría como resultado realizar búsquedas de datos en un gran número de lugares.

3.10 Fragmentación

El tema que se trata aquí, es encontrar alternativas para dividir una relación en otras más pequeñas.

¿Qué tanto se debe fragmentar?

El grado en que se debe fragmentar una base de datos es una decisión importante que afecta el rendimiento de la ejecución de consulta. El que tanto se debe fragmentar va desde un extremo, que es, no fragmentarlo todo, hasta el otro extremo, fragmentar hasta el nivel de tuplos individuales (en el caso de fragmentación horizontal) o al nivel de atributos (en el caso de fragmentación vertical).
Tenemos los efectos adversos de las unidades muy largas o muy cortas de fragmentación. Lo que necesitamos, entonces, es encontrar un adecuado nivel de fragmentación el cual esté a la mitad de los dos extremos. Tal nivel puede estar definido solamente con respecto a las aplicaciones que correrán sobre la base de datos. Las aplicaciones necesitan estar representadas con respecto a un números de parámetros. Conforme a los valores de estos parámetros, los fragmentos individuales pueden ser identificados.

Reglas para revisar que se lleve a cabo correctamente la fragmentación

Pondremos en vigor las siguientes tres reglas durante la fragmentación, las cuales juntas aseguran que la base de datos no sufrirá cambios semánticos durante la fragmentación.

1. **Integridad**: Si una relación R es descompuesta en fragmentos R_1, R_2, ... R_n, cada objeto de datos que puede ser encontrado en R puede ser encontrado en uno o más de los fragmentos R_i. Esta propiedad es también importante en la fragmentación porque asegura que los datos en una relación global son mapeados dentro de los fragmentos sin sufrir pérdida alguna. Note que en el caso de fragmentación horizontal, el objeto se refiere a una tupla, mientras que en el caso de una fragmentación vertical, se refiere a un atributo.

2. **Reconstrucción:** Si una relación R es descompuesta en fragmentos F, R_1, R_2, ..., R_n, debe ser posible definir un operador relacional ∇ tal es:

$$R = \nabla R_i, \ \forall R_i \in F_R$$

El operador ∇ será diferente para las diferentes formas de fragmentación; es importante, sin embargo, que éste puede ser identificado. La reconstrucción de la relación a partir de sus fragmentos asegura que las limitaciones definidas sobre los datos en la forma de dependencias son conservadas.

1. **Separación:** Si la relación R es descompuesta verticalmente, los atributos de su llave primaria son repetidos en todos sus fragmentos. Por lo tanto, en caso de particiones verticales, la descomposición está definida solamente en los atributos de las llaves no primarias de una relación.

Existen tres maneras para fragmentar las relaciones:

- **Horizontal**
- **Vertical**
- **Mixta**

3.11 Fragmentación horizontal

Se refiere a la partición de una relación a lo largo de sus tuplas. De esta manera cada fragmento tiene un subconjunto de tuplas de la relación. Hay dos versiones de fragmentación horizontal: *primaria y derivada.* La fragmentación horizontal primaria de una relación se lleva a cabo usando predicados que son definidos sobre esa relación y la fragmentación horizontal derivada, por otro lado, es el particionamiento de una relación que resulta de los predicados que han sido definidos sobre otra relación.

3.12 Fragmentación horizontal primaria

Una fragmentación horizontal primaria esta definida por una operación de selección sobre las relaciones propias de un esquema de base de datos. Por lo tanto, dada la relación R_i, un fragmento horizotal j, denotado como R_i^j, se define como :

$$R_i^j = \sigma F_j (R_i^j),\ 1 \leq j \leq w$$

Donde F_j es la fórmula de selección usada para obtener el fragmento R_i. Una fragmentación horizontal R_i de la relación R_i consiste en todos los tuplos de R_i que satisfacen F_j dada una F_j.

3.13 Fragmentación Horizontal derivada

La fragmentación horizontal derivada se realiza considerando la interrelación que existe entre la relación a fragmentar y una relación que ha sido fragmentada horizontalmente, es decir, se a R una relación a fragmentar y sea S una relación que ha sido fragmentada en forma horizontal en w fragmentos, los fragmentos de R se definen como:

$$R_i = R \triangleright\!\!\triangleleft S_i, \ 1 \leq i_w$$

donde w es el número máximo de fragmentos que van a ser definidos en R , y $S_i = \sigma_{F_i}(S)$, donde F_i es la fórmula de acuerdo a la cual se obtuvo el fragmento S_i de la relación S.

3.14 Fragmentación Vertical

La fragmentación vertical de una relación es la subdivisión de sus atributos en grupos. Los fragmentos se obtienen proyectando la relación global sobre cada uno de los grupos de atributos. Este tipo de fragmentación puede ser útil en aplicaciones en las cuales los atributos presentan propiedades geográficas comunes. La fragmentación vertical puede ocasionar que la relación original no pueda ser reconstruida a partir de sus fragmentos si no se toma en cuenta las dependencias funcionales al momento de fragmentar la relación. Existen dos formas para garantizar que la relación original se pueda reconstruir: primero,en cada uno de los fragmentos se incluye la llave de la relación global; segundo, se genera automaticamenteun identificador de tuplo.

3.15 Fragmentación Mixta

En muchos casos una fragmentación horizontal o vertical simple de un esquema de base de datos podría no ser suficiente para satisfacer los requerimientos de las aplicaciones de los usuarios. En este caso una fragmentación vertical debe ser seguida por una horizontal o viceversa, y dado esto resulta un tercera fragmentación estructurada(Figura 3.8). Es decir, si dos tipos de estrategias de partición son aplicadas una después de otra, se dice que se realiza una fragmentación híbrida o más conocida

como fragmentación mixta, que es como la llamaremos aquí. La fragmentación mixta se puede representar en forma conveniente por medio de una árbol de fragmentación, en el cual la raíz corresponde a la relación global, las hojas a los fragmentos y los nodos intermedios corresponden a los resultados intermedios de las expresiones que definen los fragmentos, en la Figura 3.7 se muestra un ejemplo.

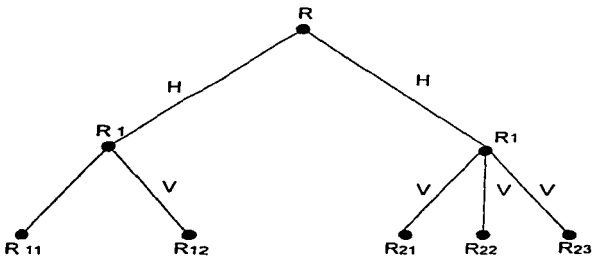


Fig. 3.7 Fragmentación Mixta

La reconstrucción de una relación global en caso de fragmentación mixta, uno empieza de por lo menos de la partición tres y se mueve hacia arriba para llevar a cabo joins y uniones (Figura 3.8). La fragmentación esta completa si el intermediario y los fragmentos de hoja son completos. Similarmente, la separación esta garantizada si el intermediario y los fragmentos de la rama están separados.

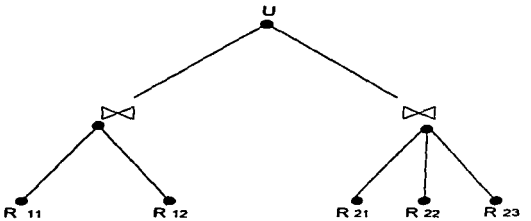


Fig. 3.8 Reconstrucción de una Fragmentación Hibrida

4. ANALISIS Y DISEÑO DEL PROTOTIPO DEL DDBMS DISBAGE

4.1 Metodología de Análisis y Diseño

Los métodos han evolucionado conforme a la complejidad creciente de los sistemas de software. Aquí se utilizará el método orientado a objetos dado que el desarrollo del sistema de realizará en C++.

El método orientado a objetos su concepto es el de modelar los sistemas como colecciones de objetos que cooperan, tratando los objetos individuales como instancias de una clase que está dentro de una jerarquía de clases . El diseño orientado a objetos refleja directamente la topología de lenguajes de programación de alto nivel más recientes, como Smalltalk, Object Pascal, C++, Common Lisp Object System (CLOS) y Ada.

La descomposición orientada a objetos produce sistemas más pequeños a través de la reutilización de mecanismos comunes. Además reduce el riesgo que representa construir sistemas de software complejos, porque están diseñados para evolucionar de forma incremental partiendo de sistemas más pequeños en los que ya se tiene confianza. Y por último la descomposición orientada a objetos resuelve directamente la

complejidad innata del software ayudando a tomar decisiones inteligentes respecto a la separación de intereses en un gran espacio de estados.

El modelo de objetos ha demostrado ser un concepto unificador en la informática, aplicable no sólo a los lenguajes de programación, sino también al diseño de interfaces de usuario, bases de datos e incluso arquitecturas de computadoras.

4.2 Prototipo del DDBMS DISBAGE

El prototipo del DDBMS propuesto en esta tesis es muy primitivo pero cumple con los requisitos esenciales de una sistema distribuido, el propósito de esta tesis es dar una punto de partida para que de aquí sea posible realizar un manejador de base de datos distribuida y sirva para otros estudiantes que tan importante e interesante es un DDBMS.

El objetivo de DDBMS **DISBAGE** es tener un punto de partida para desarrollar manejadores de bases de datos distribuidas, pues en la actualidad se encuentra todavía en investigación, pues a pesar de que se presentan desventajas también existen ventajas que ayudan eficientemente al almacenamiento de los datos y sobre todo ofrece seguridad en ellos ya que los datos no están almacenados en una sola parte y tiene la opción de contener copias de aquella información que pudiera ser importante y que en algún momento que se presentará alguna falla o pérdida de éstos estarían en otro lugar en la cuál se podría recuperar, además el tiempo de respuesta sería más rápido y eficiente ya que en los datos que se accesarán más se encontrarían distribuidos en las localidades más necesarias. Esto no quiere decir que se tiene que desplazar las bases de datos relacionales, de hecho este manejador esta diseñado en bases de datos relacionales, lo que se pretende es que los datos ya no se encuentren centralizados sino distribuidos.

El DDBMS **DISBAGE** utiliza a su vez una base de datos para guardar y manejar sus relaciones. Las relaciones que crea por default son **Atributo**, **Clave**, **Fragmento**, **Permiso**, **Relacion**, **Replicacion** y **Usuario**. Tanto las relaciones que cree un usuario como las que crea el sistema serán almacenadas en la relación **Relacion**, hasta ésta será almacenada ahí. Todas las relaciones que crea por inicio el sistema nunca podrán ser fragmentadas sobre todo la relación **Relacion** ya que si lo estuvieran fuera imposible localizar donde se encuentra cada relación. Cualquier relación que cree el usuario podrá ser fragmentada y replicada si éste así lo desea. **Base_Datos** no es un relación solo es un archivo en donde se guardará el nombre lógico de la base de datos y la ruta en donde se encuentra almacenada la base de datos.

4.3 Permisos

Políticas:

- Existen tres tipos de relaciones a las cuales se les imponen permisos, la relación Relacion, la relación Atributo y las relaciones No-sistema. Las relaciones No-sistema son todas aquellas que no son Relacion y Atributo

- Cada uno de los tres tipos de relaciones tienen un permiso especial para acceder a sus *n*-adas:

N-ada Relacion.	Permite ins, mod, elim, y cons *n*-adas en Relacion
N-ada Atributo.	Permite ins, mod, elim, y cons *n*-adas en Atributo
N-ada No-sistema.	Permite ins, mod, elim, y cons *n*-adas en relaciones No-sistema

- Cada una de los tres tipos de relaciones tienen un permiso especial otorgar o denegar sus permisos:

Permiso Relacion.	Permite otorgar/denegar permisos de ins, mod, elim, y cons *n*-adas en Relacion
Permiso Atributo.	Permite otorgar/denegar permisos de ins, mod, elim, y cons *n*-adas en Atributo
Permiso No-sistema.	Permite otorgar/denegar permisos de ins, mod, elim, y cons *n*-adas en relaciones No-sistema

- Cada vez que se inserta una relación, se dan todos los permisos posibles sobre la relación a su creador.

- El DBMS es el único usuario que tiene permisos sobre cualquier relación.

- Un usuario que tiene un permiso tipo Permiso obtiene también el permiso *N*-ada respectivo.

Todos los permisos son controlados a través de la relación Permiso:
Permiso (NombUsuar, NombLogRel, Ins, Mod, Elim, Cons, Tip)
Donde Tip es precisamente el tipo de permiso al que se hace referencia en la *n*-ada.

4.4 Formas Normales y 4NF de Boyce-Codd

Primera Forma Normal(1NF)

Una relación $R(A_1, A_2, \ldots A_n)$ se encuentra en primera forma normal si y solo si todos su atributos son atómicos, es decir, que los valores en el dominio no deben ser listas o conjuntos de valores.

Segunda Forma Normal (2NF)

Dependencia Funcional Parcial

Sea X y Y dos conjuntos de atributos de una relación R. Decimos que una dependencia funcional $X \rightarrow Y$ es total si y solo si para cualquier subconjunto propio de X ($X' \subset X$) tenemos $X' \not\rightarrow Y$. Si para algún subconjunto propio de X (X' subconjunto de X) tenemos que $X' \rightarrow Y$, entonces la dependencial funcional $X \rightarrow Y$ se denomina parcial.

Una dependencia funcional parcial indica que la relación representa dos entidades. Las llaves de una de ellas es X y la llave de la otra es el subconjunto propio X' tal que $X' \rightarrow Y$.

Sea X el conjunto de todos los atributos de la relación $R(A_1, A_2, \ldots A_n)$ que no participan en ninguna llave de R. Se dice que la relación R está en la segunda forma normal si cada atributo depende funcionalmente en forma total de cada llave de R.

Tercera Forma Normal

Dependencia Transitiva

Sea X, Y, Z conjuntos diferentes de atributos de la relación $R(A_1, A_2, \ldots A_n)$ decimos que Z depedende transitivamente de X si y solo si se cumplen las siguientes condiciones: $X \rightarrow Y$, $Y \not\rightarrow X$, $Y \rightarrow Z$ donde la dependencia funcional no es insignificante.

Una relación R esta en tercera forma normal si ninguno de sus atributos no primarios dependen transitivamente de alguna llave de R.

Forma Normal de Boyce-Codd (BCNF)

Dependencia Multivaluada (A_1, A_2,...A_n) está en BNCF si la existencia de una dependencia funcional no trivial $X \rightarrow Y$, donde X y Y son conjuntos de atributos de R, implica la existencia de una dependencia funcional $X \rightarrow A_i$, para toda i=1,2,..., n.

Una relación R(X,Y,Z) donde X, Y, Z son conjuntos de atributos que por pares son disjuntos, está en la cuarta forma normal si la existencia de una dependencia multivaluada no insignificante $X \rightarrow Y$ implica la existencia de una dependencia funcional $X \rightarrow A_i$ para todos los atributos A_i de R.

4.5 Relaciones del DDBMS DISBAGE

Atributo	(NombAtrib, NombLogRel, Pos, Llave, Tip, RestrIns, RestrElim, NombRep, Comen)
Clave	(NombLogUsuar,Cifrado)
Fragmento	(NombLogFrag, NombLogRel, Cond)
Permiso	(NombLogUsuar, NombLogRel, Ins, Mod, Elim, Cons, Tip)
Relacion	(NombLogRel, NombFis, Temp)
Replicacion	(NombLogRepl, NombLogRel)
Usuario	(NombLogUsuar, NombFis, ProgInic)
Base_Datos	[NombLogBasDat, NombFis]) No ta: No es una relación. (es un archivo)

——— = Llave primaria
..... = Llave primaria foránea
——— = Llave foránea

4.6 Dependencias funcionales

Atributo
NombAtrib, NombLogRel → Pos, Llave, Tip, RestrIns, RestrElim, NombRep, Comen

Clave
NombLogUsuar → Cifrado

Fragmento
NombLogFrag → NombLogRel, Cond

Permiso
NombLogUsuar, NombLogRel → Ins, Mod, Elim, Cons, Tip

Relacion
NombLogRel → NombFis, Temp
NombFis → NombLogRel, Temp
NombLogRel, NombFis → Temp

Replicacion
NombLogRepl → NombLogRel

Usuario
NombLogUsuar → NombFis, ProgInic

Nota: Observe que, de las dependencias funcionales de Clave y Usuario, por aditividad obtenemos NombLogUsuar→ NombFis,ProgInic,Cif. Sin embargo se ha decidido desnormalizar Usuario para que los cifrados de las claves de usuario queden en una relación separada por motivos de seguridad. Por lo demás, salta a la vista que las relaciones mínimas propuestas para nuestro prototipo de DDBMS cumplen con las formas normales 1ª, 2ª, 3ª y Boyce-Codd.

4.7 Descripción de las relaciones del DDBMS DISBAGE

La relación Atributo sirve para almacenar todos los atributos de las relaciones ya sea las del sistema o las de No-sistema.

La relación Clave se crea para tener seguridad al accesar a las relaciones ya que aquí se guardan los nombres de los usuarios y sus respectivas claves de acceso.

La relación que da los permisos para insertar, modificar, eliminar o consultar una relación es la relación Permiso y ésta se relaciona con la relación anterior.

Cuando se realiza una fragmentación cada fragmento es tomado y almacenado como una relación pero también se almacenarán en la relación Fragmento pues ésta sirve además de darle el nombre lógico del fragmento y el nombre de la relación lógica a la que pertenece el fragmento también permitirá establecer que tipo de fragmentación utilizo el usuario si vertical, horizontal o mixta.

La relación que también es de gran importancia es la relación Relacion ya que ahí se guardan todas las relaciones creadas por los usuarios incluyendo los fragmentos y las replicaciones. También se almecenan las consultas que realizan los usuarios pues como se sabe una consulta de relaciones resulta a su vez una relación temporal.

Todos los archivos de relaciones tienen la terminación .Rel.
Todos los archivos de relaciones temporales tienen la terminación .Temp.

La relación Replicacion almacenará las replicas que realice el usuario de las relaciones que existen.

La relación Usuario es creada para almacenar todos los nombres de los usuarios y el programa de inicio que utiliza cada usuario.

Base_Datos hay que hacer notar que esta no es una relación si no que se va a manejar como un archivo en donde se ubicarán y se almacenarán todas las bases de datos que se creen. Sin embargo se mostrará en el diagrama de Entidad-Relación pero eso no significa que sea una relación, se indica ahí para hacer notar que existe pero solo como un archivo.

Atributo

NombAtrib	Atributo	1	1					Nombre del atributo
NombLogRel	Atributo	2	1					Nombre lógico de la relación
Pos	Atributo	3	0					Posición del atributo en la relación
Llave	Atributo	4	0					Igual a 1 si el atributo es llave de otra forma es 0
Tip	Atributo	5	0					Tipo de dato del atributo
RestrIns	Atributo	6	0					Restricciones de inserción
RestrElim	Atributo	7	0					Restricciones de eliminación
NombRep	Atributo	8	0					Nombre de reporte el atributo
Comen	Atributo	9	0					Comentarios del Atributo
NombLogUsuar	Clave	1	1					Nombre lógico del usuario
Cifrado	Clave	2	0					Cifrado de la clave del usuario
NombLogFrag	Fragmento	1	1					Nombre lógico del fragmento
NombLogRel	Fragmento	2	1					Nombre lógico de la relación
Cond	Fragmento	3	0					Condición de fragmentación
NombLogUsuar	Permiso	1	1					Nombre lógico del usuario
NombLogRel	Permiso	2	1					Nombre lógico de la relación
Ins	Permiso	3	0					Permiso para insertar
Mod	Permiso	4	0					Permiso para modificar
Elim	Permiso	5	0					Permiso para eliminar
Cons	Permiso	6	0					Permiso para consultar
Tip	Permiso	7	0					Tipo de permiso
NombLogRel	Relacion	1	1					Nombre lógico de la relación
NombFis	Relacion	2	1					Nombre físico de la relación
Temp	Relacion	3	0					Igual a 1 si la relación es temporal de otra forma es 0
NombLogRepl	Replicacion	1	1					Nombre lógico de la relación replicación
NombLogRel	Replicacion	2	1					Nombre lógico de la relación
NombLogUsuar	Usuario	1	1					Nombre lógico del usuario
NombFis	Usuario	2	0					Nombre físico del usuario
ProgInic	Usuario	3	0					Programa inicial del usuario

Relacion

Atributo	Atributo.Rel	0
Fragmento	Framento.Rel	0
Permiso	Permiso.Rel	0
Relacion	Relacion.Rel	0
Replicacion	Replicacion.Rel	0
Usuario	Usuario.Rel	0
Clave	Clave.Rel	0

Clave

DBA	

Usuario

DBA	Data Base Administrator	DBMS_HOME/main

Permiso

							Comentarios (no forman parte de la relación Permiso)
DBMS	*	1	1	1	1	Permiso No-Sistema	Puede otorgar/denegar permisos de Ins,Del,Mod,Cons n-adas sobre la relación No-Sistema con NombLogRel=*
DBMS	*	1	1	1	1	Permiso Relacion	Puede otorgar/denegar permisos de Ins,Del,Mod,Cons n-adas sobre la relación Relacion con NombLogRel=*
DBMS	*	1	1	1	1	Permiso Atributo	Puede otorgar/denegar permisos de Ins,Del,Mod,Cons n-adas sobre la relación Atributo con NombLogRel=*
DBA	*	1	0	0	1	Permiso Relacion	Puede otorgar/denegar permisos de Ins,Cons n-adas sobre la relación No-Sistema con NombLogRel=*
DBA	*	0	0	0	1	Permiso Atributo	Puede otorgar/denegar permisos de Cons n-adas sobre la relación Atributo con NombLogRel=*
DBA	Base_Datos	1	1	1	1	Permiso No-Sistema	Puede otorgar/denegar permisos de Ins,Del,Mod,Cons n-adas sobre la relación No-Sistema con NombLogRel=Base_Datos
DBA	Fragmento	1	1	1	1	Permiso No-Sistema	Puede otorgar/denegar permisos de Ins,Del,Mod,Cons n-adas sobre la relación No-Sistema con NombLogRel=Fragmento
DBA	Permiso	0	0	0	1	Permiso No-Sistema	Puede otorgar/denegar permisos de Cons n-adas sobre la relación No-Sistema con NombLogRel=Permiso
DBA	Replicacion	1	1	1	1	Permiso No-Sistema	Puede otorgar/denegar permisos de Ins,Del,Mod,Cons n-adas sobre la relación No-Sistema con NombLogRel=Replicacion
DBA	Usuario	1	1	1	1	Permiso No-Sistema	Puede otorgar/denegar permisos de Ins,Del,Mod,Cons n-adas sobre la relación No-Sistema con NombLogRel=Usuario
*	*	0	0	0	1	N-ada Relacion	Todos pueden consultar todas las n-adas de Relacion
*	*	0	0	0	1	N-ada Atributo	Todos pueden consultar todas las n-adas de Atributo
*	Base_Datos	0	0	0	1	N-ada No-Sistema	Todos pueden consultar todas las n-adas de Base_Datos
*	Fragmento	0	0	0	1	N-ada No-Sistema	Todos pueden consultar todas las n-adas de Fragmento
*	Permiso	0	0	0	1	N-ada No-Sistema	Todos pueden consultar todas las n-adas de Permiso
*	Replicacion	0	0	0	1	N-ada No-Sistema	Todos pueden consultar todas las n-adas de Replicacion
*	Usuario	0	0	0	1	N-ada No-Sistema	Todos pueden consultar todas las n-adas de Usuario

4.8 Diagrama Entidad-Relación

Un Diagrama de E-R es un modelo de red que describe con un alto nivel de abstracción la distribución de los datos almacenados en un sistema. Es importante modelar los datos de un sistema. Primeramente, porque las estructuras de datos y las relaciones pueden ser tan complejas que algunas veces es deseable examinarlas y enfatizarlas independientemente del proceso que se llevará a cabo.

Los componentes de este DER (Diagrama Entidad-Relación) son dos principalmente: Tipos de objeto y Notación alternativa para relaciones.

El Tipo de objeto es representada por medio de una caja rectangular y representa una colección o conjunto de objetos cuyos miembros individuales tienen las siguientes características:

1. Cada uno puede identificarse de manera única por algún medio.
2. Cada uno juega un papel necesario en el sistema que se construye.
3. Cada uno puede describirse por uno o más datos.

La Notación alternativa para relaciones utilizada en el DER es multidireccional y pueden ser leídas siguiendo cualquier dirección además muestran cardinalidad en donde la flecha de dos puntas seguidas muestra la relación de uno a muchos, mientras que se emplea una flecha sencilla para mostrar relaciones de uno a uno entre objetos.

Se da un breve explicación del DER. La entidad *Relacion* puede tener muchas replicaciones y el atributo por el cual se une a la entidad *Replicacion* es NombLogRel y una replicación pertenece a una sola relación y los atributos por las que se unen son *NombLogRel/NombLogRepl*.

La relación *Relacion* puede tener muchos atributos y el atributo por el cual se relaciona a la entidad *Atributo es* el *NombLogRel*.

La relación *Permisos* puede dar muchos tipos de permisos a una relación *Relacion* y el atributo que las relaciona es *NombLogRel*.

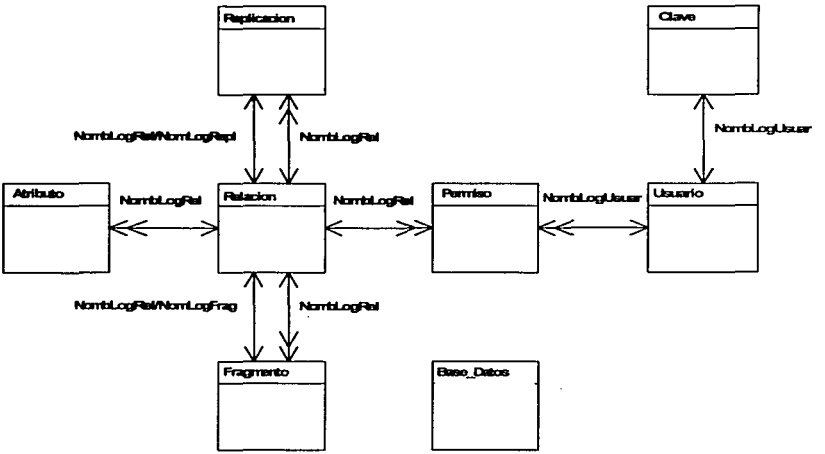
La relación *Relacion* puede tener muchos fragmentos pero un fragmento puede pertenecer a solo una relación, el atributo por las que se relacionan con la relación *Fragmento* es *NombLogRel* y *NombLogRel/NombLogFrag*.

La relación *Usuario* puede tener muchos tipos de permisos y la relación *Permisos* puede darle permisos a muchos usuarios, los atributos por los cuales se relacionan son *NombLogUsuar*.

La relación *Usuario* puede tener solo una clave y una clave es solo para un usuario, el atributo por el cual se relacionan con la relación *Clave* es *NomLogUsuar*.

Base_Datos es solo un archivo y no una relación y es utilizado para guardar el nombre lógico de la Base de datos y para guardar el nombre físico de ésta, es decir, sirve solo para almacenar la ruta o dirección en donde se guardarán la Base y sus relaciones respectivas. En realidad no se debería mostrar junto con el diagrama E-R pero se puso con la intención de que forma parte del análisis y diseño.

Diagrama Entidad-Relación



4.9 Diagrama de Clases

Una clase representa solo una abstracción, la <<esencia>> de un objeto. Un objeto es una entidad concreta que existe en el tiempo y en el espacio.

El concepto formal de una clase es :
" Una clase es un conjunto de objetos que comparten una estructura común y comportamiento común"[1]

Un único objeto es solo una instancia de una clase. Un objeto no es una clase, pero una clase puede convertirse en un objeto.

El diagrama de clase se usa para mostrar la existencia de clases y sus relaciones en la visión lógica de un sistema. Un solo diagrama de clases representa una vista de la estructura de clases de un sistema.

La clase se muestra por medio de un símbolo de forma de nube de líneas punteadas. Se requiere un nombre para cada clase. Todo nombre de clase debe ser único para la categoría de clases que engloba a ésta.

En este diagrama de clases es utilizado el símbolo de relación que es denominado herencia y es representado por una flecha →, este símbolo denota una relación de generalización/especialización. La flecha apunta a la superclase, y el extremo opuesto de la asociación designa la subclase. De acuerdo con las reglas del lenguaje de implantación elegido, la subclase hereda la estructura y el comportamiento de su superclase. Una clase puede tener una o más herencias lo cual se le conoce como herencia múltiple.

"La herencia es la jerarquía de <<clases>> más importante, y es un elementos esencial de los sistemas orientados a objetos. Básicamente, la herencia define una relación entre clases, en la que una clase comparte la estructura de comportamiento definida en una o más clases. La herencia representa así un jerarquía de abstracciones, en la que una subclase hereda de una o más superclases."[2]

El diagrama ilustra la estructura de clases del prototipo DDBMS disbage. La superclase DDBMS es un tipo de Red, QL, DML y Relación.

[1] G. Booch, Object Oriented Analysis and Design with Applications, p. 120
[2] Idem, p.66

La clase QL, DML y Relación es un tipo de Registro.

La clase Registro es un tipo de Vector(N-ada) y esta subclase a su vez es un tipo Atributo.

La subclase Relación es un tipo Archivo Descriptor y Archivo Datos. El Archivo Descriptor es usado solo si no existe el Archivo de dato.

La clase Archivo Descriptor y Archivo de Dato es un tipo de Archivo.

Como se ve en el diagrama el tipo de herencia es múltiple y se les denomina de tal manera porque una subclase tiene varias superclases como en el caso de la subclase registro.

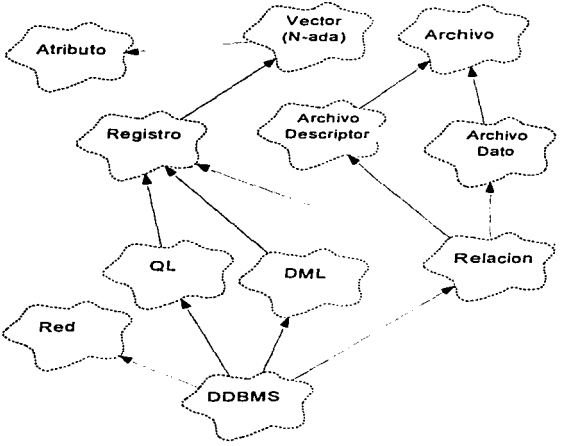


Diagrama de Clases

4.10 Lenguaje de Definición de datos (DDL)

- Crea una base de datos
- Crea una relación en la base de datos activa
- Borra una base de datos
- Borra una relación de la base de datos activa
- Cambia el nombre de una relación
- Selecciona una base de datos

Crea una base de datos

Crea una base de datos (BD) a partir de su nombre. Al crear la BD hace un nuevo directorio de trabajo llamado aNombreLogBDNueva, en el se crean las siguientes relaciones: Atributo, Clave, Fragmento, Permiso, Relacion, Replicacion, Usuario.

Parámetros:

aNombreLogBDNueva átomo que contiene de la base de datos a crear.
aNombreFisBDNueva da el nombre físico o ruta de la nueva base de datos.

Retorna:
Nil si existe algún error, de otra forma el nombre de la BD.

Crea una relación en la base de datos activa

Crea una relación para la base de datos (BD) activa. Existirá un error si la relación declarada ya había sido creada con anterioridad.

Parámetros:

aNombreLogRel átomo que contiene el nombre de la relación

Retorna:
aNombreLogRel si no existió error al crear la relación, de otra forma retorna NIL.

Borra una Base de Datos de la base de datos activa

Borra una relación llamada aNombreLogRel de la base de datos activa.

Parámetros:

aNombreLogRel átomo que contiene el nombre de la relación.

Retorna:

Verdadero si no existió error al borrar la relación, de otra forma retorna NIL.

Cambia el nombre de una relación

Cambia el nombre de la relación llamada aNombreLogRel (contenida en la base de datoa activa) al nombre aNombreRelNuev.

Parámetros:

aNombreLogRel átomo que contiene el nombre de la relación cuyo nombre será cambiado.
ANombreRelNuev átomo que contiene el nuevo nombre de la relación aNombreLogRel.

Retorna:

Verdadero si no existió error al borrar la relación, de otra forma retorna NIL.

Selecciona una base de datos

Selecciona una base de datos a partir de su nombre lógico y físico. Esta selección lee los tuplos de la relación cuyo nombre es referenciado por el símbolo de constante aNombreLogRelRels y que contiene a las relaciones integrantes de la base de datos. Si la lectura fue exitosa entonces se actualiza la variable del nombre de la base de datos activa aNombreBD y se borra los archivos temporales. Si no se encontró la base de datos llamada aNombreBDSel entonces aNombreBD asume el valor de NIL.

Parámetros:

aNombreLogBDSel átomo que contiene el nombre de la base de datos a seleccionar.
ANombreFisBDSel el nombre físico de la base de datoa a seleccionar.

Retorna:

NIL si no se encontro la base de datos aNombreLogBDSel o retorna aNombreBDSel si la operación fue exitosa.

4.11 Lenguaje de Manipulación de datos (DML)

En este DML se definirán las operaciones más importantes de una Base de datos las cuales son:

- Eliminación
- Inserción
- Modificación

Eliminación de *n*-adas de una relación que cumpla con ciertas condiciones.

Busca en la relación las *n*-adas que cumplan con las condiciones establecidas, si la condición no se cumple, no podrá ser eliminada y mandará un 0 que indicará que no se encontró ninguna *n*ada con esas condiciones. Da como parámetros en nombre lógico de la relación y la condición.

Ejemplo de llamada:

(aEliminarTups 'Persona '(EQUAL Persona.Nombre Gerardo)) ;Borra las personas cuyo nombre sea Gerardo de la relación Persona
(aEliminarTups 'Persona '(EQUAL Alejandro Nombre)) ;Borra las personas cuyo nombre sea Alejandro de la relación Persona

Inserción de una lista de *n*-adas en una relación.

Inserta las *n*-adas en la relación especificada y da como parámetros el nombre lógico de la relación y las *n*-adas nuevas a insertar.

Ejemplo de llamada:

(aInsertarTups 'Persona '((Gerardo 13 5 70 1.82))) ;Inserta una *n*-ada en la relación persona

Modificación de una *n*-ada en una relación.

Modifica los atributos con nuevos valores especificados y da como parámetros el nombre de la relación y las condiciones. Los atributos asumirán valores nuevos si las *n*-adas cumplen con el parámetro condición.

Ejemplo de llamada:

(aModificarTups 'Persona '((DiaNac 13)) '(EQUAL Persona.Nombre Gerardo)) ;Cambia el atributo dia_nac a 13 de las personas cuyo nombre sea Gerardo de la relación Persona
(aModificarTups 'Persona '((DiaNac (+ DiaNac 2)) '(EQUAL Alejandro Nombre))
Cambia el atributo dea_nac sumandole 2 a las personas cuyo nombre sea Alejandro de la relación Persona

4.12 Lenguaje de Consulta (QL)

En el lenguaje de consultas para una base de datos se verán los **operadores** del álgebra relacional.

- Composición
- División
- Exclusión
- Fusión (Intersección)
- Join
- Producto Cartesiano
- Proyección
- Resta
- Restricción
- Selección
- Unión

Cuantificadores más importantes:

- Existe es el cuantificador existencial para las *n*-adas de una relación.
- Todo es el cuantificador universal para las *n*-adas de una relación.

Operador de composición del álgebra relacional

Realiza la composición de las *n*-adas de las relaciones Relación1 y Relación2 que cumplan con la Condición1. Se puede almacenar la relación resultante en la relación RelaciónVista. Si RelaciónVista es igual a nula se crea una relación temporal con un nombre aleatorio. Se puede indicar que la relación tenga *n*-adas redundantes (es decir que se convierta en un multiconjunto), para ello existe una Relación que elimina la redundancia.

Los parámetros a utilizar son:

Relación1 es el nombre de la relación cuyas *n*-adas se compondrán con las *n*-adas de la Relación2 si se cumple la Condición1.

Relación2 es el nombre de la relación cuyas *n*-adas se compondrán con las *n*-adas de la Relación1 si se cumple la Condición1.

Nom_log_Rel_Temp es el nombre lógico de la relación temporal (vista) en la que se desea almacenar la consulta. Si es nula se asume el valor de Nom_fis_rel.

Nom_Fis_Rel_Temp si es igual a nula se genera un nombre aleatorio diferente a los de las relaciones que conforman la base de datos.

Con_Redun si es diferente de nula no se eliminan las *n*-adas redundantes, si es igual a vacía se eliminan las *n*-adas redundantes.

Ejemplo de llamada:

(aComposicionRel 'tios 'abuelos '(EQUAL abuelos.hijo tios.nombre)) ;*** Composición entre las *n*-adas de las relaciones *tios* y abuelos cuando el atributo nombre de tios sea igual al atributo nombre de abuelo

(aComposicionRel 'tios 'abuelos '(EQUAL abuelos.hijo tios.nombre) 'familia) ;*** Lo mismo que el anterior pero el resultado es almacenado en la relación temporal familia

(aComposicionRel 'tios 'abuelos '(EQUAL abuelos.hijo tios.nombre) 'familia 'T) ;*** Lo mismo que el anterior pero no elimina las redundancias

Operador de división del álgebra relacional

Regresa la división relacional de la relación Dividendo1 entre la relación Divisor1 dadas las *n*-adas Dividendo*Nada*1 y Divisor*Nada*1 de Dividento1 y Divisor1 respectivamente.

La división relacional efectúa un proceso de búsqueda de las restricciones que deben cumplir todos las *n*-adas de una relación con respecto a otra. De esta forma la división relacional es el operador inverso de el producto cartesiano. Tal proceso puede modificarse enviando al parámetro Evaluador una función de evaluación distinta a la función por omisión. Tal función de evaluación debe de aceptar dos parámetros el primero para la *nada* evaluada, el segundo para la lista de *n*-adas de Dividendo1. La *nada* evaluada esta formada por los valores de Divisor1.

Si se quiere se puede indicar que se almacena la relación temporal resultante en la RelaciónVista. Si la RelaciónVista es igual a nula se crea una relación temporal con un

nombre aleatorio. También si se quiere se puede indicar que la relación tenga *n*-adas redundantes (es decir se convierte en un multiconjunto), para ello ConRedun debe ser diferente de vacío.

Los parámetros a utilizar son:

Dividendo1 la relación que será dividida por la relación Divisor1.

Divisor1 la relación que dividirá a la relación aDividendo.

DividendoNada1 las *n*-adas de Dividendo1 que serán divididos.

DivisorNada1 las *n*-adas de Divisor1 que dividirá a las *n*-adas de Dividendo1.

Evaluador la función que evaluará la condición a cumplir por una *n*ada evaluado.

NombLogRelTemp nombre lógico de la relación temporal (vista) en la que se desea almacenar la consulta. Si es igual a vacío se asume el valor de NombFisRel.

NombFisRelTemp si es igual a vacío se genera un nombre aleatorio diferente a los de las relaciones que conforman la base de datos.

ConRedun si es diferente de vacío no se eliminan las *n*-adas redundantes, si es igual a vacío se eliminan las *n*-adas redundantes.

Ejemplo de llamada:

```
(aDivisionRel 'hombres 'alumnos '(nombre) '(nombre) ) ;*** División de la relación hombres entre alumnos
(aDivisionRel 'hombres 'alumnos '(nombre) '(nombre) '(LAMBDA (lUnTup lTodosTup) (NOT (MEMBER lUnTup lTodosTup 'EQUAL)) ) ;*** Residuo de la división en la relación hombres entre alumnos
(aDivisionRel 'hombres 'alumnos '(nombre) '(nombre) familia ) ;*** Lo mismo que el primero pero el resultado es almacenado en la relación temporal familia
(aDivisionRel 'hombres 'alumnos '(nombre) '(nombre) familia 'T ) ;*** Lo mismo que el anterior pero no elimina las redundancias
```

Operador de exclusión del álgebra relacional

Regresa la exclusión relaciona de la relación Relación1 dados las *nadas n*-adas1.

Se puede indicar que se almacene la relación temporal resultante en la relación RelacionVista. Si RelacionVista es igual a vacía se crea una relación temporal con un nombre aleatorio. Existe otra opción en la que se puede indicar que la relación tenga *n*-adas redundantes (es decir, se convierta en un multiconjunto), para ello ConRedun debe ser diferente de vacío.

Parámetros:

Relación1 es el nombre de la relación cuyas *n*-adas serán excluidas en las *n*-adas *n*-adas1.

N-adas1 es la lista de los atributos que serán excluidos de la relación Relación1.

Nom_Log_Rel_Temp es el nombre lógico de la relación temporal (vista) en la que se desea almacenar la consulta. Si es igual a vacía se asume el valor de Nom_Fis_Rel.

Nom_Fis_Rel_Temp si es igual a vacía se genera un nombre aleatorio diferente a los de las relaciones que conforman la base de datos.

Con_Redun si es diferente de vacío no se eliminan las *n*-adas redundantes, si es igual a vacío se eliminan las *n*-adas redundantes.

Ejemplo de llamada:

```
(aExclusionRel 'abuelos '(nombre nacimiento) ) ;*** Excluye la relación abuelo en los
atributos nombre y nacimiento
(aExclusionRel 'abuelos '(nombre nacimiento) 'familia ) ;*** Lo mismo que el anterior pero
el resultado es almacenado en la relación temporal familia
(aExclusionRel 'abuelos '(nombre nacimiento) 'familia 'T ) ;*** Lo mismo que el anterior
pero no elimina las redundancias
```

Operador de intersección del álgebra relacional

Regresa las *n*-adas que están en Relación1 y en Relación2. Tiene la opción de indicar que se almacene la relación temporal resultante en la relación RelacionVista. Si a RelacionVista es igual a vacía se crea una relación temporal con un nombre aleatorio. También se puede indicar que la relación tenga *n*-adas redundantes, para ellos Con_Redun debe ser diferente de vacío.

Parámetros:

Relación1 el nombre de la primera relación de la que se buscará intersección con Relacion2.

Relación2 el nombre de la segunda relación de la que se buscará intersección con Relación1.

NombLogRelTemp es el nombre lógico de la relación temporal (vista) en la que se desea almacenar la consulta. Si es igual a vacía se asume el valor de NombFisRel.

NombFisRelTemp si es igual a vacía se genera un nombre aleatorio diferente a los de las relaciones que conforman la base de datos.

ConRedun si es diferente de vacía no se eliminan las *n*-adas redundantes, si es igual a vacía se eliminan las *n*-adas redundantes.

Ejemplo de llamada:

(aInterseccionRel 'hombres 'alumnos) ;*** Intersección entre las relaciones hombres y alumnos
(aInterseccionRel 'hombres 'alumnos 'AlumnHom) ;*** Lo mismo que el anterior pero el resultado es almacenado en la relación temporal AlumnHom
(aInterseccionRel 'hombres 'alumnos 'AlumnHom 'T) ;*** Lo mismo que el anterior pero no elimina las redundancias

Operador de join del álgebra relacional

Retorna la fusión de las *n*-adas de las relaciones Relacion1 y Relacion2 que cumplan con la condición Condición1. Tiene la opción de que se puede indicar que se almacene la relación temporal resultante en la relación RelacionVista. Si RelacionVista es igual a vacía se crea una relación temporal con un nombre aleatorio. También se puede indicar que la relación tenga *n*-adas redundantes, para ello ConRedun debe ser diferente de vació.

Parámetros:

Relación1 es el nombre de la relación cuyas *n*-adas se fusionarán con las *n*-adas de la Relación2 si se cumple la condición Condición1.

Relación2 es el nombre de la relación cuyas *n*-adas se fusionarán con las *n*-adas de la Relación1 si se cumple la condición Condición1.

Condición1 la condición que deben cumplir las *n*-adas de las relaciones Relacion1 y Relacion2 para que sean fusion*n*-adas.

La condición puede ser cualquier tipo de función que puede contener como parámetros nombres de atributos de una relación. Para evitar ambigüedades, los nombres de los atributos pueden ser precedidos por el nombre de la relación a la que pertenecen y un punto

NombLogRelTemp es el nombre lógico de la relación temporal (vista) en la que se desea almacenar la consulta. Si es vacía se asume el valor de NombFisrel.

NombFisRelTemp si es igual a vacía se genera un nombre aleatorio diferente a los de las relaciones que conforman la base de datos.

ConRedun si es diferente de vacía no se eliminan las *n*-adas redundantes, si es igual a vacía se eliminan las *n*-adas redundantes.

Ejemplo de llamada:

(aJoinRel 'tios 'abuelos '(EQUAL abuelos.hijo tios.nombre)) ;*** Fusión entre los tuplos de las relaciones tios y abuelos cuando el atributo nombre de tios sea igual al atributo nombre de abuelo
(aJoinRel 'tios 'abuelos '(EQUAL abuelos.hijo tios.nombre) 'familia) ;*** Lo mismo que el anterior pero el resultado es almacenado en la relación temporal familia
(aJoinRel 'tios 'abuelos '(EQUAL abuelos.hijo tios.nombre) 'familia 'T) ;*** Lo mismo que el anterior pero no elimina las redundancias

Operador de producto cartesiano del álgebra relacional

Seleccionar las *n*-adas de la relación Relacion1 que cumplan con la condición Condicion1. Tiene la opción de que se puede indicar que se almacene la relación temporal resultante en la relación RelacionVista. Si RelacionVista es igual a vacía se crea una relación temporal resultante con un nombre aleatorio. También se puede establecer que la relación tenga *n*-adas redundantes, para ello ConRedun debe ser diferente de vacío.

Parámetros:

Relación1 es el nombre de la primera relación del producto cartesiano.

Relación2 es el nombre de la segunda relación del producto cartesiano.

NombLogRelTemp es el nombre lógico de la relación temporal (vista) en la que se desea almacenar la consulta. Si es vacía se asume el valor de NombFisRel.

NombFisRelTemp si es igual a vacía se genera un nombre aleatorio diferente a los de las relaciones que conforman la base de datos.

ConRedun si es diferente de vacía no se eliminan las *n*-adas redundantes, si es igual a vacía se eliminan las *n*-adas redundantes.

Ejemplo de llamada:

```
(aProductoCartRel 'hombres 'mujeres ) ;*** Producto cartesiano entre las relaciones hombres y mujeres
(aProductoCartRel 'hombres 'mujeres 'Parejas ) ;*** Lo mismo que el anterior pero el resultado es almacenado en la relación temporal parejas
(aProductoCartRel 'hombres 'mujeres 'Parejas 'T ) ;*** Lo mismo que el anterior pero no elimina las redundancias
```

Operador de proyección del álgebra relacional

Regresa la proyección relacional de la relación Relacion1 dados los atributos Atributos1.

Se puede indicar que se almacene la relación temporal resultante en la relación RelacionVista. Si RelacionVista es igual a vacía se crea una relación temporal con un nombre aleatorio. También se puede indicar que la relación tenga *n*-adas redundantes, para ello ConRedun debe ser diferente de vacío.

Parámetros:

Relación1 es el nombre de la relación cuyos *n*-adas serán proyectados en los atributos Atributos1.

Atributos1 la lista de los atributos que serán proyectados de la relación Relacion1.

NombLogRelTemp es el nombre lógico de la relación temporal (vista) en la que se desea almacenar la consulta. Si es vacía se asume el valor de NombFisRel.

NombFisRelTemp si es igual a vacía se genera un nombre aleatorio diferente a los de las relaciones que conforman la base de datos.

ConRedun si es diferente de vacía no se eliminan las *n*-adas redundantes, si es igual a vacía se eliminan las *n*-adas redundantes.

Ejemplo de llamada:
(aProyeccionRel 'abuelos '(nombre nacimiento)) ;*** Proyecta la relación abuelo en los atributos nombre y nacimiento
(aProyeccionRel 'abuelos '(nombre nacimiento) 'familia) ;*** Lo mismo que el anterior pero el resultado es almacenado en la relación temporal familia
(aProyeccionRel 'abuelos '(nombre nacimiento) 'familia 'T) ;*** Lo mismo que el anterior pero no elimina las redundancias

Operador de resta del álgebra relacional

Regresa las *n*-adas que están en Relacion1 y que no están en Relacion2.

Se puede establecer que se almacene la relación temporal resultante en la relación RelacionVista . Si a RelacionVista es igual a vacía se crea una relación temporal con un nombre aleatorio. También se puede establecer que la relación tenga *n*-adas redundantes, para ello ConRedun debe ser diferente de vacío.

Parámetros:

Relación1 es el nombre de la relación de la que restará la Relacion2 .

Relación2 es el nombre de la relación que se restará a Relacion1 .

NombLogRelTemp es el nombre lógico de la relación temporal (vista) en la que se desea almacenar la consulta. Si es vacía se asume el valor de NombFisRel.

NomFisRelTemp si es igual a vacía se genera un nombre aleatorio diferente a los de las relaciones que conforman la base de datos.

ConRedun si es diferente de vacía no se eliminan las *n*-adas redundantes, si es igual a vacía se eliminan los *n*-adas redundantes.

Ejemplo de llamada:
(aRestaRel 'hombres 'alumnos) ;*** Resta entre las relaciones hombres y alumnos
(aRestaRel 'hombres 'alumnos 'HomNoAlu) ;*** Lo mismo que el anterior pero el resultado es almacenado en la relación temporal HomNoAlu
(aRestaRel 'hombres 'alumnos 'HomNoAlu 'T) ;*** Lo mismo que el anterior pero no elimina las redundancias

Operador de restricción del álgebra relacional

Regresa las *n*-adas de la relación Relacion1 que cumplan con la condición Condicion1. La condición Condición1 se supone restringida con respecto a la relación Relacion2. Se puede indicar que se almacene la relación temporal resultante en la relación RelacionVista. Si a RelacionVista es igual a vacía se crea una relación temporal con un nombre aleatorio. También se puede establecer que la relación tenga *n*-adas redundantes, para ello ConRedun debe ser diferente de vacío.

Parámatros:

Relación1 es el nombre de la relación cuyas *n*-adas se estarán restringidas a las condiciones Condicion1 con respecto a la relación Relacion2.

Relación2 es el nombre de la relación cuyas *n*-adas restringirán a las *n*-adas de Relacion1 si se cumple la condición Condición1.

Condición1 la condición que deben cumplir las *n*-adas de las relaciones Relacion1 y Relacion2 para que se cumpla la restricción.

La condición puede ser cualquier tipo de restricción que puede contener como parámetros nombres de atributos de una relación. Para evitar ambigüedades, los nombres de los atributos pueden ser precedidos por el nombre de la relación a la que pertenecen y un punto.

NomLogRelTemp es el nombre lógico de la relación temporal (vista) en la que se desea almacenar la consulta. Si es vacía se asume el valor de NomFisRel.

NomFisRelTemp si es igual a vacía se genera un nombre aleatorio diferente a las de las relaciones que conforman la base de datos.

ConRedun si es diferente de vacía no se eliminan las *n*-adas redundantes, si es igual a vacía se eliminan los *n*-adas redundantes.

Ejemplo de llamada:

```
(aRestriccionRel 'tios 'abuelos '(EQUAL abuelos.hijo tios.nombre) ) ;*** Restricción
entre los tuplos de las relaciones tios y abuelos cuando el atributo nombre de tios sea
igual al atributo nombre de abuelo
 (aRestriccionRel 'tios 'abuelos '(EQUAL abuelos.hijo tios.nombre) 'familia ) ;*** Lo
mismo que el anterior pero el resultado es almacenado en la relación temporal familia
 (aRestriccionRel 'tios 'abuelos '(EQUAL abuelos.hijo tios.nombre) 'familia 'T ) ;***
Lo mismo que el anterior pero no elimina las redundancias
```

Operador de selección del álgebra relacional

Selecciona las *n*-adas de la relación Relacion1 que cumplan con la condición Condicion1. Se puede indicar que se almacene la relación temporal resultante en la relación RelacionVista . Si a RelacionVista es igual a vacía se crea una relación temporal con un nombre aleatorio. También se puede establecer que la relación tenga *n*-adas redundantes, para ello ConRedun debe ser diferente de vacio.

Parámatros:

Relación1 es el nombre de la relación de la que se seleccionarán las *n*-adas.

Condición1 la condición que deben cumplir una *n*ada de la relación Relacion1 para que sea seleccionado.

La condición puede ser cualquier tipo de selección que puede contener como parámetros nombres de atributos de una relación. Para evitar ambigüedades, los nombres de los atributos pueden ser precedidos por el nombre de la relación a la que pertenecen y un punto.

RelacionVista nombre de la relación temporal (vista) en la que se desea almacenar la consulta. Si el igual a vacía se generan nombre aleatorio diferente a los de las relaciones que conforman la base de datos.

ConRedun si es diferente de vacía no se eliminan las *n*-adas redundantes, si es igual a vacía se eliminan los *n*-adas redundantes.

Ejemplo de llamada:

```
(aSeleccionRel 'ejemplo '(EQUAL sAtributo1 "Valor buscado") ) ;Selecciona los
tuplos cuyo sAtributo1 sea igual a "Valor Buscado"
(aSeleccionRel 'ejemplo '(EQUAL sAtributo1 "Valor buscado") NIL 'T) ;Idéntico al
ejemplo anterior pero se solicita que no se eliminen las redundancias
(aSeleccionRel 'ejemplo '(EQUAL sAtributo1 "Valor buscado") 'Sel 'T) ;Idéntico al
ejemplo anterior pero se solicita que se almacene el resultado en la relación Sel
```

Operador de unión del álgebra relacional

Retorna las *n*-adas que están en Relacion1 o que están en Relación

Tiene la opción de que se puede indicar que se almacene la relación temporal resultante en la relación RelacionVista. Si RelacionVista es igual a vacía se crea una relación temporal con un nombre aleatorio. También se puede indicar que la relación tenga *n*-adas redundantes, para ello ConRedun debe ser diferente de vació.

Parámetros:

Relación1 es el nombre de la relación que se unirá a Relacion2

Relación2 es el nombre de la relación que se unirá a Relacion1

NombLogRelTemp es el nombre lógico de la relación temporal (vista) en la que se desea almacenar la consulta. Si es vacía se asume el valor de NomFisRel.

NomFisRelTemp si es igual a vacía se genera un nombre aleatorio diferente a los de las relaciones que conforman la base de datos.

ConRedun si es diferente de vacía no se eliminan las *n*-adas redundantes, si es igual a vacía se eliminan las *n*-adas redundantes.

Ejemplo de llamada:

```
(aUnionRel 'tios 'abuelos ) ;*** Unión entre las relaciones tíos y abuelos
(aUnionRel 'tios 'abuelos 'familia ) ;*** Lo mismo que el anterior pero el resultado es
almacenado en la relación temporal familia
(aUnionRel 'tios 'abuelos 'familia 'T ) ;*** Lo mismo que el anterior pero no elimina
las redundancias
```

Cuantificador existencial para los tuplos de una relación

Retorna NIL si no existe al menos un tuplo de la relación Relacion que cumpla la condición lCondicion. De otra forma retorna el primer tuplo que cumpla con lCondicion

Parámetros:

aRelacion el nombre de la relación en cuyos tuplos se buscara el cumplimento de la condición lCondicion

lCondicion un predicado que puede contener como parámetros atributos de aRelacion. Tal predicado representa la condición que debe de cumplir al menos un tuplo de aRelacion para retornar un valor diferente de NIL.

Ejemplo de llamada:

(aExisteTup? alumnos '(EQUAL Nombre "Arturo")) ;*** Busca la existencia de al menos un alumno cuyo nombre es Arturo

Ejecuta el cuantificador existencial para los tuplos de una relación

Función auxiliar de aExisteTup? que retorna NIL si no existe al menos un tuplo de la relación aRelacion que cumpla la condición lCondicion. De otra forma retorna el primer tuplo que cumpla con lCondicion

Parámetros:

aRelacion el nombre de la relación en cuyos tuplos se buscara el cumplimento de la condición lCondicion.

lCondicion un predicado previamente "EVALUADO" que puede contener como parámetros atributos de aRelacion. Tal predicado representa la condición que debe de cumplir al menos un tuplo de aRelacion para retornar un valor diferente de NIL.

Cuantificador universal para las n-adas de una relación

Regresa a vacío si ninguna *n*-ada de la relación Relacion1 cumple la condición Condicion1. De otro manera regresa a verdadero, es decir, 'T'.

Parámetros:

Relacion1 el nombre de la relación cuyas *n-adas* se buscará el cumplimiento de la condición Condicion1.

Condición1 Un predicado que puede contener como parámetros atributos de Relacion1. Tal predicado representa la condición que deben cumplir todos las *n-adas* de Relación para regresar un valor diferente de vacío.

Ejemplo de llamada:

(aTodoTup? alumnos '(> EticaProm 8)) ;*** Retorna 'T si todos los tuplos de la relación alumnos tienen una calificación promedio en ética EticaProm mayor a 8

Ejecución del cuantificador universal para las *n-adas* de una relación

Retorna a vacío si ninguna *n*ada de la relación Relacion1 cumple la condición Condición1. De otra forma retorna 'T'.

Parámetros:

Relacion1 el nombre de la relación cuyas *n-adas* se buscará el cumplimiento de la condición Condición1.

Condicion1 un predicado previamente "Evaluado" que puede contener como parámetros atributos de Relacion1. Tal predicado representa la condición que deben cumplir todos las *n-adas* de Relacion1 para regresar un valor diferente de vacío.

5. PROTOTIPO DDBMS DISBAGE Y OTROS MANEJADORES DE BASE DE DATOS

5.1 Oracle

Para prevenir que una sola base de datos se sobrecargue con demandas por el usuario, los negocios quieren que los datos claves residan en más de un sitio. Ahora la Opción Replicación Rdb hace posible al usuario a sincronizar bases de datos Rdb para replicar todo o cualquier subconjunto de una o más base de datos.

Muchos negocios son institutos de almacenes de datos, haciendo disponible datos criticos de una o mas fuentes para un propósito determinado de consultas para soporte-decisión. La Opción Replicación posibilita a los usuarios a actualizar sus bases de datos destino tan frecuente como sea necesario sin interrumpir las operaciones de la base de datos fuente. Fijar la actualización de la base de datos puede ser aplicada a la copia tan bien como la fuente.

La Opción Replicación permite a los usuarios ligar base de datos tanto restringidamente como ampliamente según lo requiere el negocio, incluyendo:

- Rendimiento programado o propósitos determinados de actualización, tan frecuente como sea necesario, este puede ser cada 20 minutos o cada fin de semana.

- Usando un conjunto seleccionado de una o más fuentes de base de datos.

- Filtrando y normalizando datos para asegurar vistas únicamente de características requeridas en una manera consistente.

Transferencia de datos flexibles

La Opción Distribución soporta consolidación de datos y distribución de datos:

1. *Rollups*: copias parciales o totales de múltiples fuentes a un solo destino.

2. *Rollouts*: copias parciales o totales de una sola fuente a múltiples destinos.

La Opción Replicación proporciona métodos diferentes para llevar a cabo transferencias de datos.

1. *Extracción:* Una completa renovación o instantánea de los datos transferidos al destino.

2. *Replicación:* Una renovación incremental de los datos destinos de los cambios que han ocurrido en la base de datos fuente.

Opciones Paralela y Distribuida

Cuando es usada con la Opción Distribución, la Opción de Consulta Paralela y el Gateway transparente Rdb, la Opción Replicación da a los usuarios un conjunto de herramientas extremadamente poderosa para construir un almacén de datos Rdb.

Usar juntas estas herramientas ofrecen acceso flexible y capacidad de integración poderosa para utilizar completamente información corporada. Las organizaciones pueden replicar, consolidar e intercambiar datos a través de empresas de la mejor manera que requiera el negocio.

La Opción Replicación puede usar la Opción Distribución en el lugar de la fuente o tomar ventajas de una sola vista lógica que la Opción Distribución da a los lugares destinos, de este modo moderniza acceso de datos.

Replicación Heterogénea y Simétrica

La replicación permite realizar múltiples copias de datos para ser mantenidos en diferentes partes en un ambiente distribuido. En un sistema de replicación data warehousing, es típicamente utilizada para transportar los datos de un sistema OLTP en el almacén de datos.

Replicación puede también permanecer datos de mercado (o almacenes de datos departamentales) al actualizar estas pequeñas base de datos de un almacén de empresas.

La Opción Replicación avanzada de Oracle7 toma en cuenta actualizaciones sincrónicas de un almacén de datos, así como eventos basados y demandas basadas en métodos de replicación asíncrona. Replicación de tablas completas y replicación de subconjuntos de tablas son soportadas con métodos rápidos-renovación. Replicación de conjuntos de tablas es también soportada.

Oracle Rdb7 ofrece la solución: La opción Rdb7 distribuido. La opción distribuida permite a cada usuario a definir una vista lógica simple que integra datos provenientes de una variedad de base de dato físicos. Este catálogo global ofrece a los usuarios datos Rdb en cualquier parte como si viniera de una sola base de datos y sin tareas adicionales.

Rdb Distribuido

- Crea Base de datos lógica
- Crea ligas
- Establece seguridad
- Importa definiciones de metadatos
- Crea vistas

La opción distribución trabaja con:

1. Todas las redes, tanto IP con DECnet.
2. Todas las aplicaciones de usuario final y administradores de base de datos actuales aplicados a través de múltiples bases de datos a la vez.

Los beneficios de la opción distribuida son:

Mejora el rendimiendo del usuario final. El tiempo y el esfuerzo necesario para satisfacer una consulta son drásticamente reducidas, porque el cátalogo global de múltiples base de datos físicas permite a los usuarios a encontrar y recuperar información mucho más rápido.

Incrementa la eficiencia del sistema. La optimización distribuida y el canal de información de datos minimizan el tráfico de redes, ruteando consultas a través de los recursos más apropiados de la computación.

Integración de datos

Las compuertas(gateways) transparentes Rdb proporciona transparencia de acceso de lectura-escritura y bases de datos no relacionales provenientes de un solo API, el API Rdb. Gateways existen para Oracle 3, DBMS, RMS, IBM's, DB2 y Sybase y fuentes de datos ODBC.

La Opción Distribución ofrece a los usuarios vistas de todos los datos de la empresa, diseñado para todas las necesidades individuales. El usuario se libra de tareas complejas de formación, estas vistas de datos están localizadas en varias fuentes heterogéneas. Ahora, los usuarios finales y programadores de aplicación accesan a estas vistas como si fueran tablas en una sola base de datos relacional local. Estas vistas creadas pueden ser particionadas horizontalmente, dividir JOINS, indexar JOINS, y otros constructores avanzados de SQL.

Para esta finalidad, la Opción Distribuida utiliza:

- La arquitectura fundamental de los gateways Rdb, ofrece capacidad de infomación de base de datos a la Opción Distribuida.

- El optimizador de consultas toma factores de costo y capacidades para determinar el plan óptimo, el cual descompone consultas distribuidas al usar tantas características como sea posible de la base de datos fundamental y al emplear técnicas state-of-the-art, tanto consultas desagrupadas como eliminación de particiones.

- El constructor del esquema visual proporciona la habilidad para definir y manejar el catálogo global.

- Vistas particionadas horizontalmente, la cual soporta procesamiento distribuido continuo cuando la conectividad esta pérdida.

- Servicios de ayuda, el cual compensa para la deficiencia de capacidades en una base de datos fundamental.

- Checa seguridad, habilita fuera de la interface con el control de acceso especificado en las fuentes de datos fundamentales.

Varias de las características de la opción distribuida proporciona integración de datos en el nivel servidor, aumenta la productividad al usuario mientras minimiza el tráfico de la red y para las computadoras de escritorio.

Cuando trabajas con getways transparentes Rdb y opciones de consulta paralela, la Opción Distribución proporciona transparencia de base de datos cruzadas heterogéneas, trabajando cooperativamente con herramientas existentes y bases de datos. Utilizar un rango de características para dar un acceso de datos robusto y con alto rendimiento, además da integración de datos entre base de datos Rdb.

5.2 Sybase

El servidor de replicación Sybase va más allá de los límites de una base de datos distribuida para proporcionar la primera solución para verdaderos sistemas distribuidos. El servidor de replicación sincroniza copias replicadas de datos en plataformas heterogéneas a través de la red cliente-servidor. Proporciona el más alto rendimiento de procesar transacciones en línea (OLTP On-Line Transaction Processing) y el rendimiento decisión-soporte. También permite un alto grado de autonomía local y flexibilidad. Localidades remotas pueden actualizar información al momento, el servidor de replicación sincroniza base de datos con diferentes esquemas, formatos y convenciones de nombre. El suceso de replicación flexible permite notificación remota de sucesos por lo tanto cada sitio puede responder basado en necesidades locales.

El servidor de replicación diseña soportes "fail-through" de computación y arquitectura de aplicaciones que permite a los usuarios continuar su trabajo aunque los componentes del sistema no estén disponibles. Después de que el componente se cayo es levantado el servidor de replicación automáticamente y resincroniza cualquiera de las bases de datos que perdieron comunicación durante la interrupción de servicios.

Como la llave para desarrollar sistemas distribuidos no son bastos que se enfrentan con requerimientos de negocios críticos, el servidor de replicación resuelve varios de los más importantes problemas de los consumidores, que no son destinados por una arquitectura de base de datos.

El servidor de replicación te permite mantener un panorama completo de operaciones distribuidas, que está apegado al tiempo real aun cuando las unidades distribuidas de negocios operan en una variedad de hardware y plataformas DBMS.

El servidor te permite localizar datos donde se necesita, haciendo unidades de negocios distribuidas mucho menos susceptible a una computadora central o a poco tiempo de red mientras que reduce en general los costos de comunicación. También permite compartir datos bidireccionales con un seguro de acercamiento para la replicación de actualizaciones remotas.

Los sistemas de servidores permanecen eficientes a pesar de las fallas típicas de hardware, software y redes, entregando aplicaciones con mucha puntualidad a un costo razonable.

El servidor de replicación puede replicar una base de datos OLTP, permitiéndole a sus analistas poner a trabajar complejas consultas de decisión-soporte en los datos que se realiza en segundos de tiempo real sin afectar el rendimiento de OLTP.

El servidor te permite mantener una base de datos de "Warm-standby" aproximadamente al tiempo real a la que las aplicaciones pueden cambiar con puntualidad virtual si el sitio primario falla.

Las limitaciones de una base de datos distribuida.

Las bases de datos distribuidas son apropiadas para soporte de decisión o actualizaciones de bajo volumen a través de un pequeño número de localidades. Con esta arquitectura, cada unidad mantiene su propia y única base de datos. Pero cuando los usuarios necesitan consultar datos en sitios múltiples, las respuestas de tiempo son lentas. Cuando las consultas requieren de largos traslados, el rendimiento se afecta rápidamente. Además, las actualizaciones a sitios múltiples usualmente significa utilizar dos fases de protocolo commit, el cual requiere que todos los sitios estén levantados y corriendo para completar una transacción distribuida.

Aun cuando no haya falla, las bases de datos distribuidas son imprácticas debido a la complejidad y lo caro de la realización de dos faces, la cual aumenta geométricamente con el número de nodos.

Las fallas de red ocurren en cualquier sistemas distribuido. Pero con el servidor de replicación no afecta las operaciones.

El servidor de replicación Sybase entrega:

El servidor de replicación Sybase conoce todas las necesidades críticas de negocios. A través de cualquier número de localidades. Replica datos a cada sitio de procesamiento local. Su replicación de sucesos dirigidos y su mecanismo confiable de almacenamiento aseguran que las localidades remotas tengan información, es decir, tan cerca del tiempo real como sea posible. Facilita a las corporaciones consolidar información provenientes de sus varias unidades de negocios. Puede trabajar a través de fuentes de datos heterogéneos extrayendo todos los datos esparcidos en una organización.

Mientras que cada rama mantiene el control de sus datos primarios, algunos o todos los datos son replicados para corporar centros de operaciones, por lo tanto los usuarios a través de la organización pueden ver información corporada actualizada a cualquier hora.

La aplicación del cliente

Primero actualiza los datos primarios que el servidor de datos controla al sitio local. Entonces el agente de replicación notifica el servidor de replicación de actualizaciones de datos primarios que coordina la replicación de datos con otros sistemas de servidores de replicación.

Decisión de soporte que no afecta el funcionamiento de OLTP

Antes del servidor de replicación, las organizaciones enfrentaron una gran duda: aplicaciones de soporte-decisión e intensivas-consultas en datos actuales proporcionaban la información más exacta pero al costo de frenar sus aplicaciones OLTP comisión críticas a un nivel inaceptable. El servidor de replicación resuelve este dilema porque las aplicaciones soporte-decisión pueden correr en contra de sus replicas actualizadas de datos sin interrumpir el alto rendimiento de OLTP.

Entregando datos remotos a oficinas locales

Organizaciones con unidades de negocios distribuidos en un número de localidades diferentes necesitan permitir a oficinas locales trabajar con datos en otros sitios. Cada unidad retiene la copia primaria de sus propios datos y con el servidor de replicación cada unidad puede accesar y actualizar datos que pertecen de otras a otras unidades. Como el servidor de replicación utiliza procedimientos almacenados asincronicamente para actualizar datos primarios provenientes de estos sitios secundarios, entrega alto rendimiento mientras que disminuye el uso de recurso del sistema. Si ocurre interrupciones en el sistema, el servidor de replicación mantiene la consistencia de datos al procesar las transacciones pendientes después de que la interrupción de servicios se resuelve.

Operaciones continuas en el mundo real

Debido a que el servidor de replicación replica datos a sitios procesadores locales, las fallas en las redes y sistemas remotos raramente les preocupa a los usuarios. Los sitios pueden continuar operando en su copia local de los datos cuando la fuente de datos primarios no funciona. Muchas organizaciones utilizan el servidor de replicación para crear un sistema primario alternado a "warm standby". Aún cuando el sitio primario de datos deja de funcionar por un largo período, los usuarios continúan obteniendo la información que necesitan porque el sistema puede fallar sobre un primario alternado. Si un remoto sitio falla, este es automáticamente resincronizado con la fuente de datos primarios cuando vuelve a funcionar.

Sitios locales pueden usar y utilizar datos como quieran

El servidor de replicación manda datos rápidamente a suscriptores y sin restricciones. Localidades remotas pueden actualizar datos replicados y estas actualizaciones serán reflejadas a través de todas las copias de los datos en segundos. Cuando se actualizan datos, los sitios locales utilizan sus esquemas estándar, tipos de datos y convenciones y el servidor de replicación transforma transacciones en curso al tiempo que las manda a las localidades destino.

Inteligentes mecanismos de suscripción también permite a los usuarios remotos recibir información altamente construidas. Un Sql fácil de usar como lenguaje de escritura te permite describir criterios que determinan qué transacciones se debe mandar a cada localidad remota. Una dinámica ruta de transacción permite a las aplicaciones mandar por determinada ruta transacciones rápidamente, una característica particularmente útil para ambientes de circuito de producción. Replicación de sucesos permite a los usuarios especificar las condiciones basadas en sucesos de negocios. Cuando un suceso encuentra estos criterios, esta característica notifica la localidad remota que puede responder con base en las necesidades locales.

Principales atracciones técnicas

- Protege la integridad de transacciones y datos
- La replicación de transacciones mantiene la integridad transaccional de toda la información distribuida.
- Las reglas de negocios, cuando son reforzadas aseguran la calidad y la consistencia de datos.
- Sitios primarios tienen el control sobre sus datos.
- Trae los datos a tu servidor local.
- La nueva colocación de datos es transparente.
- Los datos son accesibles localmente y transparentemente asegurando la facilidad de acceso para los usuarios.
- Obtiene soporte completo para cualquier arquitectura flexible de aplicación, soporta una variedad de aplicaciones y estructuras organizadas.
- Las aplicaciones soporte-decisión no pueden saber del servidor de replicación.
- Administra fácilmente tu sistema

Asegura la entrega de datos confiables a todas las locaciones

Los sitios especifican los datos que necesitan, de abajo hacia la columna/fila, entonces se suscribe a esta para recibir actualizaciones regulares de información crítica. El

servidor de replicación asincrónicamente entrega transacciones completas para mantener la información actualizada a todos los sitios suscritos.

Construye tus datos para uso local

Usuario específico en transformaciones en curso permiten al sistema replicar datos entre sitios a pesar de las diferencias de esquema y tipo de datos.

Los usuarios tienen un rango amplio de opciones para especificar suscripciones: un sql como facilidad de suscripción permite a los usuarios especificar condiciones; la ruta de transacciones dinámica permite que las aplicaciones especifiquen la ruta de información al momento, suscripciones a parámetros de procesos almacenados notifican sitios remotos de eventos específicos de negocios.

El controlador de replicación permite el monitoreo, manejo y reparación de la llave de componentes del servidor de replicación a través de la empresa.

5.3 Informix

La replicación de datos es clave para la distribución efectiva y compartición de esta información. Pero la replicación es más que un movimiento de datos de sitio a sitio. Es una tecnología extremadamente solicitada. Y como tal, el buen funcionamiento de sistemas de replicación depende de tecnología que dirige el rango completo de negocios y necesidades de aplicación.

Los sistemas de replicación deben proporcionar gran rendimiento sin sobrecargar la fuente de base de datos y sin requerir cambios de aplicación.. Tienen que maximizar la disponibilidad del sistema y asegurar entrega consistente de los datos y finalmente los administradores de bases de datos necesitan ser capaces de configurar y manejar todos los componentes del sistema de replicación de una manera que utilice las fuentes de computación de el empresa de la mejor manera.

Replicación de empresa

Replicación de empresa, la solución avanzada de replicación de Informix proporciona una fundación de grandes empresas para distribuir y compartir información corporada. La replicación de empresa da soporte a las necesidades de replicación del grupo de trabajo a MPP de alta finalidad y grupos. Un número de factores clave caracterizan las distintas ventajas de las replicaciones de empresas que encuentran un espectro amplio de negocio y requerimientos de aplicación.

Alto rendimiento. La tecnología de Informix de industrias que dirigen servidores que proporcionan un rendimiento superior, habilidad y manejabilidad para aplicaciones de grandes empresas de gran volumen. Informix ha realizado esto al ser el primer vendedor de RDBMS para implementar replicación que obtiene proceso paralelo para encontrarse con los ambientes de computación más demandados.

Alta disponibilidad. Los productos de replicación de Informix aseguran alta disponibilidad a través de una variedad de opciones:

Todo proveniente de servidores hot standby a replicación asincrona de datos a uno o múltiples sitios. Debido a que la replicación es asincrona, nudos de interrupción de servicio de red y destino se toleran, eliminando cualquier punto de fracaso. Para que las replicaciones sean replicadas, incluyendo cambios al catalogo global ER, automáticamente se propagan a los sitios remotos cuando las fuentes son restauradas.

Integridad de transacción de nivel. La consistencia se mantiene al propagar inmediatamente cambios de bases de datos de la base de datos fuente a la base de datos destino. Las transacciones se replican de una cierta manera que mantiene el orden de transacción ER (Enterprise Replicación) también proporciona un número de opciones para detectar y resolver problemas de actualización.

Arquitectura flexible. ER proporciona una arquitectura de replicación que no impone ningún modelo particular o metodología en la empresa, permitiendo a las organizaciones definir sus ambientes de replicación basados en negocios específicos y requerimientos de aplicación. La implementación ER es transparente a aplicaciones con lo cual se evitan cambios de códigos. ER también apoya una variedad de modelos de uso de replicación incluyendo master/slave, actualizaciones de circuito de producción y actualizaciones en cualquier lugar.

Manejabilidad. Configuración y monitoreo se pueden ejecutar a través de un grupo de herramientas gráficas que se integran con la estructura de trabajo de controlador de sistema.

Arquitectura y ventajas

La introducción de replicación de empresa de Informix proporciona la solución de replicación más avanzada de la industria al mercado. Construida alrededor de la industria que maneja arquitectura de capacidad dinámica (Dynamic Scalable Architecture (DSA)), la replicación de empresa entrega una tecnología de la mejor clase mientras que evita las limitaciones de otras alternativas de replicación. ER da soporte a grupos de trabajo MPP. Informix diseño ER para dirigir el rango completo de requerimientos de negocios, incluyendo:

- Alto rendimiento
- Alta disponibilidad
- Integridad de transacción consistente y confiable
- Arquitectura flexible
- Manejabilidad y
- Fuente de soporte de datos heterogéneos

Alto rendimiento

La replicación de empresa de Informix está basada en DSA, que proporciona un alto rendimiento óptimo, plataforma de habilidad para dar soporte a las altas demandas de sistemas de replicación de datos de grandes empresas.

Al principio, Informix se dio cuenta de las limitaciones de la replicación basada en disparadores (triggers) y entonces optó por implementar una captura de transacción log-based (registro-basado) altamente eficiente y un menismo de distribución paralela que esta integrada con la arquitectura de base de datos. Informix ha sido capaz de implementar esta aproximación sin la degradación de rendimiento y la carga administrativa inherente con otros sistemas de log-based que utilizan servidores externos.

Además, DSA implementa una aproximación todo paralelo al explotar el poder de proceso paralelo inherente de SMP, ampliamente en pareja y arquitecturas MPP. El resultado es que todas las operaciones de replicación se realizan paralelamente, extendiendo más el rendimiento del sistema y minimizar cualquier obstáculo.

Alta disponibilidad

Para maximizar disponibilidad y tolerancia de fallas, la replicación de empresa de Informix toma en cuenta una base de datos central para ser replicada a un servidor secundario simple. Esta opción es útil para crear un servidor hot standby para tomar el poder de procesar en caso de fallas de servidores primarios.

La replicación de empresa de Informix también protege contra fallas de sistemas primarios a través de replicación asincrónica de datos a uno o a múltiples sitios. Cualquier actualización al sitio primario, incluyendo cambios al catálogo global ER, son automáticamente propagados a sitios secundarios asegurando que todos los sitios tengan replicaciones consistentes de los datos. La transmisión de actualizaciones pueden ser inmediatas o en un tiempo determinado, en tal caso el DBA especifica los tiempos de intervalo para las actualizaciones. Las replicaciones también pueden ser un evento controlado, tales como después de una transacción hecha o especificada por el usuario.

ER también utiliza un mecanismo de entrega de mensajes confiable que almacena datos localmente y propaga los datos al servidor remoto como una transacción separada. En el caso de una falla de servidor de red, el servidor local puede seguir dando servicio a los usuarios locales, proporcionando un alto grado de tolerancia de falta. Una vez que

el servidor o red afectados operan otra vez, todos los cambios a la base de datos de fuente son propagadas a las bases de datos remotas.

Integridad de transacción consistente y confiable

La replicación empresa mantiene copias múltiples y consistentes de copias de datos en diferentes sitios en un ambiente distribuido. Esto se alcanza por medio de cambios de bases de datos propagados asincronicamente que son resultado de una transacción al servidor destino, inmediatamente después que se realizan al servidor fuente. Además, las transacciones son almacenados para mantener consistencia de transacción y ordenamiento. Esto asegura que en el caso que el mismo registro sea actualizado múltiples veces los cambios serán aplicados a bases de datos remotos en el mismo orden consistente con en la base de datos primarios.

En el caso de cualquier problema de actualización, ER proporciona built-in (construcción interna) detección automática de conflictos y resolución, aunque exista una variedad de rutinas predefinidas o alternativamente a través de métodos de resolución adaptadas a las necesidades del cliente.

Arquitectura flexible

La replicación de empresa se encuentra un espectro amplio de negocios y requerimientos de aplicación. Primero da soporte al rango completo de modelos propiedad: master/slave, circuito de producción y actualización en cualquier lado. Después ER proporciona apoyo para la replicación de subgrupos y partición/división de tablas. ER también permite esquemas múltiples para servidores replicados, proporcionando DBAs con completa discreción al definir la sección de base de datos para ser replicados como hasta abajo de la fila y nivel de la columna. Esta flexibilidad capacita a DBAs para adoptar las bases de datos para necesidades específicas de negocios. Por ejemplo, los datos de operaciones pueden ser organizadas de manera diferente para transacción orientada de bases de datos opuestas a almacenes de datos que están designados para cuestionamiento y análisis.

Manejabilidad

De cualquier consola los DBAs pueden gráficamente configurar y monitorear el sistema replicación. Por ejemplo los DBAs pueden monitorear de manera gráfica todos los servidores de la red representados por iconos. El DBA puede seleccionar el tipo de método de replicación, designar el servidor destino, definir derechos de lectura-escritura, y especifica la frecuencia de replicación. Permite agregar y configurar nuevos servidores en línea sin tener que dar de baja el sistema entero.

Otra llave característica es el catálogo global replicado, el cual guarda configuración de información de pistas de ER. Un mayor beneficio de este catálogo para los DBAs es que automáticamente propaga cualquier cambio a las definición de replicación a los otros servidores en la red. Esto libra al DBA de copiar los datos manualmente a servidores remotos y minimiza potencialmente inconsistencias con el catalogo global.

Incluye la habilidad para monitorear el tamaño de la cola replicación para cada destinación, ve la disponibilidad de servidores remotos y realiza la notificación inmediata de conflictos entre los servidores de replicación y finalmente proporciona integración robusta de la utilidades administrativas con el Centro de Comando de Empresas Informix (IECC), el ambiente diseñado de administración orientado a objetos de Informix es específicamente para administrar base de datos distribuidas, protege a los DBAs de problemas que puedan ocurrir en la replicación.

Fuente de soporte de datos heterogéneos

Para proporcionar replicación heterogénea bidireccional de fuentes no Informix al Servidor Dinámico Informix-OnLine, Informix esta utilizando Praxis International OmniReplicator. Un rango amplio de fuentes no Informix son soportadas incluyendo SQLServer, Oracle7 y DB2.

5.4 Disbage

Como se ha notado en los manejadores anteriores ofrecen distribución de una manera eficaz, rápida y hasta cuenta con una alta seguridad en la información. La distribución como se observo también ofrece transparencia para los usuarios y la recuperación de los datos es flexible y es realizada en poco tiempo lo cual permite incrementar el rendimiento del sistema. También cuentan con la característica de actualizar la información de cada localidad sin interrumpir las operaciones en línea. Cuando se cae una parte del sistema es inmediatamente levantado sin tener que afectar a la otra parte del sistema.

Así como presentan ventajas estos manejadores también tienen sus desventajas, las desventajas que tiene un sistema distribuido. Todo esto funciona muy bien siempre y cuando se mantenga con poco número de localidades pues cuando es grande las respuestas se van alentando y el rendimiento del sistema se afecta notablemente.

El manejador que se apega más a los conceptos de un DDBMS es Oracle ya que además de ofrecer replicación también ofrece fragmentación aunque con ciertas limitaciones pues no permite hacerla de manera vertical y mixta solamente menciona que hace vistas particionadas horizontalmente pero nada más.

Disbage ofrece tanto replicación y fragmentación aunque se encuentra en desventaja en cuanto a los tipos de comunicación que se utilizan para la distribución que en la actualidad existen. Su arquitectura es la de un verdadero manejador distribuido pues cuenta con distribución confiable y segura. Se dice segura porque las bases de datos que se crean solo son controladas por el DBMS y cada usuario que cree sus relaciones tendrán todos los derechos a los que deseen y podrán dárselos a otros usuarios si así lo quieren, es decir, cada usuario tendrán control sobre sus propias relaciones y podrá replicarlas o fragmentarlas si así lo quiere.

Lo que destaca Disbage en cuando a los manejadores antes mencionados es que es un manejador de bases de datos que a su vez se encuentra almacenado en una base de datos, es decir, un meta base de datos, además maneja la opción de fragmentación en todas sus variedades lo cual no maneja ninguno de los anteriores.

6. CONCLUSIONES

Con la elaboración de cada uno de los capítulos se observó que es importante tener los datos de un forma distribuida. Existen manejadores que ya están usando el concepto distribuido pero no se ha definido de manera completa.

El análisis y diseño de DISBAGE y la información que se obtuvo demuestra que en esta tesis se pudo deducir que es necesario llevar a cabo un manejador de base de datos distribuido en su mayor énfasis y que aunque es un arquitectura costosa y compleja las empresas u organizaciones se han visto en la necesidad de ya tener su información distribuida, pues poco a poco se van incrementando el número de ellas y a su vez los datos que estás manejan. En este trabajo de investigación se observó que tener la información distribuida proporciona seguridad en los datos y mayor disponibilidad en la recuperación de éstos en donde más se necesita.

Se concluye que el análisis y diseño de la construcción del manejador distribuido DISBAGE permite que este se haga más completo de tal manera que llegue a ser un manejador distribuido totalmente confiable y seguro.

Y la metodología orientada a objetos que se utilizó aquí es la que permite realizarlo, pues lo que ya se tiene hecho se puede utilizar para mejorarlo en un futuro.

En el capítulo 5 se presentaron los grandes adelantos en lo que a distribución de datos se refiere, y el gran interés que afortunadamente las empresas desarrolladoras más importantes en el mundo de las bases de datos han puesto para su realización.

BIBLIOGRAFIA

1. Lipshutz, Seymour. Teoría de Conjuntos y Temas Afines.
   México, McGrawHill, 1986.

2. Özsu, Tamer. Principles of Distributed Database Systems.
   United States, Printice Hall, 1995.

3. Date.Introducción a Base de Datos.
   United States, Adison-Wesley/Díaz de Santos, 1990.

4. F. Korth, Henry y Silberschatz, Abraham. Fundamentos de Base de Datos.
   México, McGrawHill, 1991.

5. Booch, Grady. Object-Oriented Análysis and Design with Applications.
   Santa Clara, California; The Benjamin/Cummings Publishing Company, Inc;1994.

6. Mano, Morris M. Diseño Digital.
   México, Printice Hall, 1987.

7. Publicaciones en el Navegador de Internet.
   www.oracle.com.mx
   www.sybase.com.mx
   www.informix.com.mx

GLOSARIO

Abstracción: Denota las características esenciales de un objeto que lo distinguen de todas las demás tipos de objeto y proporciona así fronteras conceptuales nitidamente definidas respecto a la perspectiva del observador.

Archivo almacenado: Es el conjunto de todas las ocurrencias de uno o mas tipos de registros almacenados.

Atributo: Es una característica que describe una entidad y también se les conoce campo. Se dice que una entidad contiene uno o más atributos.

Atributo almacenado: Es la unidad de datos con nombre más pequeña que se halla almacenada en la bases de datos.

Clase: Representa solo una abstracción, la esencia de un objeto. Es un conjunto de objetos que comparten una estructura en común y un comportamiento común.

Dominio: El dominio de un atributo es el conjunto de todos los valores posibles que pueda haber y el atributo pueda tomar su valor entre todos ellos.

Herencia: Es una relación entre clases en la que una clase comparte la estructura y/o el comportamiento definidos en uno o más clase.

Jerarquía: Es una clasificación u ordenación de abstracciones.

Llave externa: Es la llave primaria de una entidad *x* y resulta ser el atributo de una entidad *y* sirven como campo de conexión entre la entidad *x* y *y*. Este llave como su nombre lo indica es un campo externo a la relación principal.

Llave primaria: (principal) Una llave o clave principal es la que identifica de manera única a una *nada* de la relación. Una llave puede estar formada de uno o más atributos esto depende, de si es que ninguno de los atributos identifica de manera única a cada registro de la relación.

Nada: Los elementos de la relación se llaman *nadas*. Cada *nada* representa la ocurrencia de un registro. Si una relación tiene *n* dominios entonces se dice que es de orden *n* y por lo tanto consta de un conjuntos de *nadas*.

Objeto: Es una entidad concreta que existe en el tiempo y en el espacio.

Ocurrencia o instancia de un registro almacenado:	Se compone de un grupo de ocurrencias de campos almacenados relacionados.
Relación:	Es un archivo conceptual que consta de ocurrencias con la misma composición de campo. Generalmente, una relación se pone como un archivo almacenado.
Subclase:	La clase que hereda de otra o más clases.
Superclase:	La clase de las que otras heredan.

INDICE DE FIGURAS