Universidad Central "Marta Abreu" de Las Villas
Facultad de Matemática, Física y Computación

TRABAJO DE DIPLOMA

Asistentes para la caracterización de aplicaciones y fragmentación en un Sistema Integrado de Ayuda al Diseño de Bases de Datos Distribuidas

Autor

Norma Elisa Cabrera González

Tutores

Msc. Abel Rodríguez Morffi

Dra. Luisa M. González González

2006

Hago constar que el presente trabajo fue realizado en la Universidad Central Marta Abreu de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencias de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

_____

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del tutor Firma del jefe del Seminario

Resumen

El diseño de una Base de Datos Distribuida cuenta con las mismas fases que el de una centralizada y a éstas se les añaden dos nuevos problemas: la fragmentación y la asignación de los fragmentos a los sitios. El diseño de la fragmentación es el primer problema que debe enfrentar la distribución de datos y su propósito es obtener porciones no solapadas de la Base de Datos que sean unidades lógicas de asignación.

Existe actualmente un déficit de herramientas de apoyo al diseño distribuido siendo nuestro objetivo la creación de herramientas de ayuda que mitigue esta carencia. Se implementaron dos asistentes como materiales de ayuda para diseño de bases de datos distribuidas. Uno para captar la información que caracteriza a las aplicaciones, y otro que guía el proceso de diseño de la fragmentación.

Abstract

The design of a distributed database has the same phases that the centralized databases. But, they have two new problems: the fragmentation and the fragments assignment to the places. The design of the fragmentation is the first problem that the data distribution should face and its purpose is to obtain not overlapped portions of the database that are logical units of assignment.

There is a deficit of support tools to the distributed design. The objective of this work is the creation of support tools that mitigates this lack. Two assistants were implemented as help materials for distributed databases design. One to take the information that characterizes to the applications, and another that guides the fragmentation design process.

Índice

Capítulo 1. DISEÑO DE BASES DE DATOS DISTRIBUIDAS 5
1.1 Aspectos generales del diseño 5
1.1.1 Introducción al Diseño de Bases de Datos Distribuidas 5
1.1.2 Estrategias para el diseño distribuido 6
1.1.3 Resultados del diseño de distribución 9
1.2 El Problema de la fragmentación 10
1.2.1 Razones para fragmentar 10
1.2.2 Alternativas de fragmentación 11
1.2.3 Reglas para la fragmentación 13
1.3 Diseño de la fragmentación 14
1.3.1 Información requerida 14
1.3.2 Diseño de la fragmentación horizontal 16
1.3.2.1 Fragmentación horizontal primaria 17
1.3.2.2 Fragmentación horizontal derivada 18
1.3.3 Diseño de la fragmentación vertical 20
1.3.4 Diseño de la fragmentación mixta 23
1.3.5 Conclusiones parciales 25
Referencias bibliográficas 27
Capítulo 2. SOLUCIÓN DE LA FRAGMENTACIÓN 29
2.1 Fragmentación horizontal 29
2.1.1 Algoritmo para la fragmentación horizontal primaria 29
2.1.1.1 Conjunto de predicados simples mínimo y completo 33
2.1.1.2 Minterms e implicaciones lógicas 36
2.1.2 Correctitud de la fragmentación horizontal primaria 36
2.1.3 Algoritmo para la fragmentación horizontal derivada 37
2.1.4 Correctitud de la fragmentación horizontal derivada 39
2.2 Fragmentación Vertical 40
2.2.1 Algoritmo de Hoffer y Severance 41
2.2.2 Partición vertical binaria 43
2.2.3 Algoritmo basado en una función objetivo empírica 44
2.2.4 Correctitud de la fragmentación vertical 46
2.3 Conclusiones parciales 47
Referencias bibliográficas 48
Capítulo 3. DISEÑO E IMPLEMENTACIÓN DE LOS ASISTENTES 49
3.1 Casos de Uso del sistema 49
3.2 Asistente para la caracterización de las aplicaciones 50
3.2.1 Diagrama de clases 51
3.3 Asistente para la fragmentación 53
3.3.1 Diagrama de clases 53
3.4 Casos de prueba 56
3.4.1 Ejemplo de fragmentación horizontal 56

3.4.2 Ejemplo de fragmentación vertical 59
3.5 Vista del usuario 61
3.5.1 Asistente para la caracterización de aplicaciones 61
3.5.2 Asistente para la fragmentación de los esquemas 63
3.6 Conclusiones parciales 64
Conclusiones 65
Recomendaciones 66
Anexo 1 68

Introducción

En los últimos años la disponibilidad de las bases de datos y de las redes de computadoras ha promovido el desarrollo de un nuevo campo denominado bases de datos distribuidas. Éstas son bases de datos integradas que se construyen por encima de una red de computadoras en lugar de un sitio de cómputo. Esto no implica problema alguno a los usuarios, para ellos es transparente el acceso y localización de la información. Sin embargo, el diseño y administración de bases de datos distribuidas constituye un gran desafío que incorpora problemas no encontrados en bases de datos centralizadas. Por ejemplo, algunos de estos problemas son los modelos de fragmentación y localización de información, el manejo de consultas a sitios distribuidos y los mecanismos de control de concurrencia y confiabilidad.

Hay buenas razones técnicas para distribuir datos. La más obvia es la referente a la sobrecarga de los canales de entrada y salida a los discos en donde se almacena finalmente la información. Es mucho mejor distribuir los accesos a la información sobre diferentes canales que concentrarlos en uno solo. Otra razón de peso es que las redes de computadoras empezaron a trabajar a velocidades razonables abriendo la puerta a la distribución del trabajo y la información.

El hacer una descentralización de la información se justifica desde el punto de vista tecnológico por las siguientes razones (Öszu and Valduriez, 1999):

- Para permitir autonomía local y promover la evolución de los sistemas y los cambios en los requerimientos de usuario.
- Para proveer una arquitectura de sistemas simple, flexible y tolerante a fallas.
- Para ofrecer buenos rendimientos.

Existen aplicaciones que nacieron distribuidas; para ellas ha sido necesario el uso de nuevas tecnologías con el objetivo de integrar sistemas de información diferentes; de forma que, no se afecte de manera sustancial el estilo de trabajo de los usuarios.

Aunque la idea de distribución de datos es bastante atractiva, su realización conlleva la superación de una serie de dificultades tecnológicas entre las que se pueden mencionar (Öszu and Valduriez, 1999):

- Asegurar que el acceso entre diferentes sitios o nodos y el procesamiento de datos se realice de manera eficiente, presumiblemente óptima.
- Distribuir datos en los nodos del ambiente distribuido de una manera óptima.
- Controlar el acceso a los datos disponibles en el ambiente distribuido.
- Soportar la recuperación de errores de diferentes módulos del sistema de manera segura y eficiente.
- Asegurar que los sistemas locales y globales permanezcan como una imagen fiel del mundo real evitando la interferencia destructiva que pueden ocasionar diferentes transacciones en el sistema.

La fragmentación de los esquemas es una etapa dentro del diseño de bases de datos distribuidas donde se crean fragmentos de una relación de forma tal que se minimice el costo de acceso a los datos durante el procesamiento de las transacciones, mejorando los tiempos de respuesta y el paralelismo del sistema. Este proceso puede realizarse de varias maneras; existen dos alternativas fundamentales y básicas: la fragmentación horizontal y la vertical. Ambas necesitan tanto de información relativa a la base de datos como de las aplicaciones.

El esquema de fragmentación debe cumplir con dos propiedades: *completitud y reconstrucción*. La primera requiere que todos los datos de la relación global se encuentren en algún fragmento, de forma tal que no halla pérdida de datos. La segunda por su parte implica que la relación global pueda ser reconstruida a partir de sus fragmentos.

Con el objetivo de lograr la máxima localidad de los datos, o sea, tener en cada estación de trabajo aquellos fragmentos más usados, no basta con tener fragmentos horizontales o verticales. En ocasiones es ventajoso contar con los llamados fragmentos mixtos o híbridos que no es más que la combinación de estas dos alternativas.

Se han implementado algunas herramientas para la fragmentación de esquemas globales de una base de datos. En (Artiles, 2000) se habla del desarrollo de una herramienta para la caracterización de aplicaciones y fragmentación horizontal. En (Morell, 1999) se hace un estudio de la fragmentación vertical y se menciona la implementación de una aplicación para lograr dicho propósito.

Como puede verse estos dos trabajos se basan en un solo tipo de fragmentación. Se hace necesario la implementación de una herramienta que combine estas alternativas, permitiendo el uso de cualquiera de ellas: horizontal, vertical o mixta. Por todo esto se tiene como **objetivos generales** de este trabajo:

- Diseñar e implementar un asistente para la fragmentación vertical, horizontal y la combinación de ellas para el diseño de bases de datos distribuidas.
- Implementar un asistente para la caracterización de aplicaciones.

Los **objetivos específicos** que se trazan en este proyecto son:

1. Estudiar los principales aspectos relacionados con el diseño de bases de datos distribuidas; profundizando en la etapa de diseño de la fragmentación.
2. Diseñar un módulo para la fragmentación vertical, horizontal y mixta en un sistema de ayuda para el diseño de bases de datos distribuidas.
3. Implementar un asistente para guiar el proceso de diseño de la fragmentación.
4. Implementar un asistente para caracterizar las aplicaciones.

Este trabajo se encuentra estructurado en tres capítulos: en el primero se hace referencia a las cuestiones teóricas que influyen en el diseño de bases de datos distribuidas, haciendo énfasis en la etapa de diseño de la fragmentación. El segundo muestra una descripción detallada de la algoritmización de las diferentes alternativas de fragmentación, lo cual fundamenta su posterior implementación. En el tercero se explican los aspectos generales

del diseño y la implementación de los asistentes, ejemplos de prueba y los resultados obtenidos.

Capítulo 1. DISEÑO DE BASES DE DATOS DISTRIBUIDAS

En este capítulo se introducen los conocimientos básicos para el diseño de Sistemas de Bases de Datos Distribuidas (SBDD), particularizando en el estudio de la fragmentación como una etapa esencial dentro de la creación de Sistemas Distribuidos.

1.1 Aspectos generales del diseño

El advenimiento de la era Internet y el crecimiento no planeado de la información dentro de las organizaciones, ha traído como consecuencia la dispersión de los datos en localidades geográficamente distantes. La necesidad de integrar y compartir dicha información, implica el nacimiento de una nueva tecnología capaz de administrar de manera consistente la información de las organizaciones.

1.1.1 Introducción al Diseño de Bases de Datos Distribuidas

Los sistemas distribuidos se adaptan mejor a las necesidades de las organizaciones descentralizadas ya que reflejan adecuadamente su estructura y les brindan numerosas ventajas, principalmente en el orden económico (Özsu y Valduriez, 1999), pero al mismo tiempo añaden nuevas complejidades en su diseño e implementación.

Se han dado varias definiciones de lo que es un Sistema Distribuido, pero no todas están de acuerdo entre sí. Para los propósitos de este trabajo, es suficiente la siguiente caracterización: Un Sistema Distribuido es una colección de computadoras independientes que aparece ante los usuarios del sistema como una única computadora. Todo el desarrollo de la tecnología de Bases de Datos ha estado caracterizado por lograr altos grados de independencia entre los programas de aplicación y los aspectos internos (de manipulación, representación, etc.), también es deseable lograr transparencia acerca de la distribución de los datos en la red.

Se puede definir una Base de Datos Distribuida (BDD) como una colección de datos que pertenecen lógicamente a un solo sistema, pero se encuentra físicamente esparcida en varios sitios de la red, es decir, se tiene un conjunto de sitios, conectados entre sí mediante algún tipo de red de comunicaciones, en el cual cada sitio es un Sistema de Base de Datos (SBD) en sí mismo y trabajan juntos (si es necesario) con el fin de que un usuario pueda obtener acceso a los datos de cualquier punto de la red tal como si estuvieran almacenados en el propio sitio del usuario.

El diseño de un SBDD implica la toma de decisiones sobre la ubicación de los programas que accederán a la Base de Datos y sobre los propios datos que constituyen esta última, a través de los diferentes sitios de una red de computadoras.

1.1.2 Estrategias para el diseño distribuido

Para el diseño de BDD han sido identificadas dos grandes estrategias: el enfoque ascendente (Bottom-Up) y el enfoque descendente (Top-Down). La estrategia ascendente es recomendable cuando existen diferentes bases de datos locales y la tarea de diseño consiste en integrarlas y constituir el esquema conceptual global. La estrategia descendente es adecuada cuando un sistema de Bases de Datos se construye por primera vez sin tener restricciones de otros sistemas ya instalados y que deban ser integrados de alguna manera al Sistema Distribuido.

Ambos tipos de estrategia no son excluyentes, y no resultaría extraño a la hora de abordar un trabajo real que se pudiesen emplear en diferentes etapas del proyecto una u otra estrategia. Aunque la estrategia ascendente se pueda presentar con facilidad en la vida real, se prefiere pensar en el caso donde se parte de cero y se avanza en el desarrollo del trabajo siguiendo la estrategia descendente.

En lo subsiguiente se enfocará el presente trabajo a la estrategia descendente (véase la figura 1.1) donde el proceso de diseño incluye las siguientes etapas:

Análisis de requisitos. Incorpora la colección de especificaciones de datos y procesos necesitados por los usuarios potenciales de la base de datos. Esta información es almacenada en un diccionario de datos. El estudio de requisitos especifica además, hasta

donde pretende llegar el sistema final con respecto a los objetivos del sistema de gestión de bases de datos distribuidas.

Diseño de vistas. Se definen las interfaces para los usuarios finales. Esta fase soporta las aplicaciones de usuarios y los accesos a la base de datos definidas en el esquema conceptual global.

Diseño conceptual. Produce una especificación conceptual global del esquema de datos y las aplicaciones que son ejecutadas en ellos, así como información estadística sobre las mismas. Esta etapa puede ser interpretada como una integración de las vistas de usuario, la cual asegura que las entidades y relaciones requeridas para todas las vistas, estén plasmadas en el esquema conceptual global.

Diseño de distribución. Requiere para su desarrollo la información resultante del esquema conceptual global, los modelos de accesos y las definiciones del esquema externo. El objetivo del mismo es obtener el esquema conceptual local distribuyendo las entidades en los sitios del sistema distribuido. Es muy ventajoso dividir las entidades en pequeñas porciones llamadas fragmentos, los cuales son posteriormente distribuidos. El diseño de distribución puede dividirse en dos actividades: fragmentación y localización.

Diseño físico. Se encarga de ubicar los resultados del esquema conceptual local en los dispositivos físicos de almacenamiento disponibles en los sitios. Este proceso necesita también la información de los modelos de acceso de cada fragmento.

Observación y monitoreo. El diseño es completado con una etapa de monitoreo que chequea el comportamiento de la implementación de la base de datos y la integridad de las vistas de usuario.

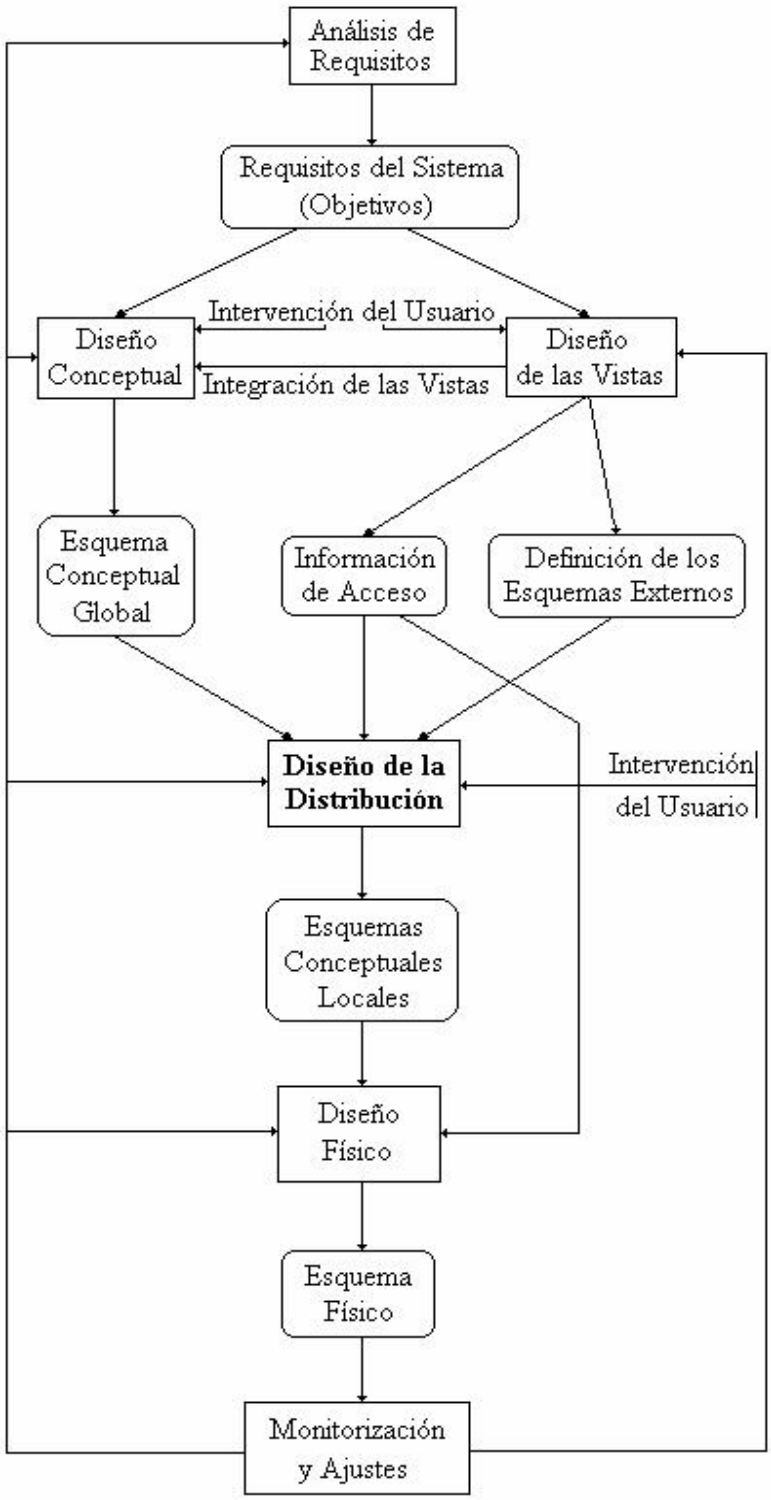

Figura 1.1 *Estrategia descendente(Özsu y Valduriez, 1999))*

1.1.3 Resultados del diseño de distribución

Un aspecto importante que se debe considerar en la etapa de diseño es cómo los datos y las aplicaciones que se ejecutan en ellos deben ser asignados a los sitios de la red. Existen dos alternativas para colocar datos: particionados (no replicados) y replicados. El particionamiento consiste en dividir la base de datos en varias porciones disjuntas, cada una de las cuales es ubicada en un sitio. Por su parte el esquema replicado puede dividirse en parcialmente replicado, donde cada partición de la base de datos es almacenada en más de un sitio pero no en todos, y completamente replicado donde cada partición se almacena completamente en todos los sitios.

Los resultados fundamentales de esta etapa son la fragmentación, que consiste en separar la base de datos en particiones llamadas fragmentos, y la distribución óptima de esos fragmentos en los sitios de la red. Durante el proceso se deben tener en cuenta los siguientes objetivos:

Procesamiento local. La distribución de los datos, para maximizar el procesamiento local corresponde al principio simple de colocar los datos tan cerca como sea posible de las aplicaciones que los utilizan. Se puede realizar el diseño de la distribución de los datos para maximizar el procesamiento local agregando el número de referencias locales y remotas que le corresponden a cada fragmentación candidata y la localización del fragmento, que de esta forma se seleccione la mejor solución de ellas.

Distribución de la carga de trabajo. La distribución de la carga de trabajo sobre los sitios, es una característica importante de los sistemas de cómputo distribuidos. Esta distribución de la carga se realiza para tomar ventaja de las diferentes características (potenciales) o utilizaciones de las computadoras de cada sitio, y maximizar el grado de ejecución de paralelismo de las aplicaciones. Sin embargo, la distribución de la carga de trabajo podría afectar negativamente el procesamiento local deseado.

Costo de almacenamiento y disponibilidad. La distribución de la base de datos refleja el costo y disponibilidad del almacenamiento en diferentes sitios. Para esto, es posible tener sitios especializados en la red para el almacenamiento de datos. Sin embargo el costo de almacenamiento de datos no es tan relevante si éste se compara con el de tiempo de procesamiento, entrada-salida y costos de transmisión de las aplicaciones.

En los siguientes epígrafes se profundizará y discutirá los aspectos esenciales del proceso de fragmentación.

1.2 El Problema de la fragmentación

El diseño de la distribución tiene como principio clave alcanzar máxima localidad de datos y aplicaciones, ubicando los datos tan cerca como sea posible de las aplicaciones que los utiliza, lo que permite reducir el tráfico de comunicaciones en la red. En un sistema bien diseñado el 90 % de los datos deben ser ubicados localmente y solo el 10 % debe ser accedido remotamente (Ceri et al., 1989). La fragmentación refleja el criterio de mantener los datos locales en el sitio donde frecuentemente son accedidos por las aplicaciones, es por ello que constituyen una unidad apropiada de asignación.

1.2.1 Razones para fragmentar

El principal problema de la fragmentación radica en encontrar la unidad apropiada de distribución. Una relación no es una buena unidad por muchas razones.

Primero, las vistas de la aplicación normalmente son subconjuntos de relaciones. Además, la localidad de los accesos de las aplicaciones no está definida sobre relaciones enteras pero sí sobre subconjuntos de las mismas. Por ello, sería normal considerar como unidad de distribución a estos subconjuntos de relaciones.

Segundo, si las aplicaciones tienen vistas definidas sobre una determinada relación (considerándola ahora una unidad de distribución) que reside en varios sitios de la red, se puede optar por dos alternativas. Por un lado, la relación no estará replicada y se almacena en un único sitio, o existe réplica en todos o algunos de los sitios en los cuales reside la aplicación. Las consecuencias de esta estrategia son la generación de un volumen de accesos remotos innecesario. Además, se pueden realizar réplicas no esenciales que causen problemas en la ejecución de las actualizaciones y puede no ser deseable si el espacio de almacenamiento está limitado.

Tercero, la descomposición de una relación en fragmentos, tratados cada uno de ellos como una unidad de distribución, permite el proceso concurrente de las transacciones.

Se debe precisar también que la fragmentación trae consigo algunos inconvenientes, si las aplicaciones presentan requisitos tales que prevengan la descomposición de la relación en fragmentos mutuamente excluyentes. Estas aplicaciones cuyas vistas estén definidas sobre más de un fragmento pueden sufrir una degradación en el rendimiento. Por tanto, puede ser necesario recuperar los datos de dos fragmentos y llevar a cabo sobre ellos operaciones costosas de unión y acople. Además como resultado de la fragmentación los atributos implicados en una dependencia se descomponen en diferentes fragmentos que pueden destinarse a sitios diferentes. En este caso, la sencilla tarea de verificar las dependencias puede resultar en una búsqueda de los datos implicados en un gran número de sitios.

1.2.2 Alternativas de fragmentación

Un sistema soporta fragmentación de datos si es posible dividir sus relaciones en pequeñas porciones o fragmentos. Existen, esencialmente, dos formas de realizar esta división: horizontal o verticalmente. Podemos hablar también de otro tipo de fragmentación que es generada por la combinación de fragmentos horizontales y verticales y es nombrada fragmentación mixta o híbrida. Aún cuando esta fragmentación no se considere un tipo primitivo, en muchas situaciones reales resulta imprescindible disponer de ella.

Una decisión compleja e importante que afecta el rendimiento de las aplicaciones, es determinar cuál relación debe ser fragmentada. Por supuesto que ésta no es una decisión arbitraria, en este sentido influyen determinados parámetros que deben caracterizar tanto a las aplicaciones como a la base de datos, pero es necesario resaltar que el grado de la fragmentación puede ir de un extremo en que no se fragmenta del todo, a otro donde se fragmenta en el ámbito de atributos individuales o al nivel de tuplas (para el caso de la fragmentación vertical y horizontal respectivamente) (Özsu y Valduriez, 1999).

Los efectos adversos de fragmentar unidades muy grandes pueden influir negativamente en relación con la réplica. Si el fragmento no es replicado se incrementan desmesuradamente los accesos remotos, y si es replicado en todos los sitios causa serios problemas con la actualización de los datos y es necesario incrementar la capacidad de almacenamiento de las computadoras personales. Por otra parte, si es necesario recuperar datos ubicados en más de un fragmento, la fragmentación de unidades muy pequeñas provoca el uso de

operadores de unión o acople (*join*) lo cual es muy costoso, y además, el problema de la localización puede resultar inmanejable. Por tal motivo el diseñador debe encontrar un nivel de fragmentación adecuado que será un compromiso entre ambos extremos. Tal nivel puede ser definido con respecto a las características de las aplicaciones que se ejecutan sobre la base de datos.

La fragmentación mixta puede realizarse de tres formas diferentes (véase la figura 1.2):

- Desarrollando primero la fragmentación vertical y luego aplicar la fragmentación horizontal sobre los fragmentos verticales (llamada partición VH).
- Desarrollando primero una fragmentación horizontal y luego aplicar la fragmentación vertical sobre los fragmentos horizontales (llamada partición HV).
- Aplicando sobre una relación, de forma simultánea y no secuencial la fragmentación horizontal y vertical, generando una rejilla y los fragmentos formaran las celdas de dicha rejilla.

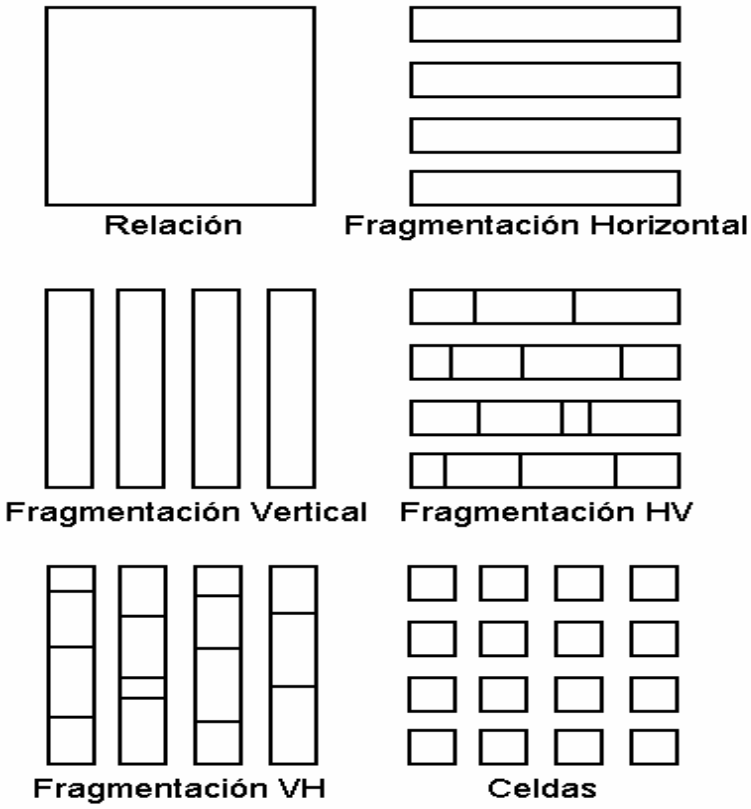

Figura 1.2 *Esquema de la Fragmentación*

1.2.3 Reglas para la fragmentación

Existen tres reglas que debe cumplir el proceso de fragmentación, las cuales asegurarán la ausencia de cambios semánticos en la base de datos durante el proceso, logrando así una fragmentación correcta.

Completitud. Dada una relación R y su descomposición en una serie de fragmentos (R1, R2,…, Rn) se considera completa si y solamente si cada elemento de datos en R se encuentra en uno o varios fragmentos Ri. Esta propiedad asegura que los datos de la relación global se proyectan sobre los fragmentos sin pérdida alguna. En el caso de la fragmentación horizontal el elemento de dato, normalmente, es una tupla, mientras que en el caso vertical es un atributo.

Reconstrucción. Si una relación R se descompone en una serie de fragmentos (R_1, R_2,…, R_n), la relación R debe poderse reconstruir a partir de sus fragmentos, es decir, se puede definir un operador relacional ∇ tal que: $R = \nabla R_i$, $\forall R_i \in FR$. El operador ∇ será diferente dependiendo de las diferentes formas de fragmentación. La reconstrucción de la relación a partir de sus fragmentos asegura que las restricciones definidas sobre los datos en forma de dependencias sean preservadas.

Disyunción. Si una relación R se descompone horizontalmente en una serie de fragmentos (R_1, R_2,..., R_n), y un elemento de datos di se encuentra en algún fragmento R_j, entonces no se encuentra en otro fragmento R_k ($k \neq j$). Esta regla asegura que los fragmentos horizontales sean disjuntos. Si una relación R se descompone verticalmente, sus atributos primarios clave normalmente se repiten en todos sus fragmentos.

Para la fragmentación vertical esta condición es definida solamente para los atributos que no forman parte de la llave primaria pues ésta es repetida en todos los fragmentos. Este requisito puede tener excepciones, y es el caso en el que se asignan identificadores de tuplas en la relación global y esos identificadores se mantienen para todos los fragmentos.

1.3 Diseño de la fragmentación

El propósito del diseño de la fragmentación es determinar los fragmentos que constituyen unidades lógicas de asignación y consiste en agrupar tuplas o atributos con las mismas propiedades. Los fragmentos deben ser colecciones homogéneas de información desde el punto de vista de accesos de las aplicaciones, o sea, que todas las instancias de los fragmentos sean accedidas uniformemente por las aplicaciones que las usan (Ceri et al., 1989).

1.3.1 Información requerida

Antes de comenzar el proceso de fragmentación el diseñador debe poseer información de la base de datos e información de las aplicaciones (Ceri y Pelagatti, 1984), (Ceri et al., 1989), (Özsu y Valduriez, 1999).

La información de la base de datos es la concerniente al esquema conceptual global, en el cual es necesario representar las relaciones, sus atributos y la conexión entre relaciones.

La información de las aplicaciones debe representarse en términos cualitativos y cuantitativos. La cualitativa guía la actividad de fragmentación, mientras que la información cuantitativa se incorpora fundamentalmente a los modelos de localización.

La principal información cualitativa describe los predicados que satisfacen las aplicaciones de los usuarios. Dichos predicados deben guiar la localidad de las aplicaciones, es por ello que se definen en base a predicados elementales o simples que los usuarios suministran para indicar las agrupaciones físicas de los datos que se presentan para las diferentes aplicaciones y luego estos predicados simples se combinan adecuadamente para reflejar, por último, la localidad integrada del grupo de aplicaciones.

Los predicados simples p_j son expresiones formadas por las cláusulas:

p_j : < A_i > <operador> <Valor>

Donde <A_i> es un atributo que pertenece a la relación R y debe estar definido sobre un dominio D_i, <operador> $\in$ {=, ≠, <, ≤, >, ≥} y <valor> es seleccionado del dominio de A_i (Valor $\in D_i$).

Frecuentemente las solicitudes de usuarios requieren predicados más complejos que pueden expresarse como combinaciones boolenas de predicados simples. Los predicados *minterm* son un tipo de combinación que puede definirse como la conjunción de todos los predicados simples tanto en su forma natural como negada (Özsu y Valduriez, 1999).

Así, para un conjunto de predicados simples $Pr_i = \{p_{i1}, p_{i2},..., p_{im}\}$, el conjunto de predicados *minterm* $M_i = \{ m_{i1}, m_{i2},..., m_{iz} \}$ se define como:

$M_i = \{ m_{ij} \mid m_{ij} = \Lambda \; p_{ik}^{*}\}$, $1 \leq k \leq m$, $1 \leq j \leq z$ donde $p_{ik}^{*} = p_{ik}$ o $p_{ik}^{*} = \rceil p_{ik}$.

$$p_{ik} \in Pr_i$$

Los predicados de selección o predicados simples, a partir de los cuales se hallan los predicados *minterm,* que determinan el proceso de fragmentación deben poseer dos propiedades fundamentales para garantizar una fragmentación correcta y eficiente: *completitud* y *minimalidad*.

Un conjunto de predicados Pr es completo si garantiza, para cada aplicación del conjunto de aplicaciones, que dos tuplas cualesquiera de un mismo fragmento sean accedidas con igual probabilidad. Esta propiedad permite que todos los elementos de un fragmento sean referenciados de forma homogénea por cada aplicación.

Por su parte la minimalidad es una propiedad mucho más intuitiva y generalmente se apela a la experiencia y sentido común del diseñador más que al empleo de técnicas formales. El conjunto de predicados Pr es minimal si todos los predicados son relevantes en la determinación de la fragmentación, o sea, si para cada predicado p_j, que causa que la relación R se fragmente en R_i y R_j, existe al menos una aplicación que accede de forma diferente a ambos fragmentos. Se puede afirmar que un conjunto minimal es siempre completo, sin embargo lo contrario no siempre es verdadero.

Con respecto a la información cuantitativa acerca de las aplicaciones es necesario conocer:

1. Selectividad del *minterm*. Número de tuplas de la relación que pueden ser accedidas por una solicitud especificada acorde a un predicado *minterm* dado.
2. Frecuencia de acceso. Frecuencia con la que las aplicaciones acceden a los datos, o sea, frecuencia de activación de la aplicación en cada sitio.

3. El tipo de acceso (lectura o escritura) realizado por la aplicación a cada objeto de datos.

1.3.2 Diseño de la fragmentación horizontal

La fragmentación horizontal de una relación R es la subdivisión de sus tuplas en subconjuntos disjuntos llamados fragmentos. Los datos contenidos en cada subconjunto deben poseer "propiedades geográficas" comunes y contener todos los atributos de la relación original. Las propiedades geográficas identifican las agrupaciones de datos que se tratan en cada sitio donde debe residir la base de datos distribuida. Su principal objetivo es producir fragmentos con máxima localidad respecto a las aplicaciones.

La posibilidad de no particionar horizontalmente una entidad debe ser considerada en dependencia de las aplicaciones que la accedan. Si el beneficio observado por fragmentar una entidad se ve limitado, el costo debido al particionamiento puede no ser compensado por la ventaja del procesamiento local (Ceri et al., 1989).

Existen varios criterios que permiten descartar la posibilidad de particionar una entidad:

1. No existan predicados que particionen esa entidad, esto puede estar dado porque no existan aplicaciones que referencien ese conjunto de datos de forma diferenciada en los sitios que procesen la entidad.
2. Se generen fragmentos que no sean unidades apropiadas de localización.
3. No se generen predicados completos y minimales que representen una fragmentación correcta, esto se fundamenta en las características de las aplicaciones.
4. Se distingan fragmentos de una relación con rasgos muy similares.

Se han realizado numerosos estudios sobre el tema de la fragmentación horizontal, esto ha contribuido al surgimiento de varias metodologías. Los mejores algoritmos que existen para realizarla son iterativos. Algunos ejemplos de estas metodologías y algoritmos son los siguientes:

En (Özsu y Valduriez, 1999) se describe un algoritmo iterativo que realiza la fragmentación horizontal de una relación usando los minterms, formados a partir de

predicados simples extraídos de las consultas del usuario. Son consideradas dos versiones de fragmentación horizontal: primaria y derivada.

En (Navathe et al., 1995) se describe un algoritmo no iterativo, basado en teoría de grafos. El mismo utiliza una matriz de afinidad de los predicados simples, y los fragmentos finales son definidos procesando la clusterización de los predicados simples.

En (Ezeife y Barker, 1995) se describe un conjunto de algoritmos para fragmentar horizontalmente cuatro modelos de clases diferentes en un sistema distribuido basado en objetos.

En (Zhou et al., 1994) se describe un algoritmo de clausura transitiva paralela para fragmentar datos en bases de datos deductivas.

En (Pramanik et al., 1992) se proponen dos estrategias para la fragmentación horizontal de relaciones recursivas, basado en el concepto de conjuntos de orden parcial.

1.3.2.1 Fragmentación horizontal primaria

La fragmentación horizontal primaria es muy común en la realidad y se aplica para lograr un incremento del paralelismo, posibilita decrementar el tiempo de respuesta de las solicitudes de usuarios y disminuir el costo de procesamiento de las solicitudes locales. Se realiza de acuerdo a un predicado que determina qué tuplas de la relación serán almacenadas en cada sitio y se define por una operación de selección en la relación propietaria del esquema global (cabe aclarar que el modelo de datos utilizado es un modelo lógico de enlaces).

En el diseño del esquema de fragmentación se sugiere que la cantidad de fragmentos disjuntos sea mayor o igual que la cantidad de sitios en la red. Esta facilidad simplifica los modelos de optimización y es justificable por el hecho de que en las aplicaciones reales el usuario asocia los fragmentos candidatos a los sitios de localización de manera natural.

Para construir el esquema de fragmentación primaria pueden seguirse tres pasos, el primero de ellos consiste en encontrar los predicados simples relevantes a cada aplicación asegurando que los mismos sean completos y minimales. La búsqueda de los predicados debe realizarse de forma exhaustiva concentrándose solamente en las aplicaciones más importantes y sin distinguir los fragmentos cuyas propiedades sean similares.

Existen controversias entre diversos autores con respecto a la determinación de los predicados simples. Para unos (Ceri y Pelagatti, 1984), basta con incluir en el conjunto de

predicados, los predicados p_j en su forma natural capaces de fragmentar una relación en dos porciones. Otros, por su parte (Özsu y Valduriez, 1999), consideran en el conjunto de predicados, los predicados simples en su forma natural y también los que están de forma negada (principalmente para predicados cuyo dominio es numérico). Esta última alternativa añade complejidad a los predicados *minterm* debido a que aumenta considerablemente la cantidad de predicados simples útiles para formar los predicados *minterm* y como resultado se obtendrían muchos *minterm* sin significado. Además de esta forma los predicados simples pierden relevancia a la vista del diseñador, porque es como si se consideraran dos predicados para particionar las tuplas de la relación R en dos partes.

Las propiedades de completitud y minimalidad son satisfechas por ambas variantes, pero se debe prestar especial atención cuando se utilice la segunda variante porque el predicado p_i causa que la relación R se fragmente en R_1 y R_2, por tanto $\neg p_i$ debe aplicársele al fragmento resultante que no cumpla la condición de selección impuesta por p_i.

El segundo paso consiste en construir el conjunto de predicados *minterm* acorde al conjunto de predicados simples. Los predicados *minterm* determinan los fragmentos candidatos en el proceso de localización.

El tercer y último paso del proceso de diseño es la eliminación de algunos predicados *minterm* que resultan insignificantes o contradictorios para un conjunto de implicaciones I (Özsu y Valduriez, 1999), que en cierta medida constituyen restricciones de integridad del sistema.

1.3.2.2 Fragmentación horizontal derivada

La fragmentación horizontal derivada es el concepto de particionar un objeto aplicándole la misma partición que se le aplica a otro objeto y es derivada a través del enlace.

Este esquema se define en una relación miembro de un enlace acorde a una operación de selección ya especificada en su relación propietaria. Es importante aclarar que el enlace entre la relación propietaria y la miembro se define como un acople por igual y este a su vez puede ser implementado con semiacoples. El propósito de la fragmentación horizontal derivada es particionar la relación miembro conforme a los fragmentos de su propietaria, pero los fragmentos resultantes deben ser definidos sólo en los atributos de la relación miembro (Özsu y Valduriez, 1999).

O sea, dado un enlace L donde propietaria (L) = S y miembro (L) = R, la fragmentación horizontal derivada de R se define como: $R_i = R \ltimes S_i$, $(1 \leq i \leq w)$ donde w es el máximo número de fragmentos que pueden definirse en R y $S_i = \sigma_{Fi}$ (S) es la fragmentación horizontal primaria de S.

Cuando se ha definido una fragmentación horizontal primaria para una relación S, las relaciones conectadas a S a través de enlaces se convierten en candidatos a fragmentación horizontal derivada y es necesario tener presente las siguientes consideraciones:

1. Correctitud de las particiones las cuales son derivadas de la relación propietaria a la miembro del enlace. Dado una partición en la relación propietaria, es siempre posible definir una partición derivada en una relación miembro. Tal partición derivada puede o no ser de interés para la aplicación.
2. Factibilidad en la realización de particiones que se derivan de la relación miembro a la propietaria del enlace. Dada la partición de una relación miembro, no es siempre posible definir una correspondiente partición derivada en la relación propietaria del enlace. Una partición horizontal puede ser derivada si las instancias de la relación de cualquier fragmento de la relación miembro se asignan a sólo un fragmento de la relación propietaria.

Como consecuencia de estas consideraciones es posible definir una jerarquía de particiones derivadas debido a que es muy común que en una relación R exista más de un enlace, por lo que hay más de una posible fragmentación horizontal derivada de R. La decisión de seleccionar una relación candidata a fragmentación horizontal derivada se basa en dos criterios (Özsu y Valduriez, 1999):

1. La fragmentación usada en más aplicaciones.
2. La fragmentación con mejores características de acople.

El primer criterio es más simple si se considera la frecuencia con la que las aplicaciones acceden a los datos. De ser posible, debe facilitar el acceso a los usuarios potenciales de tal forma que el impacto total en el desempeño del sistema sea mínimo.

Para explicar el segundo criterio es necesario considerar el efecto de este tipo de fragmentación y es que el acople entre las relaciones propietarias y miembros para

responder las consultas, es asistido primero ejecutándolo en pequeñas relaciones (ejemplo fragmentos) y en segundo lugar, desarrollando enlaces potenciales de forma distribuida.
El primer criterio es muy claro debido a que al ser los fragmentos de una relación más pequeños que la relación en sí, se agiliza el acople entre cualesquiera de los fragmentos. El segundo criterio es más importante y es el fundamento de las bases de datos distribuidas debido a que además de ejecutar solicitudes en diferentes sitios, es posible ejecutar una solicitud en paralelo lo que trae como consecuencia que se minimice el tiempo de respuesta y se mejore el desempeño del sistema.

1.3.3 Diseño de la fragmentación vertical

La fragmentación vertical no es más que la división de una relación en fragmentos, cada uno de los cuales contiene un subconjunto de los atributos de la relación original. Su principal objetivo es mejorar el rendimiento de las transacciones; para esto los fragmentos deben ser diseñados de tal manera que las aplicaciones accedan al menor número de atributos irrelevantes posibles. Se entiende por atributos irrelevantes aquellos que no son necesarios para resolver la consulta.
En cualquier base de datos las transacciones usualmente no requieren que todos los atributos de las relaciones que son usadas sean extraídos durante su ejecución. Cuando una relación es accedida, los atributos almacenados que son irrelevantes incrementan los costos de almacenamiento y procesamiento, especialmente cuando el número de tuplas involucradas es muy grande. En el caso de una base de datos centralizada esto conduce a un mayor número de accesos a páginas de memoria. En una BDD, cuando los atributos relevantes están en diferentes fragmentos de datos y localizados en diferentes sitios se genera un costo adicional debido al acceso de datos remotos. Por esto, una de las características deseables de una BDD que se debe favorecer con la fragmentación vertical es la accesibilidad local en cualquier sitio.
Si cada aplicación accediera a un subconjunto diferente y disjunto de atributos, el diseño de la fragmentación vertical sería obvio; sin embargo, en la práctica, lo más común es que las aplicaciones accedan a conjuntos de atributos diferentes y solapados. Un aspecto a decidir es si los fragmentos van a ser solapados o disjuntos. En el primer caso, los fragmentos se

adaptan mejor a los requerimientos de las aplicaciones al aumentar las posibilidades de encontrar en un solo fragmento los datos requeridos, mejor aún si se asignan al mismo sitio donde se ejecutará la aplicación, algo que es particularmente beneficioso si la misma es de lectura, pero aumenta considerablemente la complejidad del Sistema Gestor de Bases de Datos Distribuidas (SGBDD) que tendría que encargarse de mantener la consistencia de todos los datos. Esto puede reducir el rendimiento del sistema si no se realiza adecuadamente. En el segundo caso no hay redundancia, pero una determinada fragmentación puede afectar aquellas aplicaciones que al ejecutarse usen atributos de más de un fragmento, siendo necesaria la realización de un acople, operación que es bastante costosa, en particular cuando los fragmentos se encuentran en distintos sitios de la red.

Por tanto, no basta con maximizar el número de aplicaciones que acceden a un solo fragmento y minimizar las que acceden a más de uno, sino que es necesario evaluar cómo influye una determinada fragmentación en el rendimiento total del sistema. Para esto se debería tener en cuenta toda una serie de aspectos como son: la frecuencia de activación de las transacciones, los métodos de acceso empleados, las estrategias de procesamiento de transacciones, costos de transmisión y posibilidad de replicación, entre otros.

Una solución teórica simple al problema de la fragmentación vertical sería escoger un criterio o función objetivo, evaluarlo para todas las posibles particiones y seleccionar aquella que optimiza el criterio. Esta solución se tropieza con dos grandes dificultades. Primero, la obtención de una función objetivo que convierta todos los aspectos enumerados anteriormente en un modelo matemático razonable. Segundo, el número posible de particiones es muy grande incluso para un número moderado de atributos, de manera que incluso evaluar el criterio más simple para todas las particiones no resulta factible. El número total de particiones en una relación con n atributos está dado por el n-ésimo número de Bell B(n) (Niamir, 1978), donde n es el número de atributos de la relación, y está definido de la manera siguiente:

$$B_{n+1} = \sum_{i=0}^{n} \binom{n}{i} B_i$$

Para valores grandes de n, $B(n) \approx n^n$. Esto conduce a la idea, bastante aceptada, de que es inútil buscar soluciones óptimas al problema de la fragmentación vertical y es necesario

recurrir a heurísticas que disminuyan el espacio de solución. Cuando éstas son usadas, se obtiene una solución cercana a la óptima para la partición de los atributos usando un proceso de minimización paso a paso. En este caso se comienza con una partición dada (por ejemplo: la partición trivial) y se intenta derivar de ella una nueva partición que sea superior a la original por el hecho de que la base de datos fragmentada de acuerdo a la nueva partición tendrá un menor costo de procesamiento. Cuando esto es logrado, la heurística trata de obtener mejoras a partir de la nueva partición derivada. Cada vez que tiene éxito mejorar una partición, mejora el rendimiento de la base de datos. Este proceso continúa hasta que no se puedan hacer mejoras a la última fragmentación obtenida; por tanto, ésta será el resultado de la heurística de fragmentación de atributos. Desde este punto de vista existen dos alternativas, llamadas: agrupamiento y particionamiento. La primera parte de ver cada atributo como un fragmento y a partir de ahí ir uniendo los fragmentos entre sí para obtener las particiones candidatas. La segunda parte de un solo fragmento que abarca toda la relación y va dividiéndose para ir obteniendo los nuevos fragmentos candidatos. Esta última se considera la más apropiada pues la solución óptima debe estar más cerca de la tabla original que de fragmentos compuestos por cada atributo independiente.

El concepto de usar la fragmentación vertical con el objetivo de mejorar el rendimiento de los SGBD ha aparecido con frecuencia en la literatura, a continuación se mencionan algunos de los trabajos que más sobresalen.

Hoffer y Severance (Hoffer y Severance, 1975) definieron un algoritmo que agrupa los atributos de una relación basándose en la afinidad entre ellos. Los atributos accedidos juntos por las aplicaciones tienen mayor afinidad y el algoritmo BEA desarrollado por McCormick (McCormick et al., 1972) es usado para agruparlos en clusters.

Navathe, et al (Navathe et al., 1984) extienden el trabajo de Hoffer y definen algoritmos para el agrupamiento de atributos en fragmentos solapados y no solapados; aplicando la fragmentación vertical a tres tipos de bases de datos: distribuidas, con un nivel de memoria y con jerarquía de memoria; con el objetivo de minimizar el número de fragmentos que usa una transacción y refinar los fragmentos usando factores de costo que reflejan el ambiente físico donde se almacena la base de datos. Cornell y Yu (Cornell y Yu, 1987) optimizaron este trabajo desarrollando un algoritmo que obtiene una partición binaria óptima para bases

de datos relacionales. Ellos usan información de factores físicos para disminuir el número de accesos a disco. Posteriores refinamientos son logrados aplicando el algoritmo de partición binaria iterativamente.

Navathe y Ra (Navathe y Ra, 1989) desarrollaron un algoritmo que usa una técnica gráfica donde se construye una matriz de afinidad de atributos (AA), que es representada por un grafo a partir del cual se genera un árbol linealmente conectado y todos los ciclos del árbol forman fragmentos de relaciones. Öszu y Valduriez (Özsu y Valduriez, 1999) discuten este trabajo previo sobre fragmentación vertical en bases de datos distribuidas usando información de las frecuencias de acceso y aplican el algoritmo BEA que agrupa los atributos de una relación basándose en los valores de la afinidad entre ellos. Los grupos de atributos son clusterizados y se usan ecuaciones de costo para definir la mejor posición a lo largo de la diagonal de la matriz clusterizada para dividir la relación en fragmentos.

Muthuraj (Muthuraj et al., 1993) argumenta que los primeros algoritmos para la fragmentación vertical son *ad hoc*, por eso él propone una función objetivo llamada Evaluador de Particiones (EP) para determinar la "calidad" de las particiones generadas por varios algoritmos. Dicho evaluador consta de dos términos: los costos por accesos a atributos locales irrelevantes y los costos por acceder a atributos remotos irrelevantes. El primero mide los costos del procesamiento local de las transacciones debido a accesos a atributos irrelevantes, y el segundo mide los costos del procesamiento remoto debido a las transacciones remotas que acceden a atributos relevantes.

1.3.4 Diseño de la fragmentación mixta

La fragmentación mixta es el proceso de clusterizar tuplas y columnas relevantes accedidas por una aplicación en una unidad llamada fragmento. Este proceso reduce la cantidad de accesos irrelevantes de una aplicación. La fragmentación mixta o también denominada fragmentación híbrida o anidada puede ser construida alternando, en cada relación, fragmentaciones horizontales y verticales recursivamente hasta obtener un fragmento apropiado.

La fragmentación mixta puede ser convenientemente representada por un árbol de fragmentación, en el cual la raíz corresponda a la relación global, las ramas correspondan a

los fragmentos y los nodos intermedios correspondan a resultados intermedios de las expresiones de fragmentación definidas (véase la figura 1.3). Los nodos hijos de un nodo determinado representan la descomposición de este nodo por una operación de fragmentación (Ceri y Pelagatti, 1984), (Özsu y Valduriez, 1999).

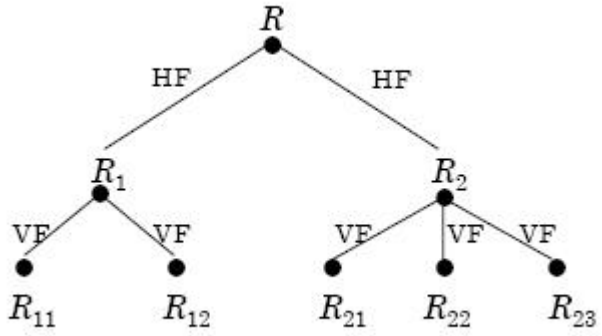

Figura 1.3 *Árbol de Fragmentación Mixta*

El número de niveles de anidamiento puede ser grande pero siempre finito. En el caso de la fragmentación horizontal, el punto de parada es cuando cada fragmento esté compuesto por una tupla, mientras que para la fragmentación vertical la condición para terminar es cuando los fragmentos estén constituidos por atributos. Sin embargo las aplicaciones con más de dos niveles de fragmentación no son de interés práctico debido a que las relaciones globales normalizadas generalmente son de pequeño grado y no se deben ejecutar demasiadas fragmentaciones verticales porque el costo de los acoples sería demasiado alto.

Existen varias propuestas de soluciones para la fragmentación mixta. Pernul (Pernul et al., 1991) presenta una metodología para fragmentar las relaciones donde uno o varios fragmentos representan una vista. Su metodología consiste en aplicar particionamientos verticales y horizontales a las vistas de la relación. Por su parte Navathe (Navathe et al., 1990) propone un método gráfico para la fragmentación horizontal, y para generar la fragmentación mixta sugieren combinar los algoritmos de fragmentación horizontal y vertical. Karlapalem (Karlapalem et al., 1996) también combina los algoritmos de los esquemas de fragmentación vertical y horizontal para lograr un esquema de fragmentación

mixta optimizado. Estas últimas dos metodologías no imponen un orden como es el caso del método de Pernul.

Otras investigaciones como es el caso de Rothnie (Rothnie et al., 1980) y Meghini y Thanos (Meghini y Thanos, 1991) concluyen que la fragmentación mixta debe realizarse en dos pasos separados. Primero, se selecciona el conjunto de tuplas de la relación global acorde a un predicado dado y seguidamente, la relación resultante es proyectada sobre un conjunto de atributos.

Para diseñar fragmentación mixta es más conveniente comenzar con una fragmentación horizontal lo que se justifica considerando los beneficios que guían la naturaleza de la fragmentación horizontal y dentro de los cuales es necesario resaltar la localidad de referencia (la mayoría de las solicitudes y transacciones se originan y relacionan actividades locales) con lo que se garantiza menor tiempo de respuesta y menor costo de procesamiento para solicitudes locales; otra ventaja adicional es que el esquema de particionamiento es, en muchos casos, consistente con la estructura de la organización. Teniendo en cuenta además, que la fórmula Π_{Az} (σ_{Fj} (R_i)) es más usada en los modelos de optimización de consulta porque al aplicar primeramente el operador de selección se disminuyen más cardinalidades y por tanto el tiempo de respuesta de las solicitudes es menor, y que la fórmula σ_{Fj} (Π_{Az} (R_i)) implica que en el momento de realizar la proyección hay que garantizar que no se eliminen los atributos necesarios para la selección, lo que le añade complejidad al diseño de fragmentación mixta ya que habría que comprobar en cada paso, que el atributo que se vaya a eliminar no esté presente en el predicado de selección y esto incrementaría los tiempos de respuesta de las solicitudes. No obstante el diseñador de la base de datos tiene la posibilidad de decidir cuál de los dos tipos de fragmentación aplicar primero, en base al conocimiento que tenga de las aplicaciones.

1.3.5 Conclusiones parciales

En este capítulo se ha hecho una introducción a los conocimientos básicos del diseño de las BDD y se han mencionado las principales investigaciones de diferentes autores en el tema de la fragmentación. Después de hacer un estudio de las metodologías referenciadas se seleccionaron las que proponen Özsu (Özsu and Patrick, 1999) y Navathe (Navathe et al.,

1984) porque se consideran relativamente fáciles de implementar. En el Capítulo 2, se hace una descripción mas detallada de dichas metodologías.

Referencias bibliográficas

CERI, S. y PELAGATTI, G. (1984) *Distributed Databases: Principles and Systems*, McGraw-Hill New York, NY, USA.

CERI, S., PERNICI, B. y WIEDERHOLD, G. (1989) Optimization problems and solution methods in the design of data distribution. *Inf. Syst.*

CORNELL, D. W. y YU, P. S. (1987) A Vertical Partitioning Algorithm for Relational Databases. *In Proceedings of the Third International Conference on Data Engineering.* IEEE.

EZEIFE, C. I. y BARKER, K. (1995) A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System. *Distributed and Parallel Databases.*

HOFFER, J. A. y SEVERANCE, D. G. (1975) The use of cluster analysis in physical database design. *Proceedings of the 1st International Conference on Very Large Databases.* ACM.

KARLAPALEM, K., LI, Q. y VIEWEG, S. (1996) Method Induced Partitioning Schemes for Object-Oriented Databases.**,** 377 - 384.

MCCORMICK, W. T., SCHWEITZER, P. J. y WHITE, T. W. (1972) *Problem decomposition and data reorganization by a clustering technique*, Operations Research.

MEGHINI, C. y THANOS, C. (1991) The Complexity of Operations on a Fragmented Relation. *ACM Trans. Database Syst.,* 16**,** 56-87.

MUTHURAJ, J., CHAKRAVARTHY, S., VARADARAJAN, R. y NAVATHE, S. (1993) A Formal Approach to the Vertical Partitioning Problem in Distributed Database Design. *2nd International Conference on Parallel and Distributed Information Systems.* San Diego, IEEE Computer Society.

NAVATHE, S., CERI, S., WIEDERHOLD, G. y DOU, J. (1984) Vertical partitioning algorithms for database design. *ACM Transactions on Database Systems.*

NAVATHE, S., KARLAPALEM, K. y MINYOUNG, R. (1995) A mixed fragmentation methodology for initial distributed database design. *Journal of Computer and Software Engineering.*

NAVATHE, S. y RA, M. (1989) Vertical partitioning for database design: A graphical algorithm. *International Conference on Management of Data.* ACM.

NAVATHE, S., RIVERA-VEGA, P. I. y VARADARAJAN, R. (1990) Scheduling Data Redistribution in Distributed Databases. *Proceedings of the Sixth International Conference on Data Engineering.* Los Angeles, California, USA, IEEE Computer Society.

NIAMIR, B. (1978) *Attribute partitioning in a self-adaptive relational data base system*, Massachusetts Institute of Technology.

ÖZSU, M. T. y VALDURIEZ, P. (1999) *Principles of Distributed Database Systems, Second Edition*, Prentice-Hall.

PERNUL, G., KARLAPALEM, K. y NAVATHE, S. (1991) Relational Database Organization Based on Views and Fragments.

PRAMANIK, S., KAO, D. T. y VINEYARD, D. (1992) *Fragmentation of Recursive Relations in Distributed Datbases*, Springer.

ROTHNIE JR., J. B., BERNSTEIN, P. A., FOX, S., GOODMAN, N., HAMMER, M., LANDERS, T. A., REEVE, C., SHIPMAN, D. W. y WONG, E. (1980) Introduction to a System for Distributed Databases. *ACM Trans. Database Syst.,* 5**,** 1-17.

ZHOU, X., ZHANG, Y. y ORLOWSKA, M. E. (1994) A new Fragmentation Scheme for Recursive Query Processing. *Data & Knowledge Engineering.*

Capítulo 2. SOLUCIÓN DE LA FRAGMENTACIÓN

En la creación del asistente que guía el proceso de diseño de la fragmentación, fueron implementados algoritmos usando las metodologías de Özsu (Özsu and Patrick, 1999) y Navathe (Navathe et al., 1984) para la fragmentación horizontal y vertical respectivamente. En este capítulo se describen dichas metodologías y algoritmos.

2.1 Fragmentación horizontal

Para realizar la fragmentación horizontal como se vio en el Capítulo 1 es necesario tener información acerca de la base de datos y de las aplicaciones que las utilizan. Sobre la base de datos la información necesaria es la referida al esquema conceptual global; en este sentido es importante tener las relaciones que lo componen, la cardinalidad de cada relación y las dependencias entre ellas. De aquí se tiene a la hora de fragmentar horizontalmente sobre qué esquemas se realiza la fragmentación primaria o la derivada. Sobre las aplicaciones, la información útil son los predicados simples definidos por el usuario.

2.1.1 Algoritmo para la fragmentación horizontal primaria

Un fragmento se define como el conjunto de todas las tuplas que satisfacen un predicado minterm.

$R_j = \sigma_{Fj}$ (R), $1 \leq j \leq w$ (w representa la cantidad de predicados *minterm*).

Para que los fragmentos *minterm* reflejen correctamente la fragmentación, el conjunto de predicados simples que le sirve de base, debe cumplir determinadas propiedades, como ser completo y minimal.

Sea $P = \{ p_1,...,p_n \}$ un conjunto de predicados simple:

1. P es un conjunto de predicados completo si y sólo si cualesquiera dos uplos pertenecientes al mismo fragmento *minterm* son referenciados con igual probabilidad por cualquier aplicación.

2. P es minimal si todos sus predicados son relevantes para definir una fragmentación. Un predicado simple p_i es relevante respecto a P si existen al menos dos predicados *minterm* de P cuyas expresiones difieran sólo en el predicado p_i el que aparece en su forma simple en un caso y negada en el otro.

Con esto tenemos que el primer paso para la fragmentación horizontal primaria consiste en obtener un conjunto de predicados simples mínimos y completos *Pr'*, a partir del conjunto de predicados simples *Pr,* definido por el diseñador a través del caracterizador de aplicaciones.

El siguiente método permite construir P completo y minimal.

***Regla 1*:** Regla fundamental de completitud y minimalidad; dividir una relación o fragmento en al menos dos partes que sean accedidas de forma diferente por al menos una aplicación acorde a un predicado p1. Sea P= p1

***Método*:** Considerar un nuevo predicado p_i que particione al menos a un fragmento en dos partes que sean referenciadas en una forma diferente por al menor una aplicación. Sea P= P U pi, Eliminar predicados no relevantes de P. Repetir este método hasta que P sea completo.

Para implementar éste método se puede seguir el algoritmo iterativo denominado COM_MIN, donde se usa la siguiente notación.
F_i de P_M: Fragmento f_i respecto a un predicado minterm definido sobre lo predicados de P_M.

Algoritmo *COM_MIN*

entrada:

R: relación

Pr: conjunto de predicados simples

salida:

```
        Pr': conjunto de predicados simples
declarar
        F: conjunto de predicados minterm
inicio
   encontrar un pi ∈ Pr tal que pi parta a R de acuerdo con la Regla 1;
   Pr'←pi
   Pr←Pr – pi
   F←fi            // fi es el fragmento minterm  respecto a pi
   hacer
            inicio
                    encontrar un pj ∈ Pr tal que pj  parta a un fk de Pr' de acuerdo
                    con la Regla 1;
                    Pr'←Pr'∪pj
                    Pr←Pr – pj
                    F←F∪fj
                    si ∃pk∈Pr' irrelevante entonces
                    inicio
                            Pr'←Pr' – pk
                            F←F – fk
                    fin-si
            fin-inicio
   hasta que Pr' esté completo
fin. {COM_MIN}
```

El segundo paso es derivar el conjunto de predicados *minterm* que pueden definirse sobre los predicados del conjunto *Pr'*. Estos predicados *minterm* establecen los fragmentos candidatos para el proceso de asignación.
El establecimiento de los predicados *minterm* es trivial; la dificultad radica en el tamaño del conjunto, que puede ser muy grande (de hecho, exponencial respecto al número de predicados simples).

El tercer paso aborda la eliminación de algunos fragmentos *minterm* que puedan ser redundantes. Esta eliminación se desarrolla identificando aquellos *minterm* que puedan resultar contradictorios sobre un conjunto de implicaciones *I*. Por ejemplo, si $Pr' = \{p_1, p_2\}$, donde

p1: atributo = valor_1

p2: atributo = valor_2

y el dominio de *atributo* es {*valor_1, valor_2*}, es obvio que *I* contiene las dos implicaciones siguientes

i1: (atributo = valor_1) $\Rightarrow$ $\neg$(atributo = valor_2)

i2: $\neg$(atributo = valor_1) $\Rightarrow$ (atributo = valor_2)

Los siguientes cuatro predicados *minterm* se definen de acuerdo a *Pr':*

m1: (atributo = valor_1) $\wedge$ (atributo = valor_2)

m2: (atributo = valor_1) $\wedge$ $\neg$(atributo = valor_2)

m3: $\neg$(atributo = valor_1) $\wedge$ (atributo = valor_2)

m4: $\neg$(atributo = valor_1) $\wedge$ $\neg$(atributo = valor_2)

En este caso los predicado *minterm* m_1 y m_4 resultan contradictorios respecto a las implicaciones *I* y pueden eliminarse de *M*.

La entrada al algoritmo HORIZONTALP (Özsu and Patrick, 1999) es una relación R_i sujeta a la fragmentación, y Pr_i, el conjunto de predicados simples que se establecieron de acuerdo a las aplicaciones definidas sobre la relación R_i.

Algoritmo *HORIZONTALP*

entrada:

Ri: relación

Pri: conjunto de predicados simples

salida:

Mi: conjunto de fragmentos minterm

inicio

Pr'i←COM_MIN(Ri, Pri) // construir Pr'i completo y minimal

determinar el conjunto Mi de predicados minterm;

determinar el conjunto Ii de implicaciones de pi∈Pr'i;

para cada mi∈Mi **hacer**

si mi es contradictorio respecto a I

entonces

Mi←Mi – mi

fin-si

fin-para

fin. {HORIZONTALP}

2.1.1.1 Conjunto de predicados simples mínimo y completo

La esencia del algoritmo COM_MIN es eliminar los predicados redundantes para producir un conjunto mínimo y luego usar los complementos para hacer el conjunto completo cuando se necesite crear los minterms. Sin embargo, implementar la Regla 1, la cual juzga si un conjunto de predicados es mínimo y completo no es tan sencillo.

Existe un algoritmo diferente para encontrar el conjunto de predicados simples mínimo y completo, con resultado en esencia igual al producido por el algoritmo COM_MIN y sencillo a la hora de implementarlo (Artiles, 2000). Este algoritmo consta de los siguientes pasos:

1. Leer el conjunto de predicados simples que se utilizan sobre la relación a fragmentar y agruparlos por atributo. Las aplicaciones se basan en uno o más atributos, diferentes aplicaciones pueden tener consultas basadas en los mismos atributos y quizás en los mismos valores. Para obtener un conjunto de predicados simples mínimo y completo se necesita primeramente agruparlos. Por ejemplo, {salario > 300, salario = 200, salario ≥ 300, salario = 300, salario < 300}, pueden pertenecer a aplicaciones diferentes.
2. Ordenar los predicados de acuerdo a su valor. Este paso es para lograr mayor eficiencia a la hora de eliminar los redundantes. Si existen varios predicados con igual significado, el valor debe ser el mismo. Por tanto después del proceso de ordenamiento, ellos deben estar adyacentes.
3. Cambiar los predicados a forma estándar y eliminar los redundantes. Un predicado simple es esencialmente el mismo que su forma complementaria, por ejemplo salario ≥ 300 y salario < 300. Luego si uno de los predicados está no se necesita el otro, el cuál no puede dividir más al fragmento. En este algoritmo se escogió una forma estándar, por tanto no aparecen los operadores > y ≥ . Si apareciera alguno de ellos en el conjunto de predicados se debe cambiar a su forma complementaria. En el ejemplo mencionado arriba será cambiado a {salario = 200, salario ≤ 300, salario < 300, salario = 300, salario < 300}. Entonces se pueden eliminar los redundantes, aquellos que estén repetidos (reduciéndolo a una sola aparición). Aplicando esto al ejemplo se obtiene {salario = 200, salario ≤ 300, salario < 300, salario = 300}.
4. Después del paso tres, se puede obtener un conjunto en el cual el mismo valor aparezca a lo sumo tres veces. Bajo esta condición basta con solo tener los operadores < e =. Luego si existen tres valores adyacentes iguales, se deben

cambiar por dos predicados con igual atributo y valor pero con los operadores < e =, eliminando el tercer predicado. En el ejemplo se obtendrá {salario = 200, salario < 300, salario = 300}.

5. Se necesita adicionar los complementos al conjunto. Para aquellos predicados con operador = y que no tenga el mismo valor que su predecesor (se esta asumiendo que siempre el < y ≤ estarán delante de = para un mismo atributo e igual valor) y no todos los predicados tengan como operador el = , insertar el operador < delante. En el ejemplo se obtiene, {salario < 200, salario = 200, salario < 300, salario = 300}.
6. Adicionar el complemento en la posición final. Cuando todos los predicados sean de operadores =, insertar una condición al final que signifique la negación de todos los predicados anteriores a esta posición. en caso contrario insertar en la posición final una condición complemento en dependencia del predicado predecesor. En el ejemplo se obtendría {salario < 200, salario =200, salario < 300, salario = 300, salario > 300}. Al concluir este paso se tiene un conjunto de predicados mínimo y completo.
7. Para todos los conjuntos de predicados agrupados por atributos hacer los pasos del 1 al 6. Al final se obtienen varios conjuntos de predicados basados en el mismo atributo. Si unen todos esos conjuntos se obtiene un conjunto de predicados simples mínimo y completo.

En el caso de aquellos conjuntos de predicados agrupados por atributos que sean desigualdades, por ejemplo: {salario < 200, salario =200, salario < 300, salario = 300, salario > 300}, cuando se vaya ha obtener los minterms se deben asumir como subrangos, en el ejemplo anterior se toman el predicado salario < 200, salario = 200 y el predicado que viene a continuación debe verse como salario > 200 y salario < 300. Luego a la hora de adicionar al *minterm* se adiciona como subrango y no salario < 300 como un predicado solo.

2.1.1.2 Minterms e implicaciones lógicas

Después de obtener el conjunto de predicados, es fácil obtener los *minterms*. Solo hay que seleccionar un predicado de cada conjunto, usando la operación *AND*. En este proceso se puede observar que no se obtendrán *minterms* contradictorios debido a que se selecciona una sola condición de cada atributo por vez. Además se puede saber cuántos *minterms* serán obtenidos, es decir cuántos fragmentos se obtendrán con solo multiplicar la cantidad de elementos de cada conjunto intermedio. La complejidad del algoritmo es de un $O(n^4)$.

Como se garantiza que no existan *minterms* contradictorios, por tanto existen implicaciones lógicas que no son necesarias tratar. Además se garantiza que el número de *minterms* a generar sea menor que el conjunto total de combinaciones posibles, logrando así que se generen sólo aquellos que estén correctos. Es preciso destacar que siempre quedan aquellas implicaciones lógicas que representan contradicciones de carácter semántico. Estas implicaciones son muy específicas del sistema que se esté diseñado y por tanto es muy difícil automatizar su eliminación.

El diseñador debe proporcionar estas implicaciones lógicas de carácter semántico de forma explícita para luego, al aplicarlas sobre el conjunto de *minterms* eliminar aquellos que las cumplan.

No es objetivo del presente trabajo la captación y verificación de estas implicaciones lógicas de carácter semántico. Para resolver este problema momentáneamente, se le da la posibilidad al diseñador de poder revisar los fragmentos horizontales primarios y eliminar aquellos que el considere son contradictorios desde el punto de vista semántico.

2.1.2 Correctitud de la fragmentación horizontal primaria

Completitud. Se basa en la selección de los predicados a usar. En la medida en que los predicados seleccionados sean completos, se garantizará que la fragmentación también lo

sea. Como el algoritmo antes mencionado parte de la base de obtener el conjunto de predicados Pr' completo y mínimo, se garantiza la completitud.

Reconstrucción. La reconstrucción de una relación global a partir de sus fragmentos se desarrolla con el operador de unión. Si la relación R es fragmentada en $F_R = (R_1, R_2,..., R_r)$, entonces, $R = \cup R_i \quad \forall R_I \in F_R$

Disyunción. Se garantiza en la medida en que los predicados *minterm* que determinan la fragmentación son mutuamente exclusivos.

2.1.3 Algoritmo para la fragmentación horizontal derivada

Una fragmentación horizontal derivada (FHD) se define sobre una relación miembro de acuerdo a una operación de selección especificada sobre su propietaria. Se deben dejar claros dos puntos. Primero, el enlace entre las relaciones propietarias y miembros se define como un equijoin. Segundo, un equijoin puede desarrollarse a través de semiacoples. Este segundo punto es especialmente importante, ya que se desea fraccionar una relación miembro según la fragmentación de su propietaria, además es necesario que el fragmento resultante se defina únicamente sobre los atributos de la relación miembro.

El algoritmo de fragmentación derivada consta de tres entradas:

1. El conjunto de particiones de la relación propietaria.
2. La relación miembro.
3. El conjunto de predicados de semiacople entre el propietario y el miembro.

El algoritmo de fragmentación es trivial y no se presenta aquí.

La FHD es usada para facilitar el acople entre fragmentos, un acople distribuido se hace entre relaciones fragmentadas horizontalmente. Cuando una aplicación requiere el acople entre dos relaciones globales R y S, todas las tuplas de R y S necesitan ser comparados, luego en principio es necesario comparar todos los fragmentados R_i de R con todos los fragmentos S_j de S. En ocasiones es posible deducir que algunos acoples parciales $R_i \bowtie S_j$

son intrínsecamente vacíos como es el caso en que los valores de los atributos de acople en R_i y S_j son disjuntos. Un acople distribuido se representa eficientemente usando grafos de acople.

Un grafo de acople distribuido para $R \bowtie S$ es un grafo $G(N, E)$ donde el conjunto de nodos N representa a los fragmentos de R y S; los arcos no dirigidos entre nodos representan acoples entre fragmentos no intrínsecamente nulos, figura 2.1.

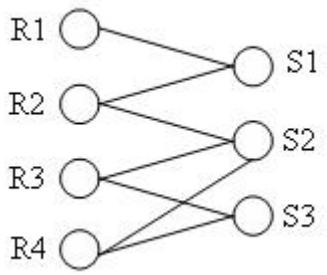

Figura 2.1 *Grafos de acople*

Un grafo de acople es total si contiene todos los posibles arcos entre fragmentos y es reducido si se omiten algunos. Hay dos grafos reducidos de particular interés:

- Grafo particionado: Contiene más de un subgrafo sin arcos entre ellos (véase la figura 2.2)
- Grafo Simple: Es particionado y cada subgrafo tiene sólo un arco (véase la figura 2.3)

Los grafos simples son importantes en el diseño pues los pares de fragmentos pueden ser ubicados en el mismo sitio y el acople se puede ejecutar en forma distribuida acoplando localmente pares de fragmentos y luego coleccionando los resultados de estos acoples parciales.

Es importante diseñar una base de datos distribuida de forma que los acoples más frecuentes tengan un grafo de acople simple. Desgraciadamente, la obtención de grafos de acople simples no siempre es posible. En tal caso, la mejor alternativa sería realizar un

diseño que provoque un grafo de acople particionado. Los fragmentos que se obtengan no se distribuirán para ejecuciones paralelas de un modo tan fácil como aquellos obtenidos a través de grafos simples, pero su asignación aún será posible.

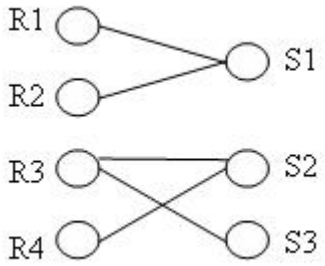

Figura 2.2 *Grafo de acople particionado*

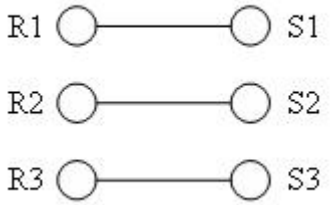

Figura 2.3 *Grafo de acople simple*

2.1.4 Correctitud de la fragmentación horizontal derivada

Completitud. Sea R la relación miembro de un enlace cuya propietaria es la relación S, fragmentada como $F_S = \{S_1, S_2, ..., S_w\}$. Además, sea A el atributo de acople entre R y S. Entonces para cada tupla t de R, existirá una tupla t' de S tal que $t[A] = t'[A]$.
A esta regla se le conoce como integridad referencial y asegura que las tuplas de cualquier fragmento de la relación miembro están también en la relación propietaria.

Reconstrucción. La reconstrucción de una relación global a partir de sus fragmentos se desarrolla con el operador de unión. Si la relación R es fragmentada en $F_R = (R_1, R_2, ..., R_r)$, entonces, $R = \cup R_i \quad \forall R_I \in F_R$

Disyunción. Establecer esta condición para la fragmentación horizontal derivada no es tan simple como lo es en la fragmentación horizontal primaria ya que existe una operación de semiacoples que añade complejidad al asunto. Esta condición puede garantizarse si el grafo de acople es simple. Si no es simple, será necesario investigar los valores de las tuplas.

En general, no se desea unir una tupla de una relación miembro con dos o más tuplas de una relación propietaria cuando estas tuplas se encuentran en fragmentos diferentes a los de la propietaria. Esto no es fácil de establecer, e ilustra por qué los esquemas de la fragmentación derivada que generan un grafo de acople simple son siempre más atractivos.

Por último se deben destacar dos puntos: primeramente la fragmentación derivada puede seguir una cadena donde una relación miembro se fragmenta como resultado del diseño de otra y provoca la fragmentación de otra relación miembro. Segundo, normalmente, existirá más de una forma de fragmentar una relación y es posible que la decisión final deba estar guiada por la asignación.

2.2 Fragmentación Vertical

Para la fragmentación vertical la información requerida, fundamentalmente está relacionada con las aplicaciones y cómo estas acceden a los datos. Sea $Q=\{q_1, q_2,\ldots, q_q\}$ el conjunto de consultas o aplicaciones que se ejecutan sobre la relación $R(A_1, A_2, \ldots, A_n)$. Se define:

$$u_{ki} = \begin{cases} 1 & \textit{si la transacción k usa el atributo i.} \\ 0 & \textit{en caso contrario.} \end{cases}$$

u ki define la Matriz de Uso de atributos (AU) donde la entrada (k, i) denota uki.

freq kj: frecuencia de activación de la transacción k en el sitio j.

n kj: número de tuplas de la relación R que la transacción k trata en el sitio j.

acck: número total de tuplas que la transacción k trata en todos los sitios.

Por tanto:

acck= ∑j n kj freq kj

A partir de la matriz AU se obtiene la matriz AA. La afinidad entre dos atributos se define como:

$$aff_{ij} = \sum_{k \,/\, u_{ki} = 1 \wedge u_{kj} = 1} acc_k$$

Esta es una medida del vínculo que existe entre dos atributos, y se calcula en función de las aplicaciones que los acceden juntos. En la matriz AA la entrada (i, j) se corresponde con aff_{ij}. Esto produce una matriz cuadrada de orden n y simétrica. Los valores de la diagonal son calculados aunque carecen de sentido.

2.2.1 Algoritmo de Hoffer y Severance

Hoffer y Severance (Hoffer and Severance, 1975) miden la afinidad entre cada par de atributos de la forma definida anteriormente, construyen la Matriz de Afinidad de Atributos (AA) y sobre la base de esta agrupan los atributos usando el Algoritmo de Energía de Enlace (BEA) (McCormick et al., 1972). Esto permite reducir el número de fragmentos a evaluar en las fases posteriores del diseño.

Este algoritmo se considera apropiado por las siguientes razones:

1. Está diseñado específicamente para determinar grupos de elementos similares y no una ordenación lineal de estos. Por ejemplo, agrupa los valores con mayor afinidad en un grupo y los de menor afinidad en otro.
2. El agrupamiento final no depende del orden en que los elementos fueron presentados al algoritmo.
3. La complejidad del algoritmo es razonable (O(n) donde n es el número de atributos).
4. AA es simétrica. Por tanto permite la permutación en parejas de filas y columnas lo que reduce la complejidad.

El algoritmo toma como entrada la matriz de afinidad, permuta sus filas y columnas y genera una Matriz de Afinidad Clusterizada (CA). La permutación se realiza maximizando la siguiente medida de afinidad global (AM):

$$AM = \sum_{i=1}^{n}\sum_{j=1}^{n} aff_{i,j}\left[aff_{i,j-1} + aff_{i,j+1} + aff_{i-1,j} + aff_{i+1,j}\right]$$

donde:

$$aff_{0,j} = aff_{i,0} = aff_{n+1,j} = aff_{i,n+1} = 0$$

Este último conjunto de condiciones tiene en cuenta el caso cuando se va a colocar un atributo fuera de los límites de los ya colocados.

La función de maximización considera sólo la vecindad más cercana a una posición, lo que resulta en una agrupación que tiende a unir los atributos de mayor y menor afinidad entre ellos.

La generación de la matriz CA es hecha en tres pasos:

Inicialización: Toma arbitrariamente dos de las columnas de la matriz AA y la sitúa en la matriz CA.

Iteración: Toma cada una de las restantes n-i columnas (donde i es el número de columnas ya situadas en la matriz CA) e intenta colocarlas en las restantes i+1 posiciones de la matriz CA. Se escoge aquella ubicación que haga la mayor contribución a la medida de afinidad global. Se continúa este paso hasta que no queden columnas por localizar.

Ordenamiento de filas: Una vez determinado el orden de las columnas debe cambiarse el orden de las filas para que estas últimas se correspondan con las primeras.

Donde:

$$cont_{ikj} = 2 * bond_{ik} + 2 * bond_{kj} - 2 * bond_{ij}$$

es la contribución neta a la medida de afinidad global de situar el atributo k entre los atributos i y j, siendo:

$$bond_{ij} = \sum_{z=1}^{n} aff_{z,i} aff_{z,j}$$

Después de determinada la matriz CA se deja a juicio del diseñador la selección de los fragmentos, identificando en ella los grupos de atributos con valores similares de afinidad. Sin embargo no siempre esta identificación es sencilla, sobre todo cuando muchas transacciones se ejecutan sobre una misma relación y estas usan diferentes grupos de atributos.

2.2.2 Partición vertical binaria

Como se mencionó anteriormente el algoritmo de Hoffer y Severance deja al criterio subjetivo del diseñador la selección de los grupos de atributos que formarán los fragmentos. Además, la similitud entre un par de atributos puede ser inadecuada si no se tiene en cuenta la similitud entre grupos mayores de atributos. Navathe et al (Navathe et al., 1984) extienden lo anterior dando algoritmos para la determinación automática de los grupos de atributos que forman los fragmentos y teniendo en cuenta en ellos grupos de atributos con propiedades semejantes. La metodología empleada es de particionamiento en lugar de agrupamiento. Esto es debido a que la solución óptima debe estar más cerca del grupo compuesto por todos los atributos tomados como punto de partida, que de los grupos formados por atributos simples.

El diseño de la fragmentación consta de dos fases. En la primera, el particionamiento vertical está guiado por una noción empírica, en la cual se minimiza la necesidad de una transacción de visitar varios fragmentos. Esta fase no requiere factores de costos detallados. Durante la segunda fase el diseño de los fragmentos puede ser refinado con la incorporación de factores de costo, los cuales reflejan el ambiente físico en el que los fragmentos van a estar almacenados.

2.2.3 Algoritmo basado en una función objetivo empírica

En este caso el objetivo de la fragmentación es encontrar conjuntos de atributos que sean accedidos solos o al menos por distintos conjuntos de aplicaciones. El algoritmo parte de la matriz CA para dividir un objeto en dos subconjuntos disjuntos. Se asume que un punto x es fijado a lo largo de la diagonal principal de la matriz CA como se muestra en la figura 2.4

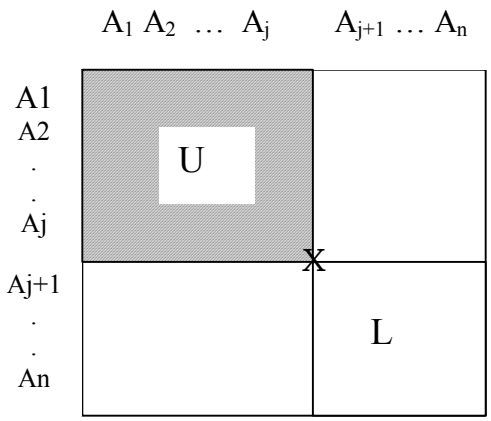

Figura 2.4 *Localización del punto divisor.*

El punto x define dos bloques: U (*upper*) y L (*lower*). Cada bloque define un fragmento vertical formado por los atributos en ese bloque.

Sea A(k) el conjunto de atributos usados por la transacción k, calculados de la siguiente forma:

$$A(k)= \{i \mid u_{ki} = 1\}$$

Y se define:

$$T= (t \mid t \text{ es una transacción})$$

$$LT= (t \mid A_t \subseteq L)$$

$$UT= (t \mid A_t \subseteq U)$$

$$IT= T - (LT \cup UT)$$

Donde T representa el conjunto de todas las aplicaciones; LT y UT representan el conjunto de transacciones que sólo necesitan acceder a L o T, respectivamente; e IT representa el

conjunto de transacciones que tienen que acceder a ambos fragmentos. Sobre la base de los conjuntos anteriores se definen los siguientes costos:

$$CU = \sum_{k \in UT} acc_k$$

$$CI = \sum_{k \in IT} acc_k$$

$$CT = \sum_{k \in T} acc_k$$

$$CL = \sum_{k \in LT} acc_k$$

Donde CT determina el número total de accesos de las transacciones al objeto considerado; CL y CU cuentan el número total de accesos de las transacciones que necesitan un solo fragmento y CI el número total de accesos de las transacciones que necesitan ambos fragmentos. Se consideran un total de n-1 posibles localizaciones del punto x a lo largo de la diagonal principal, donde n es el tamaño de la matriz. Una fragmentación no solapada es obtenida seleccionando un punto x que maximize la siguiente función objetivo z:

$$\max z = CL * CU - CI^2$$

La partición que corresponde al máximo valor de z es aceptada si z es positiva.

La función anterior proviene de un juicio empírico de lo que debe ser una buena fragmentación. La función incrementa CL y CU y disminuye CI. Para un valor dado de CI se selecciona CL y CU de manera tal que CL * CU sea máxima. Como resultado de esto se seleccionan valores de CL y CU que son lo más cercanos posible, por tanto se producen fragmentos que están balanceados con respecto a la carga de las transacciones. Este algoritmo tiene la desventaja de no poder producir particiones de un objeto mediante la selección de un bloque interior en CA. Esta desventaja puede ser evitada si se usa un procedimiento que mueva la columna que está más a la izquierda en la matriz CA hacia el extremo derecho y la fila del extremo superior hacia el extremo inferior, corriendo las demás. Este procedimiento se activa un total de n veces, de forma tal que cada bloque tenga la posibilidad de situarse en la esquina superior izquierda, donde si puede ser reconocido.

Después de cada movimiento se ejecuta el algoritmo de partición binaria determinándose la partición que maximiza z. Esto incrementa la complejidad del algoritmo en un factor de n, resultando ahora $O(n^2)$.

Es posible diseñar una partición n-aria, aunque computacionalmente costoso. En el caso de una fragmentación n-aria no solapada, en general un algoritmo debe probar 1, 2 o … n-1 puntos a lo largo de la diagonal principal como puntos candidatos para definir la partición. La complejidad de este algoritmo sería $O(2^n)$. Una solución para evitar esta complejidad es aplicar recursivamente el algoritmo de partición binaria. En cada iteración los fragmentos producidos por el algoritmo son considerados como dos objetos distintos y la partición binaria es aplicada para fragmentar cada uno de ellos. Esto genera dos subproblemas distintos. En cada subproblema son consideradas sólo aquellas transacciones que usan algún atributo del fragmento considerado. El proceso termina cuando no se recomienda fragmentar porque z no sea positiva.

2.2.4 Correctitud de la fragmentación vertical

Completitud. La completitud de una fragmentación vertical es garantizada por el algoritmo de particionamiento. Ya que cada atributo de la relación global se asigna a uno de los fragmentos. Siempre y cuando el conjunto de atributos *A* sobre los cuales se define una relación *R* consiste en

$$A = U \bigcup L$$

la completitud de la fragmentación vertical se asegura.

Reconstrucción. La reconstrucción de la relación global original se hace por medio de la operación de unión. Así, para una relación *R* con fragmentación vertical *Fr* = {R1, R_2,..., R_r} y llave *K*

$$R = \rhd\lhd_k \; R_i \forall R_i \in F_r$$

Por lo tanto, siempre que R_i sea completo, la operación de unión reconstruirá adecuadamente *R*. Otro punto importante es que o cada R_i debe contener a la llave de *R*, o debe contener los identificadores de tuplas asignados por el sistema.

Fragmentos Disjuntos. Existen dos casos:

1. Los identificadores de tuplas no se considera que se traslapan ya que ellos son mantenidos por el sistema.
2. Las llaves duplicadas no se considera que se traslapan.

2.3 Conclusiones parciales

Este capítulo describe detalladamente en qué consisten las metodologías de Özsu y Navathe para lograr la fragmentación de los esquemas conceptuales. Los algoritmos mencionados serán implementados en la construcción de un módulo que resuelva el problema de la fragmentación en un Sistema de Ayuda al Diseño de Bases de Datos Distribuidas. El siguiente capítulo, explica el diseño e implementación de dicho módulo.

Referencias bibliográficas

ARTILES, M. (2001) Herramienta de ayuda a la fragmentación horizontal. *Facultad de Matemática Física y Computación.* Santa Clara, UCLV.

HOFFER, J. A. & SEVERANCE, D. G. (1975) The use of cluster analysis in physical database design. IN KERR, D. S. (Ed.) *Proceedings of the 1st International Conference on Very Large Databases.* Framingham, Massachusetts, USA, ACM.

MCCORMICK, W. T., SCHWEITZER, P. J. & WHITE, T. W. (1972) *Problem decomposition and data reorganization by a clustering technique*, Operations Research.

NAVATHE, S., CERI, S., WIEDERHOLD, G. & DOU, J. (1984) Vertical partitioning algorithms for database design. *ACM Transactions on Database Systems.*

MORELL, A. (1999) Un enfoque a la fragmentación vertical en bases de datos distribuidas. *Facultad de Matemática Física y Computación.* Santa Clara, UCLV.

ÖZSU, M. T. & PATRICK, V. (1999) Principles of Distributed Database Systems, Second Edition, Prentice-Hall.

Capítulo 3. DISEÑO E IMPLEMENTACIÓN DE LOS ASISTENTES

El presente trabajo se ha concebido como un módulo que posteriormente será insertado dentro de un Sistema de Ayuda al Diseño de Bases de Datos Distribuidas, y no como una herramienta independiente para guiar el proceso de la fragmentación. Por esta razón, la entrada y salida de los asistentes implementados está condicionada al diseño de dicha herramienta, donde se ha definido un catálogo al que todos los módulos que la componen tienen acceso, tanto para obtener sus valores de entrada como para guardar sus resultados, (véase el anexo 1). El objetivo del presente capítulo es mostrar los aspectos generales de diseño e implementación del módulo de fragmentación.

3.1 Casos de Uso del sistema

Este sistema tiene un único actor, el diseñador de bases de datos distribuidas; para quien quedan definidos los casos de usos (figura 3.1), *caracterizar las aplicaciones* y *fragmentar los esquemas globales*.

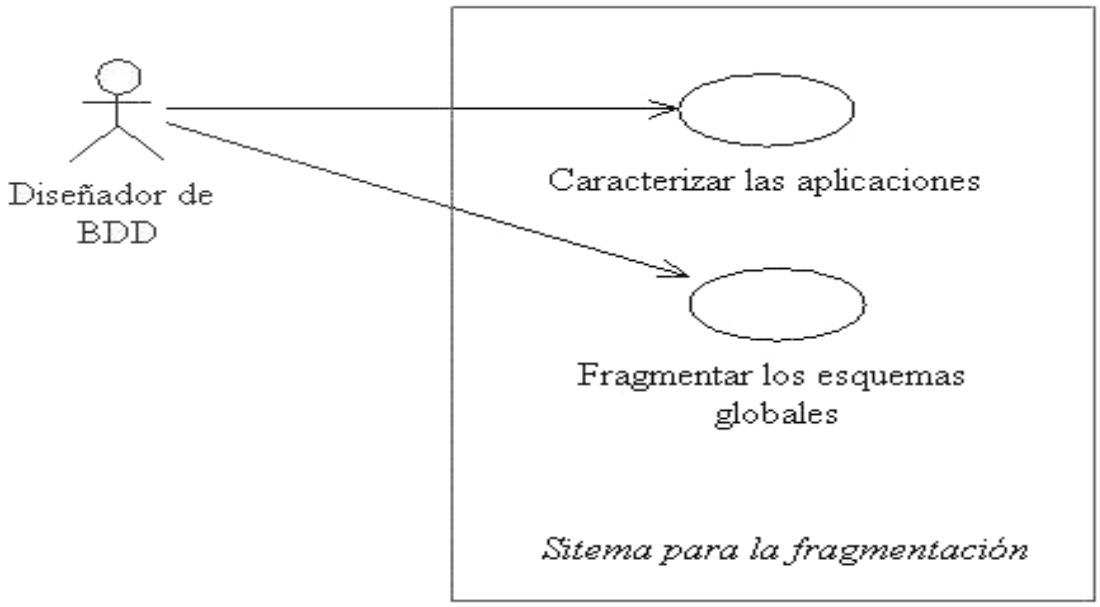

Figura 3.1 *Diagrama de Casos de Uso*

3.2 Asistente para la caracterización de las aplicaciones

Este asistente le brinda la posibilidad al diseñador de introducir en el catálogo de la herramienta general los datos que, bien por su experiencia o estudios realizados, caracterizan a las aplicaciones. Generalmente no es necesario investigar y caracterizar todas las aplicaciones, sino aquellas que se activan con mayor frecuencia.
La información captada de cada aplicación es:

1. Nombre de la aplicación
2. Esquemas que son utilizados
3. El tipo de acceso, si es de lectura y/o escritura
4. Información para la fragmentación vertical
    a. Selectividad de los esquemas, cantidad de tuplas a las que accede la aplicación
    b. Atributos usados de cada esquema seleccionado
    c. Sitios donde se activa la aplicación y la frecuencia con qué lo hace
5. Información para la fragmentación horizontal
    a. Los predicados simples

Los datos de entrada al asistente están almacenados en las tablas:

- Schemas
- GlobalAttributes
- Sites

La salida del asistente se guarda en las tablas:

- Applications
- App_Schema
- App_Attribute
- App_Site
- Predicates

3.2.1 Diagrama de clases

La información que la herramienta permite captar puede dividirse en grupos o categorías, éstas serían: el acceso de la aplicación a los esquemas, a los atributos, la definición de los predicados simples y los sitios donde se activa. Para manejar esta información que difiere de una categoría a otra se implementaron las clases *Schema*, *Attribute*, *Predicate* y *Sites* respectivamente, con el objetivo de independizar el trabajo con el catálogo. La colaboración de las clases entre sí, se realiza a través de *TCharacterizer*. La figura 3.2 muestra la colaboración entre las clases implementadas para la construcción del asistente, y la figura 3.3 una vista de sus atributos y métodos.

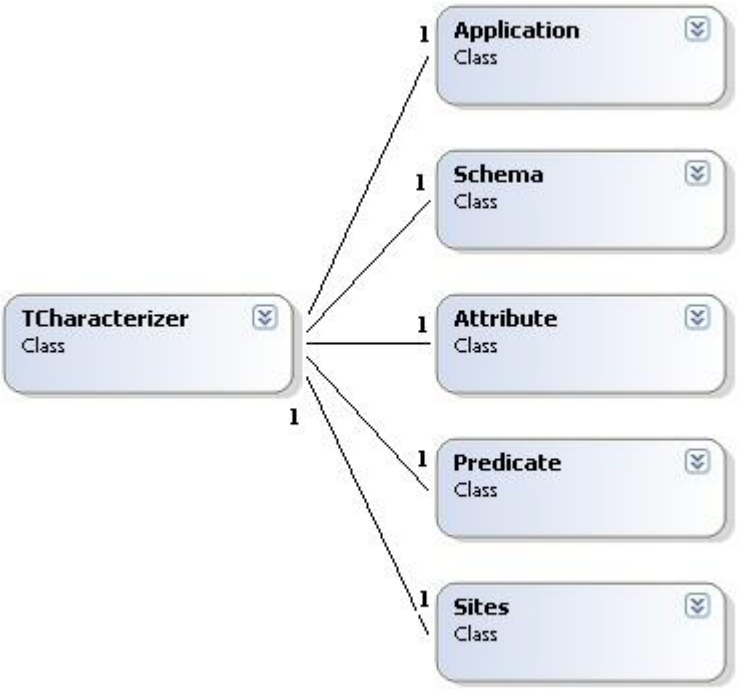

Figura 3.2 *Diagrama de Clases del Caracterizador de Aplicaciones.*

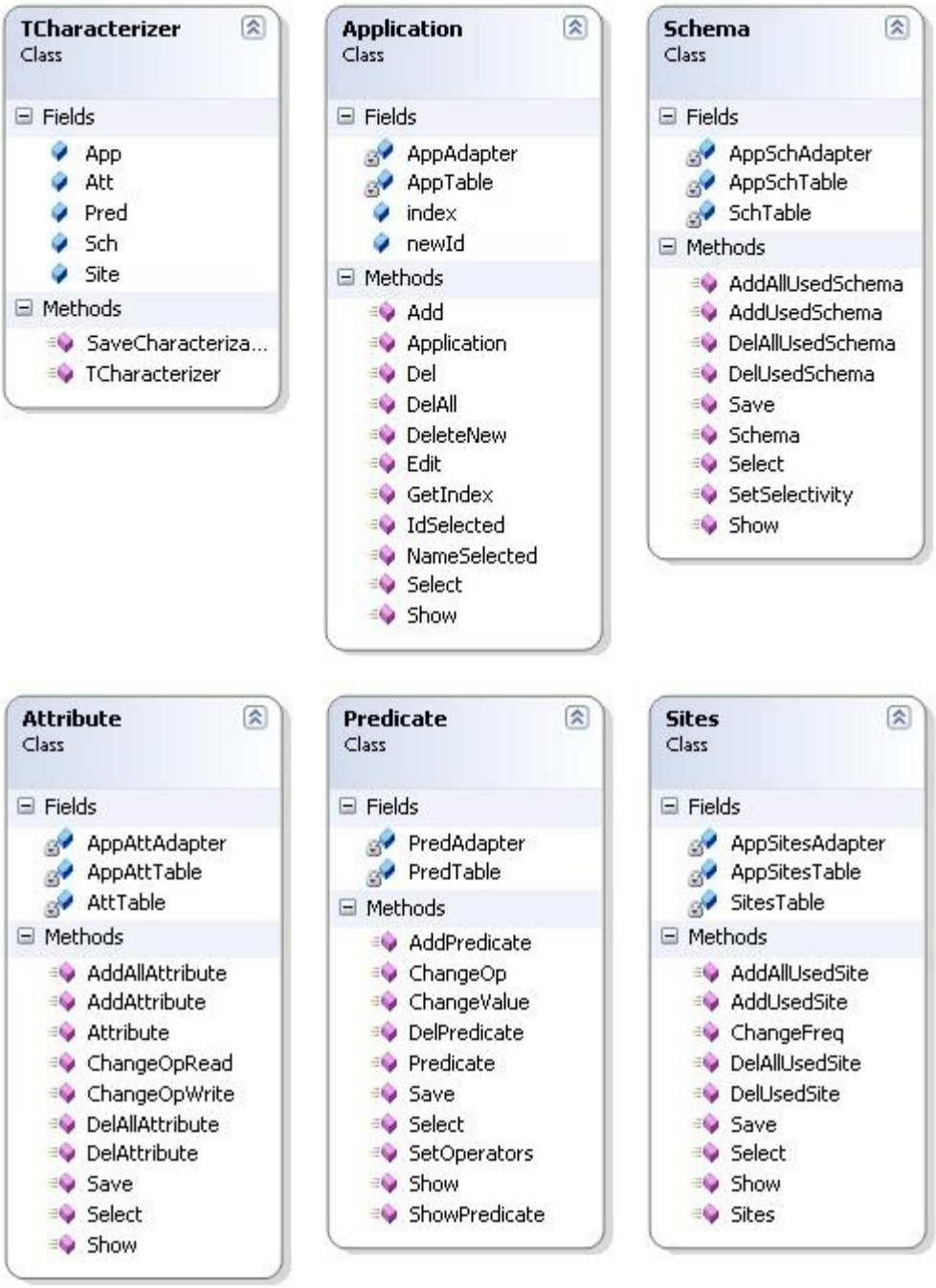

Figura 3.3 Vista detallada de las clases implementadas

3.3 Asistente para la fragmentación

Este asistente guía al diseñador en el proceso de fragmentar los esquemas globales. A pesar de que la literatura recomienda comenzar con la fragmentación horizontal, como se explica en el Capítulo 1; por la experiencia del diseñador y las características de la base de datos que se esté fragmentando, se puede realizar sólo la fragmentación vertical o iniciar el proceso con ella. La herramienta permite que, independientemente de la alternativa que se aplique, después de concluida la etapa de la fragmentación se pueda realizar la otra alternativa obteniendo una fragmentación mixta.

3.3.1 Diagrama de clases

La colaboración entre las clases implementadas se muestra en la figura 3.4 y en la figura 3.5 una vista detalla de los métodos y atributos de las principales clases.

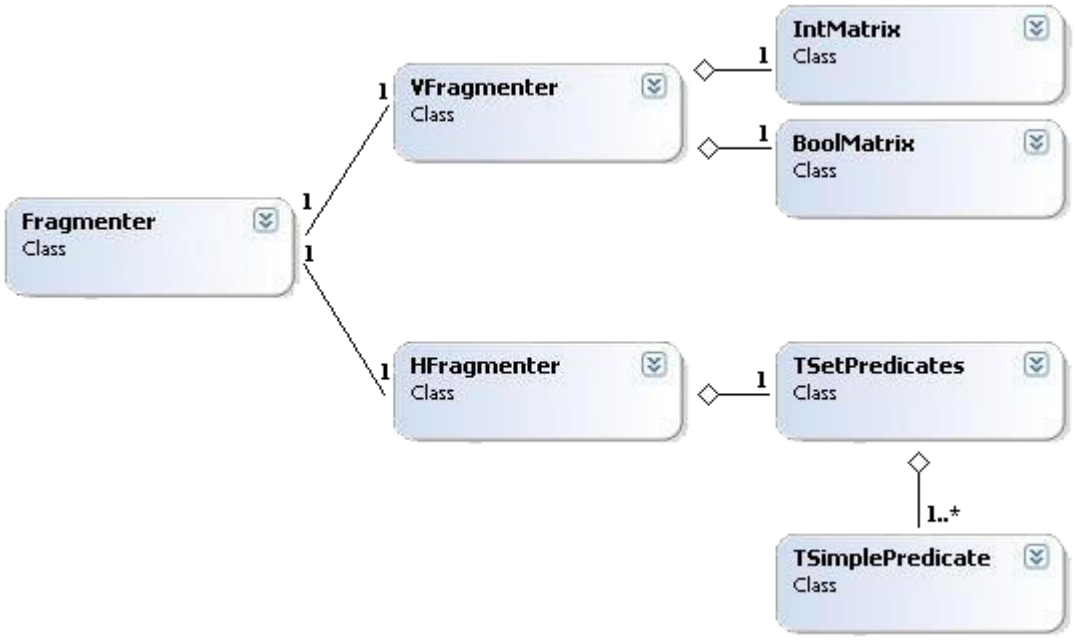

Figura 3.4 Diagrama de Clases del Asistente para la Fragmentación

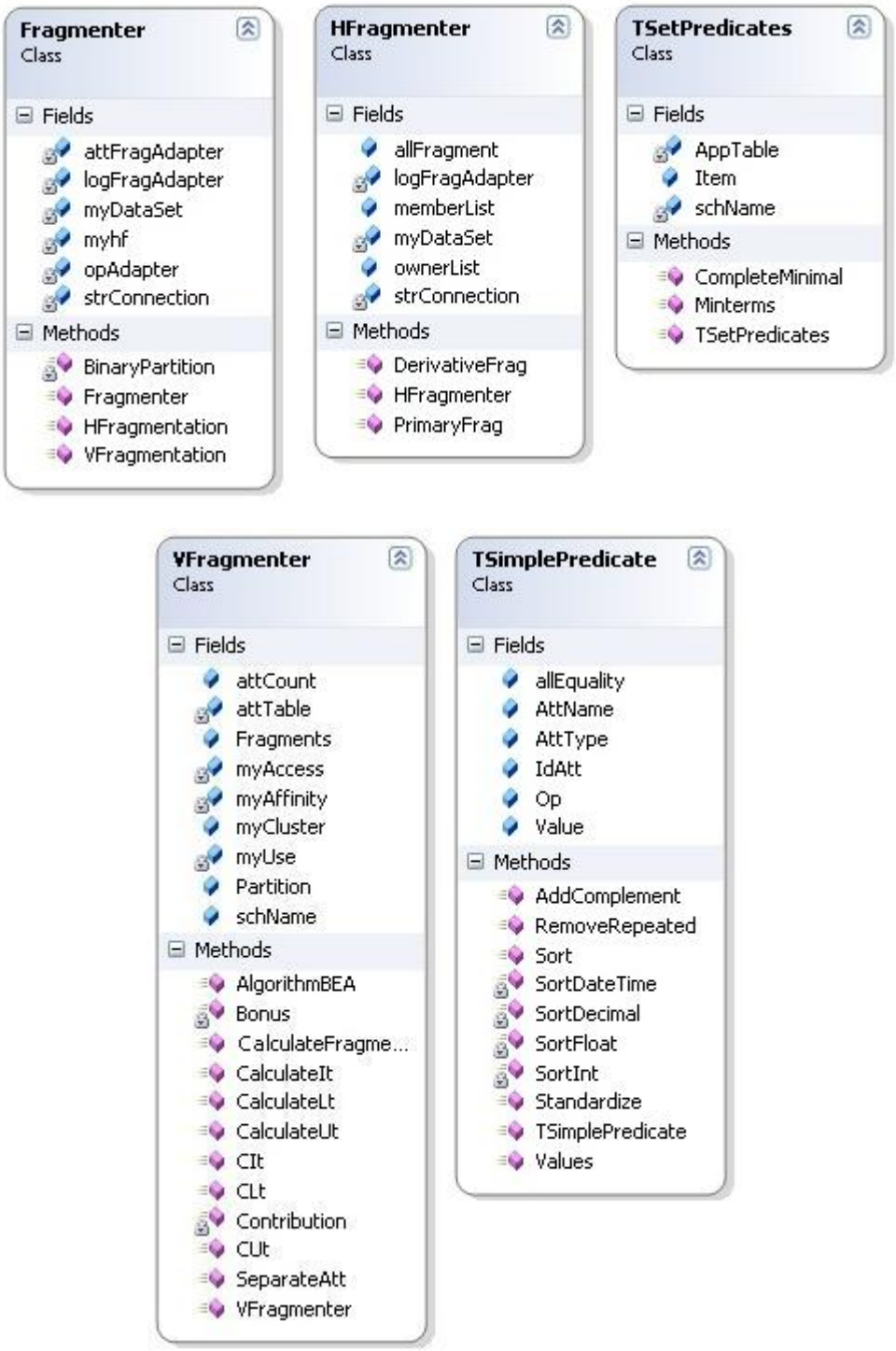

Figura 3.5 *Vista detallada de las principales clases implementadas*

Las principales clases de este módulo son: *TSimplePredicate*, *TSetPredicates*, *HFragmenter*, *VFragmenter* y *Fragmenter*.

TSimplePredicate

Esta clase, dado un atributo determinado, agrupa todos los predicados simples definidos por el usuario en el que está involucrado dicho atributo. Los métodos de la clase son los pasos del algoritmo para la construcción de un conjunto completo y minimal, descrito en el *Epígrafe 2.1.1.1*.

TSetPredicates

El objetivo principal de esta clase es derivar el conjunto de predicados *minterm* que pueden definirse sobre los predicados de la relación a fragmentar. Por cada atributo sobre los que se han definido predicados simples en la relación se crea una instancia de la clase *TSimplePredicate*, donde estas instancias son los elementos del arreglo que representa el atributo *Item* en la clase.

Además, *TSetPredicates* tiene dos métodos uno *CompleteMinimal*, en el que construye el conjunto completo y minimal de predicados y *Minterms*, donde se deriva el conjunto.

CompleteMinimal

```
para cada Item hacer
        Item.Standardize          // estandarizar los predicados simples en Item
        Item.RemoveRepeated       // elimirar los repetidos
        Item.Sort                 // odenar
        Item.AddComplement        // adicionar el complemento
fin-hacer
```

HFragmenter

Se encarga de de decidir qué tipo de fragmentación horizontal primaria y derivada se le aplicará a cada esquemas. El método *PrimaryFrag* implementa la fragmentación primaria y *DerivativeFrag* la derivada. El criterio implementado fue “la fragmentación usada en más

aplicaciones" descrito en el epígrafe 1.3.2.2. Su implementación fue muy directa, se tuvo en consideración la frecuencia con la cual las aplicaciones acceden a los datos.

VFragmenter

El principal método de la clase es *AlgoritmBEA*, donde está implementado el Algoritmo de Energía de Enlace (BEA), que permite reducir el número de fragmentos a evaluar en las fases posteriores del diseño. (véase el epígrafe 2.2.1).

Fragmenter

Es la clase principal o controladora en el proceso de fragmentación de los esquemas. Los métodos *HFragmentation* y *VFragmentation* son los encargados de crear una instancia de los objetos *HFragmenter* y *VFragmenter* respectivamente y aplicar los algoritmos generales de fragmentación.

3.4 Casos de prueba

La herramienta pasó por un período de prueba y depuración, lográndose resultados satisfactorios. A continuación se muestran ejemplos de baja complejidad para dar una idea de la solución que brinda la aplicación; aunque el sistema fue probado con ejemplos de complejidad superior. Los resultados alcanzados con el sistema son coincidentes con las propuestas de diseñadores expertos.

3.4.1 Ejemplo de fragmentación horizontal

Para ejemplificar la fragmentación horizontal nos basaremos en el ejemplo de la figura 3.6. Esta base de datos controla la información de los empleados en un departamento. El salario del trabajador depende del cargo que ocupe. También se tiene la información de los proyectos en que cada empleado trabaja.

Cargos, *Empleados* y *Proyectos* son los esquemas propietarios de las relaciones I, II y III respectivamente. Ahora bien, en la relación I *Empleados* es miembro; por tanto al hacer una

fragmentación horizontal sólo *Cargos* y *Proyectos* son objetos de la fragmentación horizontal primaria.

Figura 3.6 *Diseño de la BD*

Si ya tenemos información sobre el de diseño de la bases de datos, falta saber las características de las aplicaciones que acceden a ella para definir cómo fragmentar.

Vamos a suponer que existe una única aplicación que accede al esquema *Cargos* y chequea la información del atributo *Salario.* Esta aplicación tiene lugar en dos sitios. Se desea manejar los registros de empleados con salarios menores o iguales a $300 en un sitio y aquellos que devengan más de $300 en otro. Para lograrlo se definen los predicados simples:

p1: Cargos.Salario <= 300

p2: Cargos.Salario > 300

Para analizar el acceso al esquema *Proyectos,* se asume que existen dos aplicaciones. La primera es emitida en tres sitios y maneja los nombres y presupuestos de los proyectos. Cada sitio trabaja con los proyectos que pertenecen a una de las universidades involucradas. Para esta aplicación, los predicados simples serían:

p1: Proyectos.Sitio = “UCLV”

p2: Proyectos.Sitio = “UCI”

p3: Proyectos.Sitio = “UH”

La segunda aplicación es emitida en dos sitios y tiene que ver con la administración de los proyectos. Selecciona aquellos que tienen un presupuesto menor o igual a $20 000 para ser administrados en una estación de trabajo y los de presupuesto mayor para una segunda estación. Así los predicados simples que deben ser utilizados para fragmentar son:

p4: Proyectos.Presupuesto <= 20 000

p5: Proyectos.Presupuesto > 20 000

Con esta información de las aplicaciones el resultado de la fragmentación horizontal primaria en las relaciones *Cargos* y *Proyectos*, quedaría:

Cargos1: $\sigma Salario \leq 300(Cargos)$

Cargos2: $\sigma Salario > 300(Cargos)$

Proyectos1: $\sigma Sitio = UCLV \wedge Presupuesto \leq 20000(Proyectos)$

Proyectos2: $\sigma Sitio = UCLV \wedge Presupuesto > 20000(Proyectos)$

Proyectos3: $\sigma Sitio = UCI \wedge Presupuesto \leq 20000(Proyectos)$

Proyectos4: $\sigma Sitio = UCI \wedge Presupuesto > 20000(Proyectos)$

Proyectos5: $\sigma Sitio = UH \wedge Presupuesto \leq 20000(Proyectos)$

Proyectos6: $\sigma Sitio = UH \wedge Presupuesto > 20000(Proyectos)$

Teniendo los fragmentos de las relaciones propietarias, se puede definir la fragmentación sobre las relaciones miembros *Empleados* y *Empleado_Proyecto*. Los empleados quedarían agrupados por sus salarios. Al llegar a este punto *Empleado_Proyecto* puede ser dividida de acuerdo a *Empleados* o a *Proyectos*. Como se dijo anteriormente, el criterio implementado fue la fragmentación utilizada por más aplicaciones. En este caso *Empleado_Proyecto* quedaría agrupado según los fragmentos obtenidos de *Proyectos*.

Finalmente, el resultado de la fragmentación derivada es el siguiente:

Empleados1: Empleados $\ltimes$ Cargos1
Empleados2: Empleados $\ltimes$ Cargos2
Empleado_Proyecto1: Empleado_Proyecto $\ltimes$ Proyectos1
Empleado_Proyecto2: Empleado_Proyecto $\ltimes$ Proyectos2
Empleado_Proyecto3: Empleado_Proyecto $\ltimes$ Proyectos3
Empleado_Proyecto4: Empleado_Proyecto $\ltimes$ Proyectos4
Empleado_Proyecto5: Empleado_Proyecto $\ltimes$ Proyectos5
Empleado_Proyecto6: Empleado_Proyecto $\ltimes$ Proyectos6

3.4.2 Ejemplo de fragmentación vertical

Para este ejemplo se usa el mismo diseño descrito en el epígrafe anterior. Ahora se asumen cuatro aplicaciones que acceden a la relación *Proyectos*:

- La primera se emite en tres sitios con una frecuencia de activación de 15, 20 y 10 respectivamente y tiene que ver con un análisis de los *Presupuestos* de los proyectos dados sus *ID_Proyecto*.
- La segunda solo es ejecutada en el primer sitio con frecuencia 5 y se encarga de hacer reportes donde la información que se requiere de los proyecto es su *Nombre* y el respectivo *Presupuesto*.
- La tercera tiene la misma frecuencia de activación en los tres sitios en que se emite e igual a 25, es encargada de hacer reportes y la información requerida para estos son los nombres de los proyectos y el *Sitio* al que pertenecen.
- La cuarta aplicación solo se emite en un sitio con una frecuencia de 3 y hace una comparación entre los presupuestos de los proyectos según el *Sitio* al que pertenecen.

Con esta información la herramienta construye la matriz de uso (ver tabla 3.1), la matriz de frecuencias de activación (ver tabla 3.2) y la matriz de afinidad (ver tabla 3.3).

	ID_Proyecto	***Nombre***	***Presupuesto***	***Sitio***
Aplicación 1	1	0	1	0
Aplicación 2	0	1	1	0
Aplicación 3	0	1	0	1
Aplicación 4	0	0	1	1

Tabla 3.1 *Matriz de uso.*

	Sitio 1	***Sitio 2***	***Sitio 3***
Aplicación 1	15	20	10
Aplicación 2	5	0	0
Aplicación 3	25	25	25
Aplicación 4	3	0	0

Tabla 3.2 *Matriz de frecuencias de activación.*

	ID_Proyecto	***Nombre***	***Presupuesto***	***Sitio***
ID_Proyecto	45	0	45	0
Nombre	0	80	5	75
Presupuesto	45	5	53	3
Sitio	0	75	3	78

Tabla 3.3 *Matriz de afinidad entre atributos.*

Finalmente queda particionada la relación *Proyectos* en (*ID_Proyecto, Presupuesto*) y (*Nombre, Sitio*), pero la llave primaria *ID_Proyecto* debe estar en todos los fragmentos obtenidos. El resultado de la fragmentación vertical para este ejemplo, queda:

Proyectos1: $\prod_{ID_Proyecto,Presupuesto}(Proyectos)$

Proyectos2: $\prod_{ID_Proyecto,Nombre,Sitio}(Proyectos)$

3.5 Vista del usuario

El objetivo de estos asistentes es guiar al diseñador de bases de datos distribuidas. En cada etapa del proceso se brinda información suficiente y orienta al diseñador, incluso para cuando no sea un experto.

En los subsiguientes epígrafes se muestran algunas de las vistas principales de los asistentes, así como una breve descripción de cómo interactuar con ellos.

3.5.1 Asistente para la caracterización de aplicaciones

Inicialmente se le informa al diseñador que no es necesario caracterizar todas las aplicaciones que actúan en la base de datos, sólo el 20% de las más importante, como se ha mencionado con anterioridad.

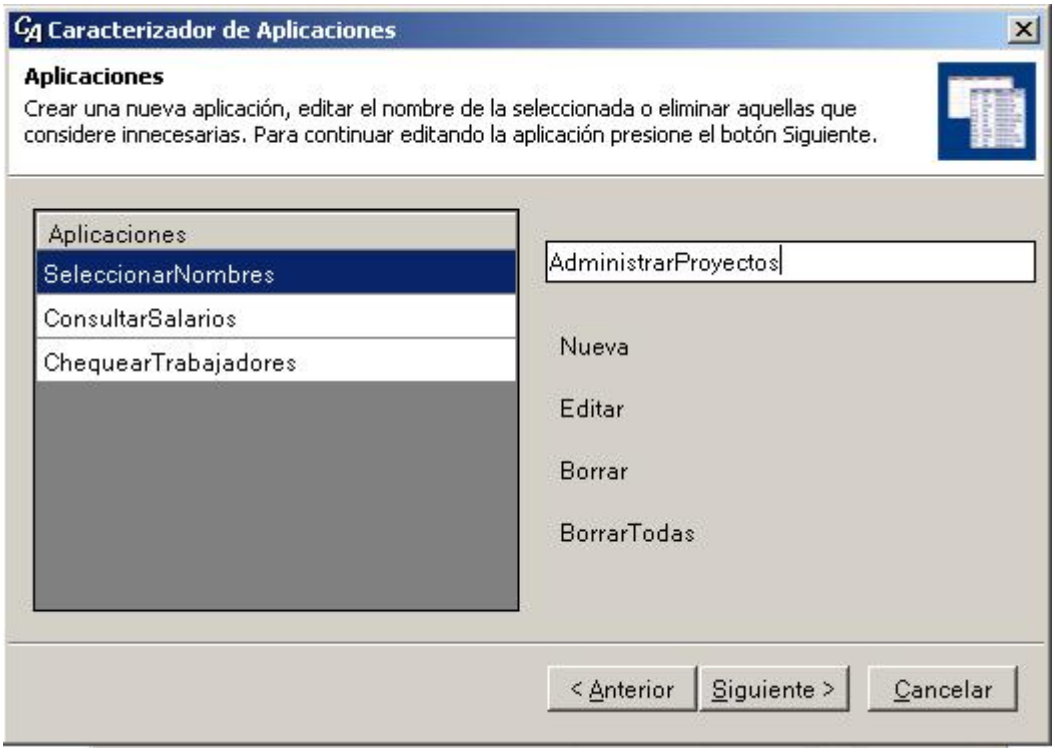

Figura 3.7 *Adicionar, editar o eliminar aplicaciones.*

La figura 3.7 muestra la etapa en que el diseñador puede cambiar el nombre de cualquier aplicación (*Editar*), borrarla (*Borrar*) o borra todas las aplicaciones (*BorrarTodas*); pero

sólo puede adicionar una (*Nueva*) y hasta que no haya concluido el proceso de caracterizarla no tiene la posibilidad de añadir otra. El botón *Siguiente* permite pasar a la siguiente fase de la caracterización de la aplicación seleccionada.

Luego pasa a seleccionar los esquemas usados por la aplicación, especificando la selectividad y de cada uno de ellos, los atributos involucrados. A partir de esa información pueden definirse los predicados simples que guiarán la fragmentación horizontal.

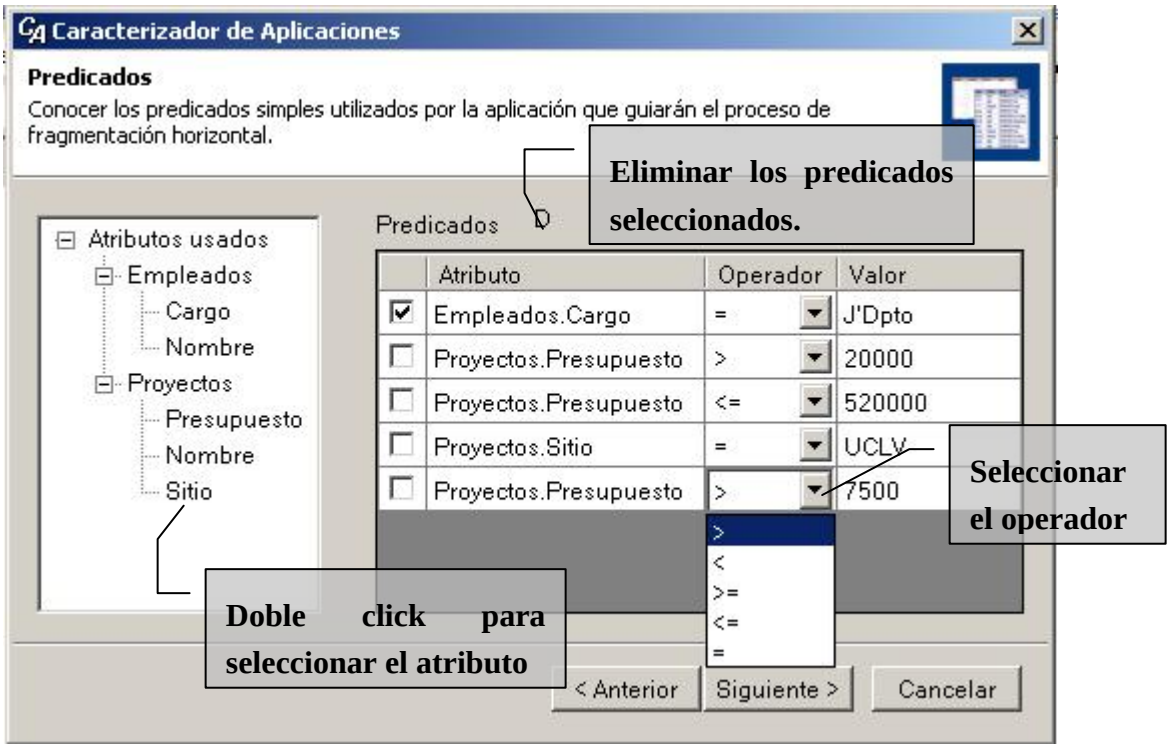

Figura 3.8 *Definición de los predicados simples.*

Al finalizar la caracterización puede cerrar el asistente o salvar los cambios y volver al inicio para caracterizar otra aplicación.

3.5.2 Asistente para la fragmentación de los esquemas

En el asistente el diseñador encuentra cuáles son los aspectos que debe tener en cuenta para hacer un buen diseño de fragmentación y se le brinda la oportunidad de seleccionar la alternativa a aplicar.

De escoger la fragmentación vertical pasaría a seleccionar los esquemas que serán fragmentados y luego visualiza los fragmentos resultantes; para terminar debe pasar a la siguiente etapa donde se salvan los resultados en el catálogo y se le brinda la posibilidad de terminar o volver al inicio para tomar otra decisión.

De seleccionar la fragmentación horizontal, pasa a la etapa de visualizar los *minterms* de cada relación primaria (ver figura 3.9) y debe eliminar aquellos que considere semánticamente contradictorios.

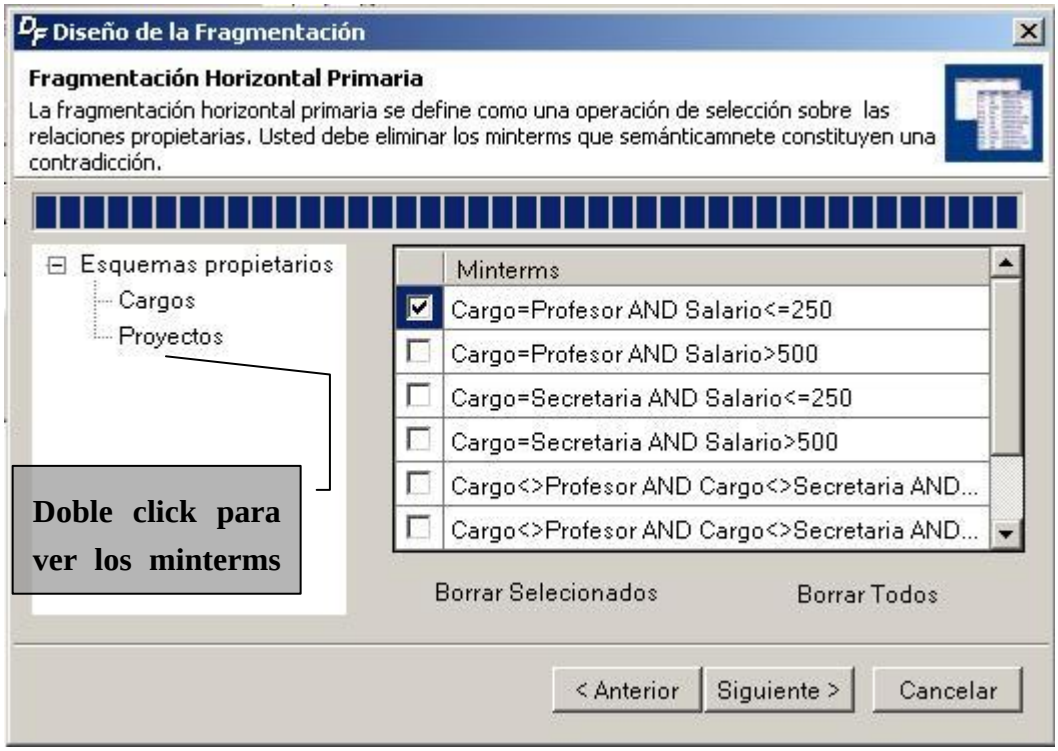

Figura 3.9 *Minterms de las relaciones propietarias*

Al tener los *minterms* sobre los que se definen cada fragmento horizontal, se le muestra al diseñador tanto los fragmentos horizontales primarios, como los derivados (Figura 3.10).

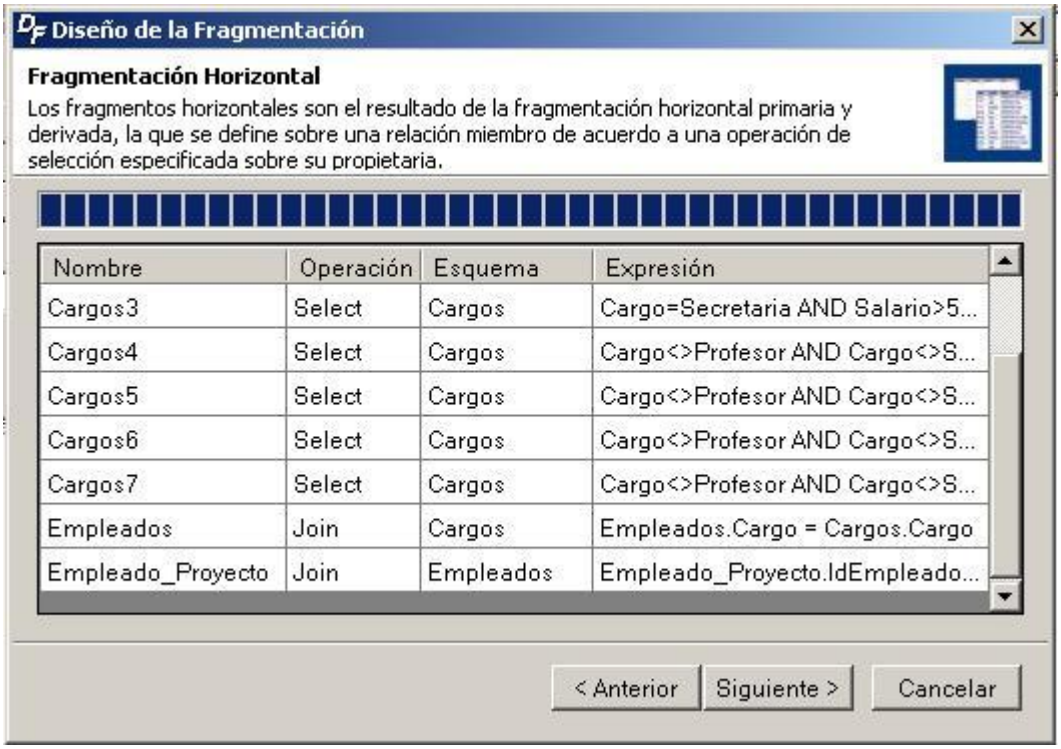

Figura 3.10 *Fragmentos horizontales primarios y derivados*

3.6 Conclusiones parciales

En este capítulo se han descrito los aspectos generales de diseño he implementación de los asistentes que conforman el módulo de fragmentación. La herramienta es auto-explicativa por definición y además cuenta con un fichero de ayuda. No obstante aquí se describe la interacción con los asistentes en las etapas que no resulta muy evidente las acciones a tomar. El usuario es un diseñador de bases de datos distribuidas, familiarizado con herramientas de este tipo.

Conclusiones

En este proyecto se realizó un estudio de las cuestiones teóricas del diseño de las bases de datos distribuidas, enfatizando en la etapa de diseño de la fragmentación. Se analizaron las principales metodologías propuestas por diferentes investigadores en esta esfera. Como resultado, se implementó un asistente para captar la información que caracteriza a las aplicaciones. Además se diseñó e implementó un asistente que guía el proceso de diseño de la fragmentación. Estas dos herramientas podrán ser añadidas como material de ayuda a una aplicación de diseño de bases de datos distribuidas. Finalmente se realizaron varios ejemplos de prueba que permitieron la validación de esta herramienta.

Recomendaciones

Se hacen las siguientes recomendaciones:

- Adicionar un módulo para la captación de las implicaciones lógicas de carácter semántico y su aplicación sobre el conjunto final de *minterms* con el objetivo de eliminar automáticamente los que sean irrelevantes.
- La implementación de otras metodologías o algoritmos para realizar la fragmentación, dándole la posibilidad al diseñador de elegir la que desee utilizar.
- Realizar un estudio comparativo entre las metodologías implementadas con vista a poder brindarle al diseñador una documentación que le permita tomar una decisión a la hora de elegir cuál utilizar o tener versiones de fragmentación.

Bibliografía

Sitios Web: http://lunar.cs.byu.edu/papers.html

http://www.cs.ualberta.ca/~database/ddbook/notes/Design/

http://citeseer.nj.nec.com/navathe95mixed.html

http://www.cs.cinvestav.mx/SC/prof_personal/adiaz/Disdb/Cap_3.html

http://www.ittc.ukans.edu/~sgauch/647/s00/notes/Chxx2a/sld019.htm

Anexo 1

Una herramienta de ayuda al diseño de bases de datos distribuidas debe almacenar información detallada de las características de cada base datos que es objeto de diseño. El catálogo de almacenamiento es una base de datos relacional, que no está disponible directamente para el uso de los usuarios de las herramientas, sino que se accede a ella indirectamente a través de los asistentes. Las tablas que lo componen son usadas internamente, compartiendo toda la información relevante para el proceso de diseño de bases de datos distribuidas. Al crear cada nuevo proyecto de diseño, en el asistente para el esquema conceptual global, se crea automáticamente un nuevo catálogo.

El catálogo del sistema colecciona la información sobre:

- Las características de diseño de la bases de datos.
- Las aplicaciones que usan a la base de datos para dar solución a los casos de uso o solicitudes de usuarios.
- Los sitios que intervendrán en el alojamiento de los datos y el procesamiento de las solicitudes.
- La red que soporta la conexión entre sitios y el intercambio entre estos a fin de efectuar sus funciones.

El esquema relacional del catálogo, incluyendo sus atributos es:

Schema (SchemaName, Cardinality)

GlobalAttributes (SchemaName, IdAtt, AttName, IdType, Pkey, Fkey)

Sites (SiteName, IP, ProcessorType, ProcessorClock, TotalDiskSpace,
FreeDiskSPace, ReadAccessTime, WriteAccessTime, MaxAnswerTime)

Applications (IdApp, AppName, MaxAnswerTime)

App_Schema (IdApp, SchemaName, Selectivity)

App_Attributes (IdApp, SchemaName, IdAtt, OpRead, OpWrite)

App_Site (IdApp, SiteName, ActivationFreq)

Predicates (IdApp, SchemaName, IdAtt, IdOperator, AttValue)

Links (SchemaMemberName, IdAtt, SchemaOwnerName)

Operations (IdOperation, SchemaName, IdPreviousOp, IdOpType, Expression)

Fragments (IdLogFragm, IdOperation, FragmentName, Cardinality)

AccessTime (FromStation, ToStation, AverageTime)

PhysicalFragment (IdPhysicalFragment, SiteName, PhysicalFragmName, IdLogFragm)

FragmentAttributes(IdFragmAtt, IdLogFragm, IdGlobalAtt, SchemataName, IdDataType, FragAttName)

DataType(IdType, Type, Size)